TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO.

Objetivos

JavaScript (JS) es un lenguaje de programación cuyo uso principal ha venido siendo dotar de dinamismo, rapidez y efectos atractivos a las páginas web, mediante su uso combinado junto a HTML, CSS y otros lenguajes. Este curso permite aprender los fundamentos de JavaScript, imprescindible para trabajar con páginas web hoy día.

Destinatarios

Personas que ya tengan unos conocimientos mínimos sobre creación de páginas web (HTML y CSS) y sobre programación. Para realizar este curso de JavaScript, debes tener conocimientos previos de HTML y CSS, así como de fundamentos de programación (haber programado antes). Se recomienda haber realizado los cursos "Tutorial básico del programador web" sobre HTML desde cero y CSS desde cero, de aprenderaprogramar.com, antes de seguir este curso.

Contenidos

- ✓ INTRODUCCIÓN A JAVASCRIPT. QUÉ ES Y PARA QUÉ SIRVE JAVASCRIPT. VERSIONES. EL ECMA.
- ✓ INSERTAR JAVASCRIPT EN WEBS. EN LÍNEA, INTERNO Y EXTERNO.
- ✓ CREANDO SCRIPTS BÁSICOS. COMENTARIOS, VARIABLES, OPERADORES, CONTROL DEL FLUJO.
- ✓ FUNCIONES JAVASCRIPT. CREACIÓN E INVOCACIÓN. PARÁMETROS Y RETORNO.
- ✓ OBJETOS EN JAVASCRIPT. ¿QUÉ SON? TRABAJAR CON ARRAYS, DATE, IMAGE, STRING, MATH.
- ✓ DOM. JERARQUÍA DE OBJETOS DEL NAVEGADOR. WINDOW, DOCUMENT.
- ✓ GESTIÓN DE EVENTOS CON JAVASCRIPT. TIPOS DE EVENTOS. MANEJADORES DE EVENTOS.
- ✓ APLICANDO JAVASCRIPT: CSS, FORMULARIOS, CALENDARIOS, RELOJES, MENÚS, GALERÍAS…
- ✓ JAVASCRIPT AVANZADO. APIS, LIBRERÍAS. JQUERY. FIREBUG Y DEPURACIÓN DE CÓDIGO.

Duración

150 horas de dedicación efectiva, incluyendo lecturas, estudio y ejercicios.

Dirección, modalidades y certificados

El curso está dirigido por César Krall, Responsable del Departamento de Producción de aprenderaprogramar.com del portal web aprenderaprogramar.com. Se oferta bajo la modalidad web (gratuito).

APRENDERAPROGRAMAR.COM

INDICE DEL CURSO "TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO"(CU01101E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº1 del curso Tutorial básico del programador web: JavaScript desde cero. Indice del curso.

Autor: César Krall

INDICE DEL CURSO

5.5. Añadir y borrar elementos de arrays.
5.6. Arrays unidimensionales y arrays multidimensionales. Recorrido de arrays.
5.7. Métodos de arrays. Comprobar si un objeto es tipo array.
5.8. La clase Date. Trabajar con fechas en JavaScript.
5.9. Trabajar con imágenes en JavaScript.
5.10. La clase Math JavaScript. Trabajar con funciones matemáticas.

6. DOM Y JERARQUÍA DE OBJETOS.

6.1. ¿Qué es el DOM? Jerarquía de objetos en páginas web.
6.2. Del documento HTML a un árbol de nodos. Tipos de nodos.
6.3. Selección de elementos en un documento. Atributos y contenido de elementos.
6.4. Crear, insertar y borrar nodos con JavaScript.
6.5. El objeto Window. Manejo del tiempo (timers). Trabajo con múltiples ventanas y marcos.
6.6. Document. Elementos de document como propiedades de Window.

7. GESTIÓN DE EVENTOS CON JAVASCRIPT.

7.1. Tipos de eventos. Manejadores de eventos o event handlers.
7.2. Eventos en la carga del documento (Load).
7.3. Eventos generados por el ratón (mouse).
7.4. Eventos arrastrar y soltar (drag and drop).
7.5. Eventos generados por el teclado.
7.6. Información generada por los eventos y su manejo.

8. APLICANDO JAVASCRIPT EN DESARROLLOS WEB.

8.1. Manejo de CSS a través de JavaScript.
8.2. JavaScript para la gestión de formularios web. Usos en validación de formularios.
8.3. Calendarios con JavaScript.
8.4. Utilidades tipo reloj, cronómetro, cuenta atrás, etc. con JavaScript.
8.5. Efectos para menús basados en JavaScript: desplegado dropdown.
8.6. Rotación de imágenes y galerías de imágenes basadas en JavaScript.

9. JAVASCRIPT AVANZADO.

9.1. APIS HTML. Api Canvas. Api Drag and Drop. Api Geolocation. Api Storage. Api File.
9.2. Librerías JavaScript. Una visión general. jQuery, Mootools, Prototype, Modernizr, Google API…
9.3. Adentrándonos en jQuery. Conceptos básicos y aplicaciones básicas.
9.4. Depuración de errores JavaScript. Herramientas. Firebug.

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ORIENTACIÓN SOBRE EL CURSO "TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO" (CU01102E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº2 del Tutorial básico del programador web: JavaScript desde cero. Orientación sobre duración del curso y conocimientos previos recomendados.

Autor: César Krall

ORIENTACIÓN SOBRE EL CURSO PASO A PASO "TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO"

JavaScript es un lenguaje ideado para dotar de dinamismo, rapidez y agilidad a las páginas web. JavaScript puede tener distintas aplicaciones, pero la más común es la de ser un lenguaje de programación que se ejecuta del lado del cliente. Por si alguien se está preguntando qué es un cliente…

Para quienes estén menos habituados a la terminología cliente – servidor, podemos decir simplificadamente que un servidor es un computador con capacidades especiales para atender las peticiones de muchos otros computadores. Normalmente los servidores están situados en centros de datos gestionados por empresas. Por el contrario, un cliente sería un computador u ordenador personal con el que trabajamos en nuestra casa u oficina. Esto no siempre es así, pero nos sirve para situarnos de forma introductoria en el contexto de los desarrollos web y del lenguaje JavaScript.

A veces se dice que JavaScript no es un lenguaje de programación propiamente dicho, sino un lenguaje de script. Un script vendría siendo un programa normalmente simple, almacenado en un archivo de texto, y que es ejecutado por un intérprete del lenguaje para conseguir unos resultados. Pero muchos estudiosos no se ponen de acuerdo sobre qué es exactamente un lenguaje de script y cuáles son lenguajes de script y cuáles no lo son, por lo que nosotros no vamos a interesarnos demasiado por esta terminología.

Para tener un enfoque global de lo que es JavaScript dividiremos los lenguajes que intervienen en los desarrollos web en estas categorías:

a) **Metalenguajes:** incluiríamos aquí HTML y CSS. No proveen la potencia que aporta la programación.

b) **Lenguajes de programación de propósito general:** incluiríamos aquí lenguajes como C, C++, Visual Basic, PHP, etc., creados con el objetivo amplio de crear aplicaciones informáticas de todo tipo. Proporcionan toda la potencia de la programación.

c) **Lenguajes de programación de propósito específico:** son lenguajes con toda la potencia de la programación pero orientados a una tarea concreta específica. Aquí incluiríamos JavaScript, como lenguaje que ha sido creado para facilitar la operación rápida en páginas web.

En los desarrollos web JavaScript se mezcla de alguna manera con HTML y CSS o con lenguajes de programación como PHP. Esta mezcolanza implica que a veces el código JavaScript esté junto al HTML o PHP, o que a la hora de desarrollar una web o solucionar un problema en una web sean necesarios conocimientos de HTML, CSS, JavaScript y de un lenguaje de programación de propósito general.

JavaScript es un lenguaje de programación completo, con una sintaxis, conjunto de sentencias e instrucciones similares a las de otros lenguajes. Como peculiaridades tenemos que JavaScript es un lenguaje normalmente interpretado por el navegador web y por tanto podremos obtener en ocasiones resultados diferentes al pasar de un navegador a otro.

Este curso que estamos comenzando va dirigido a aquellas personas que quieran adquirir unos fundamentos básicos para utilizar JavaScript dentro de desarrollos web con vistas a poder desarrollar en el futuro páginas web atractivas y de cierta complejidad. No vamos a desarrollar un manual de referencia de JavaScript, sino un curso básico paso a paso. No vamos a contemplar todos los aspectos del lenguaje, sino aquellos que consideramos básicos desde el punto de vista didáctico, con vistas a que posteriormente la persona que lo desee amplíe sus conocimientos. Nuestro objetivo es ser **claros, sencillos y breves**, y para eso tenemos que centrarnos en determinadas cuestiones de JavaScript y dejar de lado otras.

Como conocimientos previos para iniciar este curso recomendamos estos (seguir la recomendación o no queda a criterio del alumno y/o profesor que vayan a seguir el curso): Ofimática básica (saber copiar, pegar, mover y abrir archivos. Uso de un editor de textos, etc.) y haber realizado los siguientes cursos (o tener los conocimientos que se aportan en ellos):

a) El Curso básico de HTML que se ofrece en aprenderaprogramar.com. Su URL es la siguiente: http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=75&Itemid=203

b) El curso básico de CSS que se ofrece en aprenderaprogramar.com. Su URL es la siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=75&Itemid=203

Se recomienda también conocer algún lenguaje de programación, de modo que a la persona que siga el curso le resulten familiares los conceptos de variable, arrays o arreglos unidimensionales y multidimensionales, condicionales, bucles, algoritmos, etc. **Si no tienes ningún conocimiento de programación** te recomendamos realizar el curso "Programación en Visual Basic nivel I" de aprenderaprogramar.com como forma de familiarizarte con estos conceptos que te van a resultar necesarios para seguir este curso. La URL del curso de Visual Basic es la siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=37&Itemid=61

Los conocimientos previos son, como hemos dicho, deseables pero no imprescindibles.

Aprender JavaScript requiere tiempo y esfuerzo. Para hacer ese recorrido más llevadero, te recomendamos que utilices los foros de aprenderaprogramar.com, herramienta a disposición de todos los usuarios de la web (http://www.aprenderaprogramar.com/foros/), que te servirán para consultar dudas y recabar orientación sobre cómo enfrentarte a los contenidos. Entre los miembros del portal web y otros usuarios, trataremos de ayudarte para que el estudio te sea más llevadero y seas capaz de adquirir los conocimientos necesarios y avanzar como programador o diseñador web.

El tiempo necesario (orientativamente) para completar el curso incluyendo prácticas con ordenador, suponiendo que se cuenta con los conocimientos previos necesarios, se estima en 150 horas de

dedicación efectiva o aproximadamente tres meses con una dedicación de 2,50 horas diarias de lunes a viernes. Aprender a crear páginas web requiere dedicación y esfuerzo.

El curso ha sido generado paso a paso usando Windows como sistema operativo y por ello contiene algunas indicaciones específicas para usuarios de Windows, pero también puede ser utilizado en otros entornos (Linux, Macintosh, etc.), ya que los desarrollos web no son dependientes del sistema operativo con el que trabajemos en nuestro computador.

Estamos seguros de que con tu esfuerzo y la ayuda que te podamos brindar este curso te resultará de gran utilidad.

Próxima entrega: CU01103D

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

¿QUÉ ES JAVASCRIPT? PRINCIPALES USOS. SERVIDOR Y CLIENTE. HTML, CSS Y PROGRAMACIÓN. EFECTOS DINÁMICOS (CU01103E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº3 del Tutorial básico “JavaScript desde cero”.

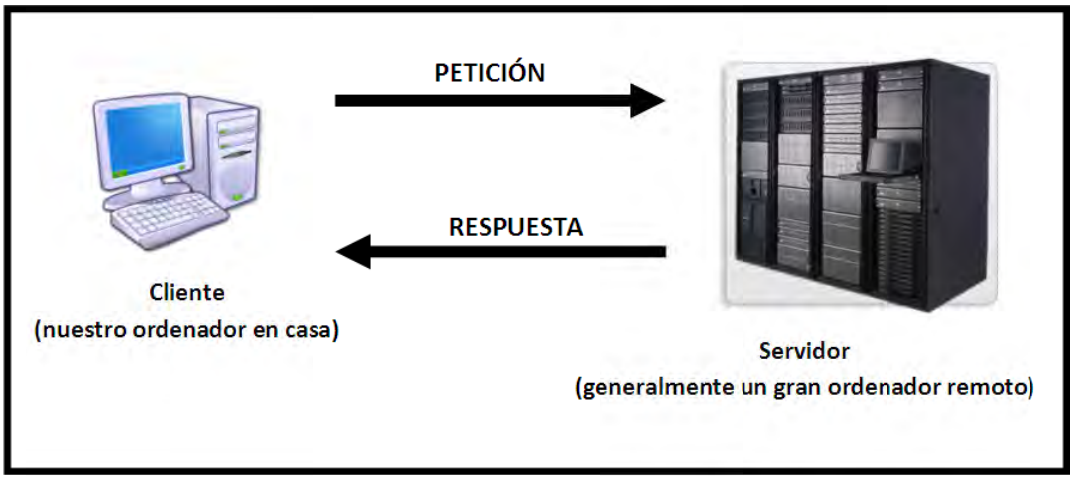
Autor: César Krall

QUÉ ES JAVASCRIPT

JavaScript es un lenguaje utilizado para dotar de efectos y procesos dinámicos e "inteligentes" a documentos HTML. Un documento HTML viene siendo coloquialmente "una página web". Así, podemos decir que el lenguaje JavaScript sirve para ejecutar acciones rápidas y efectos animados en las páginas web. Las acciones controladas por JavaScript pueden ser el despliegue de un menú, hacer aparecer, desaparecer o cambiar texto e imágenes, realizar cálculos y mostrar resultados, mostrar mensajes de aviso (por ejemplo si faltan datos en un formulario) y "efectos animados" en general.

Este lenguaje es principalmente utilizado por parte de programadores web para dar respuestas rápidas a las acciones del usuario sin necesidad de enviar la información de lo que ha hecho el usuario al servidor y esperar respuesta de éste (lo que haría más lento los procesos). El código JavaScript se carga al mismo tiempo que el código HTML en el navegador, y reside en el cliente (computador en el que nos encontramos), por lo que JavaScript sigue funcionando incluso aunque se produzca un corte en la conexión a internet (en este caso no podremos seguir navegando hacia otras direcciones web, pero sí podremos ejecutar procesos "locales" en nuestro computador para la página web en que nos encontráramos).

En el siguiente esquema vemos un esquema básico de lo que supone navegar por internet desde un computador personal.



El proceso básico es el envío de una petición (que puede llevar incorporada información como los datos de un formulario) a un servidor, esperar respuesta por parte del servidor y recibir la respuesta en nuestro computador. Cada proceso de este tipo consume tiempo, el tiempo total podríamos verlo desde el lado de nuestro computador como Tiempo Total Proceso = tiempo envío petición + tiempo procesamiento petición + tiempo recepción respuesta.

Aún con velocidades rápidas de navegación cuantos más procesos de este tipo realicemos más lenta será la navegación web. JavaScript podemos decir que supone que las respuestas del servidor sean más completas y permite que se realicen más procesos en nuestro computador (aquellos procesos que realmente pueden ser resueltos en nuestro propio computador sin necesidad de estar enviando peticiones al servidor), de modo que se reduce el número de peticiones y respuestas necesarias entre cliente y servidor.

El código JavaScript es interpretado directamente por el navegador web, sin necesidad de otros programas o procesos intermedios. Un ejemplo puede ayudarnos a comprender la idea. Supongamos que en una página web pedimos al usuario que rellene un formulario con sus datos personales, y que entre los requisitos para enviar el formulario tenemos que es obligatorio que se incluya el nombre de usuario y correo electrónico, siendo obligatorio que el nombre tenga más de una letra y que el correo electrónico contenga el carácter @ (arroba).

Supongamos que una petición y respuesta de servidor requiere de un tiempo de 2 segundos y comprobemos qué ocurriría con el control del proceso del lado del servidor o controlándolo del lado del cliente con JavaScript. Como JavaScript está en el propio computador del usuario (cliente), suponemos que los tiempos de respuesta implican 0 s de consumo de tiempo, es decir, la respuesta es inmediata.

CONTROL DE PROCESO DEL LADO DEL SERVIDOR

Paso	Acción del usuario	Respuesta	Tiempo navegación
1	Envía un formulario donde por error el nombre está en blanco y el correo no contiene el carácter arroba	Servidor informa de que los datos en el formulario no son válidos	2 s
2	Corrige el nombre pero se olvida de corregir el correo electrónico	Servidor informa de que los datos en el formulario no son válidos	4 s
3	Corrige el correo y todo está ok	Servidor informa que el formulario ha sido enviado correctamente	6 s

CONTROL DE PROCESO DEL LADO DEL CLIENTE

Paso	Acción del usuario	Respuesta	Tiempo navegación
1	Envía un formulario donde por error el nombre está en blanco y el correo no contiene el carácter arroba	Cliente detecta error e informa de que los datos en el formulario no son válidos	0 s
2	Corrige el nombre pero se olvida de corregir el correo electrónico	Cliente detecta error e informa de que los datos en el formulario no son válidos	0 s
3	Corrige el correo y todo está ok	Servidor informa que el formulario ha sido enviado correctamente	2 s

Aquí comprobamos cómo una ventaja importante de JavaScript es hacer más ágil y dinámica la navegación por páginas web, evitando los tiempos de espera.

¿Significa esto que podemos hacer todo mediante JavaScript? Algunas páginas web pueden basarse en combinaciones de HTML con CSS y JavaScript. Incluso una página web podría ser sólo HTML sin CSS ni JavaScript, pero en los desarrollos profesionales lo normal es que intervengan estos tres elementos junto a un lenguaje del lado del servidor (código que reside en el servidor y no en el cliente).

Hay varios motivos por lo que en los desarrollos web profesionales se combinan procesos del lado del cliente con procesos del lado del servidor. Vamos a citar algunos y para ello nos valdremos del ejemplo de una tienda de comercio electrónico.

a) Los datos en la web cambian con frecuencia. Para que los datos se mantengan actualizados es necesario refrescar la información haciendo nuevas peticiones al servidor. Para que el usuario vaya navegando por la tienda quizás podamos enviar los datos de 10 ó 12 productos pero para cargar nuevos productos será lógico hacer una nueva petición al servidor.

b) Los datos pueden sobrecargar el computador del usuario. Si tenemos una tienda con 7.000 productos y enviáramos todos los datos al computador del usuario para que fueran gestionados mediante JavaScript tendríamos problemas. En primer lugar, el envío de volúmenes muy grandes de información consume mucho tiempo (y posiblemente el usuario se vaya a otra tienda si lo hacemos esperar demasiado). En segundo lugar, el computador del usuario puede tener problemas para gestionar volúmenes demasiado grandes de información (sobrecarga). Los volúmenes grandes de información normalmente residen en bases de datos gestionadas por el servidor y los datos son servidos en pequeños paquetes de datos a medida que resulta necesario.

c) Hay procesos que tienen que ser realizados del lado del servidor porque necesitan de verificaciones de seguridad que no pueden residir en el computador de un usuario. Por ejemplo, para el pago con una tarjeta de crédito es necesario que el usuario envíe el número de su tarjeta de crédito al servidor y que éste mediante un proceso seguro verifique la tarjeta y el pago. Sería disparatado pensar en enviar los números de tarjetas de crédito válidas al computador del usuario y que el proceso tuviera lugar en el computador cliente para luego informar al servidor de que el pago es correcto.

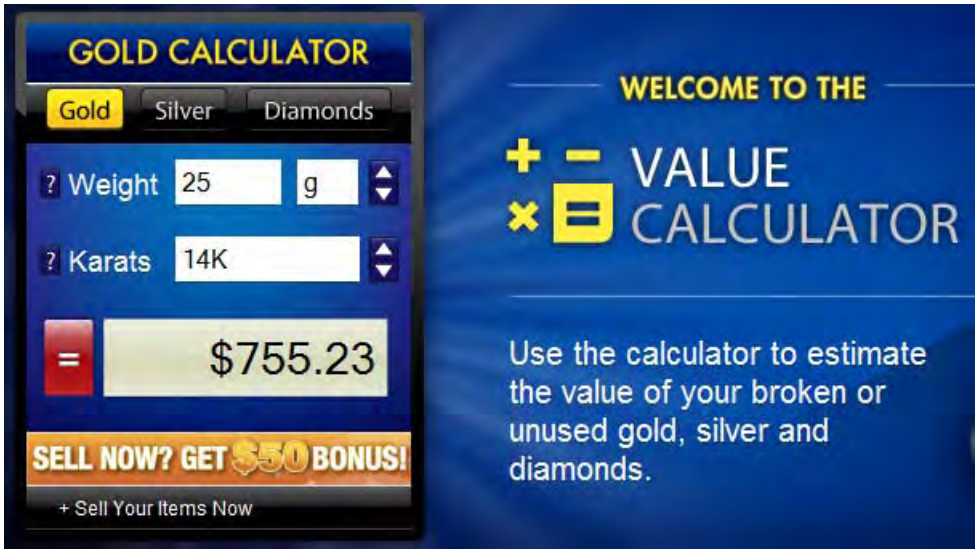
Conforme vayamos adquiriendo experiencia como programadores nos daremos cuenta de que hay procesos que claramente es más adecuado realizarlos del lado del servidor, otros que claramente es más adecuado realizarlos del lado del cliente, y otros que podrían realizarse tanto del lado del servidor como del lado del cliente. También con la experiencia iremos aprendiendo a tomar decisiones relacionadas con esto.

En este esquema vemos cómo se combina un lenguaje del lado del servidor con JavaScript y HTML (no citamos CSS pero obviamente CSS debe incluirse también en la respuesta del servidor).

Aquí hemos indicado como lenguaje del lado del servidor PHP, pero podría ser cualquier otro como ASP, JSP, etc.

Algunos usos típicos de JavaScript son:

- Despliegue de menús
- Galerías de imágenes que van rotando automáticamente
- Relojes
- Contadores de tiempo hacia delante o hacia detrás (cuenta atrás)
- Cronómetros
- Calculadoras
- Cambiar la apariencia de la página web cuando el usuario hace click en un botón o imagen. Por ejemplo, los estilos CSS pueden variar al pulsar un botón y así permitir que el usuario personalice la apariencia de una página web.
- Ofrecer distintos tipos de respuesta según el navegador y sistema operativo que esté utilizando el usuario.
- Validación de datos en formularios (impedir envío de formularios con datos erróneos o incompletos y mostrar mensajes de aviso).
- Ejecutar cálculos o pequeños programas del lado del cliente.
- Modificar código HTML en respuesta a la acción del usuario sin necesidad de establecer conexión con el servidor.



Ejemplo: Calculadora en una página web para calcular el valor de una pieza de oro creada con JavaScript.

En este curso vamos a centrarnos en aprender las bases fundamentales para usar JavaScript del lado del cliente. No vamos a entrar a detallar todas las instrucciones y detalles del lenguaje, ni las diferencias entre versiones, porque lo que nos interesará será comprender cuál es la filosofía y cómo podemos sacarle partido a una herramienta muy potente como JavaScript. Para hacer el curso didáctico, tenemos que centrarnos en lo fundamental y dejar los detalles de lado. Para aquellas personas que lo deseen, daremos referencias de cómo encontrar la especificación oficial del lenguaje donde se podrán consultar detalles específicos.

Debido a lo indicado anteriormente, para seguir este curso son necesarios como conocimientos previos:

- HTML
- CSS
- Fundamentos de la programación (haber programado en algún lenguaje y conocer conceptos como variable, array o arreglo, condicionales tipo if y bucles tipo for).

Para seguir el curso puedes utilizar como herramienta de apoyo los foros aprenderaprogramar.com, disponibles en http://aprenderaprogramar.com/foros, donde puedes plantear consultas y dudas relativas al contenido del curso.

Próxima entrega: CU01104E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT: LENGUAJE DE PROGRAMACIÓN WEB . JAVASCRIPT DEL LADO DEL SERVIDOR. NODE.JS, JAXER, RINGOJS, ETC. (CU01104E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº4 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

JAVASCRIPT EN EL MARCO DE LA PROGRAMACIÓN WEB

Ya hemos comentado que JavaScript se integra dentro de los desarrollos web con otros lenguajes como HTML, CSS y generalmente con un lenguaje del lado del servidor como PHP, JSP ó ASP u otros. Hemos dicho también que su uso principal es ejecutarse del lado del cliente (computador del usuario). En esta entrega del curso vamos a ver cómo se integra JavaScript dentro de los desarrollos web en general y cómo existen posibilidades para usar JavaScript no sólo del lado del cliente.

Si hiciéramos un símil entre una orquesta y un desarrollo web, el director de orquesta sería el lenguaje del lado del cliente (por ejemplo PHP), que actúa como “cerebro” controlador de la página web. CSS sería el encargado de vestuario y maquillaje, es decir, quien controla la apariencia de lo que se presenta al público. HTML sería la estructura: el encargado de cómo se distribuyen las sillas en el escenario, dónde se coloca cada músico y qué instrumento lleva cada músico. Supongamos que para distintas canciones (urls) los músicos cambian de posición (cambia la estructura HTML). JavaScript sería el encargado de efectos especiales, por ejemplo quien mueve las luces y dispara un cañón de confetis en un momento dado, una persona ágil y rápida. Por último tenemos a la base de datos, un encargado de guardar todas las partituras (información) y de ir entregándolas a medida que se lo requieren los músicos u otros encargados.

Esto es sólo un símil, pero nos sirve de introducción para contextualizar el papel habitual de JavaScript dentro de los desarrollos web.

Una orquesta puede tocar sin director (sin lenguaje del lado del servidor), sin encargado de efectos especiales (JavaScript), sin encargado de vestuario y presentación (CSS), pero no sin músicos (HTML). De todas formas, una buena orquesta normalmente tendrá todo el personal (recursos de programación) necesarios para que su puesta en escena sea “brillante”.

Un lenguaje de programación es un lenguaje que se usa para realizar procesos de interés a través de un ordenador o dispositivo electrónico, desde un cálculo para un estudiante o ingeniero, a una compra por internet, pasando por cualquier cosa que se te ocurra. Un lenguaje de programación tiene como características básicas el tener la capacidad para “tomar decisiones” o ejecutar un proceso u otro en función de las circunstancias (por ejemplo dependiendo del botón que pulse el usuario), así como el ser capaz de repetir procesos numerosas veces hasta que se cumpla una condición. JavaScript es un lenguaje que permite cumplir estas funciones, por tanto es un lenguaje de programación, aunque se use junto a otros lenguajes de programación y lenguajes de etiquetas como HTML y CSS.

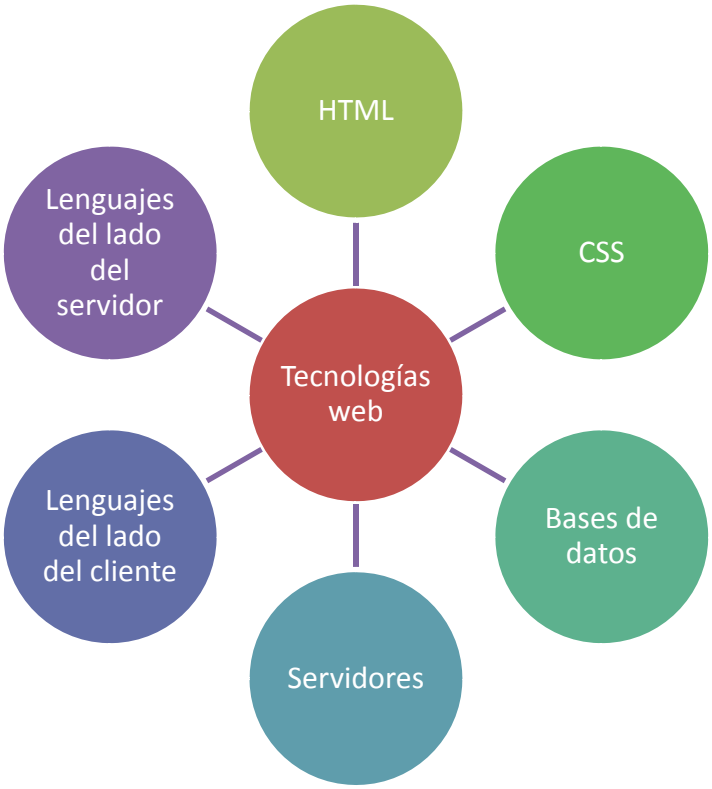
JavaScript es un lenguaje que apareció para hacer más fáciles de programar y más fáciles para navegar los desarrollos web. Un desarrollo web comprende múltiples áreas de conocimiento:

En la clasificación que hemos hecho, JavaScript estaría englobado dentro del área de programación.

Los desarrollos web tienen dimensiones muy variables. Podemos hablar desde una pequeña página web para una empresa local hasta un gran portal para una empresa de ámbito internacional. En ambos casos podríamos decir que interviene la programación web y el diseño web. Sin embargo, un pequeño desarrollo puede ser llevado a cabo por una sola persona que abarque tanto programación como diseño, mientras que un gran desarrollo requiere de un equipo de trabajo más o menos amplio y con distintos especialistas, ya que en torno a los desarrollos web hay diferentes áreas de conocimiento implicadas (análisis, diseño, programación, sistemas, integración, testing, etc.).

En un gran desarrollo existen personas especializadas en las distintas áreas, de modo que el programador no suele trabajar en el diseño (excepto para hacer algún retoque o cambio, o para solucionar problemas). No obstante, sí resulta conveniente que un programador web tenga los conocimientos suficientes de HTML y CSS ya que le resultarán útiles y necesarios, por un lado para la solución de problemas y por otro para integrar cuestiones donde el diseño y la programación se entremezclan. Por ejemplo, podremos hacer modificaciones rápidas del aspecto de una página web, cambiar un color de fondo o imagen, etc. cuando el usuario pulse un botón, mezclando JavaScript con CSS.

Si miramos a los lenguajes o tecnologías que hay en torno a los desarrollos web podríamos hacer una clasificación que comprende: HTML, CSS, Bases de datos, Servidores, Lenguajes de programación del lado del cliente (p.ej. JavaScript) y Lenguajes de programación del lado del servidor (p.ej. PHP). Aquí nos estamos refiriendo a paradigmas o situaciones más frecuentes. Como comentaremos más adelante, JavaScript también se está usando en algunos casos como lenguaje de programación del lado del servidor.



JavaScript es una tecnología (o lenguaje) aceptada por todos los navegadores y que interviene en prácticamente todo desarrollo web, grande o pequeño. Se encarga de dotar de respuesta rápida y efectos controlados directamente desde el computador del usuario a las páginas web.

Los lenguajes de programación del lado del servidor realizan procesos en el servidor (computador remoto que se encarga de enviar las páginas web a través de internet): podemos citar entre estos lenguajes Java (JSP), ASP.NET, o PHP entre los principales.

Los lenguajes de programación del lado del cliente realizan procesos en el ordenador personal del usuario (efectos visuales, cálculos, etc.): podemos citar entre estos lenguajes JavaScript, Java (applets), VBScript ó Dart (impulsado por Google), entre los principales. JavaScript es el lenguaje de programación del lado del cliente más utilizado hoy día en los desarrollos web y es aceptado por todos los navegadores.

En cuanto a bases de datos podemos nombrar MySQL, SQLServer y Oracle, entre las principales.

Las tecnologías se combinan entre ellas de muy diversas maneras. Podemos citar algunas combinaciones bastante habituales entre lenguajes de programación y bases de datos: Java + Oracle, ASP.NET + SQLServer, PHP + MySQL. Sea cual sea la combinación utilizada, en un desarrollo web moderno siempre intervendrá HTML, CSS y JavaScript.

En resumen, JavaScript es un lenguaje de programación del lado del cliente cuyos aspectos básicos deben ser conocidos por los programadores web. En la práctica, muchas veces se entremezcla el código de programación del lado del servidor o del lado del cliente con el código HTML y código CSS, de ahí

que coloquialmente se hable de “programación web” para referirse a todo este conjunto, aunque formalmente ni HTML ni CSS son lenguajes de programación.

Fíjate que estamos tratando de dejar claro qué es y para qué sirve JavaScript antes de empezar a estudiar este lenguaje porque si tenemos los conceptos claros nos será mucho más sencillo el aprendizaje, ahorraremos tiempo y cometeremos menos errores.

JAVASCRIPT DEL LADO DEL SERVIDOR

Hemos comentado que JavaScript es un lenguaje que aparece en prácticamente todo desarrollo web como lenguaje del lado del cliente. No obstante, hace ya varios años que ha empezado a tener uso en algunos desarrollos web como lenguaje del lado del servidor para permitir desarrollos web completos.

El desarrollo en JavaScript del lado del servidor se hace instalando en el servidor herramientas que permiten el uso de JavaScript del lado del servidor como:

- **Node.js:** es quizás la herramienta más utilizada dentro de los desarrollos que usan JavaScript del lado del servidor. Node.js es un entorno de programación que incluye numerosas librerías preparadas para ser usadas por parte de los programadores.
- **Jaxer**
- **RingoJS**
- **EJScript**
- **AppengineJS**

El desarrollo con JavaScript del lado del servidor permite crear desarrollos web completos, pero es una práctica que todavía no está generalizada y en cierta medida se sigue considerando no habitual, o al menos no recomendable para quienes no conozcan JavaScript básico.

Nosotros en este curso no estudiaremos ninguna aplicación de JavaScript del lado del servidor: nos limitaremos a estudiar los fundamentos del JavaScript “tradicional”, JavaScript del lado del cliente. Entendemos que JavaScript del lado del servidor debe ser materia de estudio en cursos más avanzados, cuando ya se tengan unas bases sólidas de JavaScript del lado del cliente y de otras tecnologías relacionadas con los desarrollos web.

Próxima entrega: CU01105E

APRENDERAPROGRAMAR.COM

DIFERENCIAS ENTRE JAVASCRIPT Y JAVA, HTML, CSS, PHP, ETC. FRONTERA ENTRE LENGUAJES EN DESARROLLOS WEB (CU01105E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº5 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DIFERENCIAS ENTRE JAVA Y JAVASCRIPT

Como hemos visto, JavaScript es un lenguaje de programación para desarrollos web que normalmente usaremos del lado del cliente (es decir, que se ejecuta en el computador personal del usuario). Muchas veces nos han planteado la pregunta: ¿es JavaScript y Java lo mismo? ¿Por qué tienen nombres tan similares?

Java y JavaScript son dos lenguajes de programación con un nombre parecido y algunas similitudes en algunos aspectos, pero que en el fondo son dos lenguajes completamente independientes y completamente distintos. Una persona puede saber mucho de JavaScript y no tener “ni idea” de Java, y al revés.

Si son distintos, ¿por qué tienen un nombre tan parecido? La causa de esta similitud en los nombres se debe a el origen de JavaScript. Inicialmente fue desarrollado por la empresa Netscape en 1995 con el nombre de LiveScript. Posteriormente pasó a llamarse JavaScript quizás tratando de aprovechar que Java era un lenguaje de programación de gran popularidad y que un nombre similar podía hacer que el nuevo lenguaje fuera atractivo. Pero salvando algunas similitudes, ambos lenguajes son bien distintos. Su principal parecido podemos decir que es el nombre y algunos aspectos de sintaxis, ya que su finalidad y filosofía son muy distintos.

A continuación indicamos algunas similitudes y diferencias entre uno y otro lenguaje:

Java	JavaScript
Es un lenguaje de programación de propósito general, utilizado tanto en aplicaciones tradicionales de computadores como en desarrollos web.	Es un lenguaje de programación de propósito específico (desarrollos web)
Es un lenguaje que requiere de compilación (traducción previa a código máquina antes de ser ejecutado).	Es un lenguaje que no requiere de compilación al ser interpretado directamente por los navegadores.
Es un lenguaje que se puede considerar pesado, potente y robusto en el sentido de que permite hacer de todo con un gran control.	Es un lenguaje que se puede considerar ligero, ágil y poco robusto en el sentido de que no permite hacer todo lo que permiten otros lenguajes.
Es un lenguaje bajo la filosofía o paradigma de orientación a objetos completamente.	Es un lenguaje no clasificable bajo un paradigma concreto y admite algunas formas de programación no admitidas por Java.
Se puede utilizar tanto del lado del servidor como del lado del cliente. Tiene su uso principal del lado del servidor.	Se puede utilizar tanto del lado del servidor como del lado del cliente. Tiene su uso principal del lado del cliente.

Java	JavaScript
Su sintaxis está inspirada en la sintaxis del lenguaje de programación C	Su sintaxis también está inspirada en la sintaxis del lenguaje de programación C
Requiere de un kit de desarrollo y máquina virtual Java para poder programar en él.	No requiere nada específico para poder programar en él (únicamente un navegador web para ver los resultados y un editor de texto para escribir el código).
Es un lenguaje fuertemente tipado: las variables tienen un tipo declarado y no pueden cambiar el tipo de contenido que almacenan.	Es un lenguaje débilmente tipado: las variables pueden no ser de un tipo específico y cambiar el tipo de contenido que almacenan.
Es un lenguaje con más capacidades y más complejo que JavaScript.	Es un lenguaje con menos capacidades y menos complejo que Java.

Si estás dudando si te conviene estudiar Java o estudiar JavaScript te recomendamos lo siguiente:

- Si quieres orientarte específicamente a los desarrollos web estudia JavaScript.
- Si quieres formarte integralmente como programador y no conoces ningún lenguaje orientado a objetos, estudia primero Java y luego JavaScript.

En este curso nos centramos en el estudio de JavaScript en el contexto de los desarrollos web. Si estás interesado en un curso de Java puedes acceder a él en esta dirección web:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188

JAVASCRIPT Y HTML, CSS, PHP…

A veces nos encontraremos que se puede lograr un mismo efecto usando HTML, usando CSS ó usando un lenguaje de programación. ¿Por qué tantas formas para hacer una misma cosa? ¿Dónde está la frontera entre cada lenguaje?

Esta pregunta no es de fácil respuesta. Vamos a ver con un ejemplo lo que puede ocurrir para algo tan sencillo como aplicar algunos efectos a un texto. No obstante, ten en cuenta que este ejemplo relativo a texto podría aplicarse a otros conceptos como bordes, márgenes, animaciones, etc.

El lenguaje HTML permite aplicar algunos efectos visuales al texto. Escribe o copia este código y guárdalo en un archivo de nombre ejemplo1.html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
	<head>
		<title>Ejemplo HTML aprenderaprogramar.com</title>
		<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
	</head>
	<body>
		<p>Negrita: <strong>Quiero aprender a programar</strong></p>
		<p>Itálica:  <i>Quiero aprender a programar</i></p>
		<p>Tachado:  <strike>Quiero aprender a programar</strike></p>
		<p>Color fuente:  <font color ="green">Quiero aprender a programar</font></p>
	</body>
</html>
```

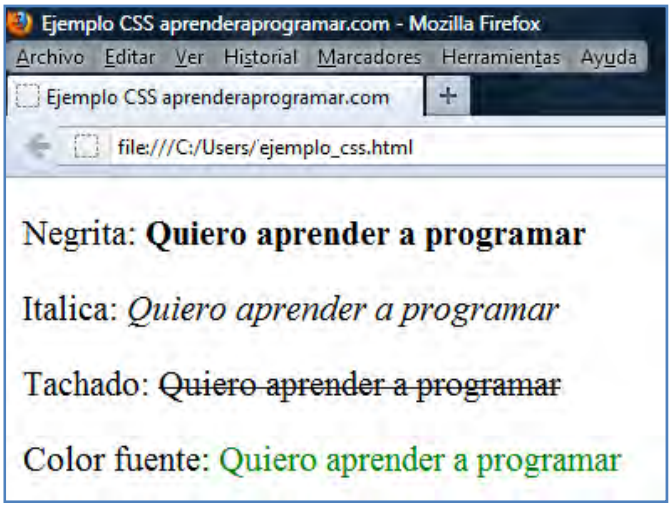
Con Javascript podemos hacer algo parecido. Escribe o copia este código y guárdalo en un archivo de nombre ejemplo2.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
	<head>
		<title>Ejemplo Javascript aprenderaprogramar.com</title>
		<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
	</head>
	<body>
		<script>
		var txt = "Quiero aprender a programar";
		document.write("<p>Negrita: " + txt.bold() + "</p>");
		document.write("<p>Itálica: " + txt.italics() + "</p>");
		document.write("<p>Tachado: " + txt.strike() + "</p>");
		document.write("<p>Color fuente: " + txt.fontcolor("green") + "</p>");
		</script>
	</body>
</html>
```

Y por último en vez de aplicar Javascript o simple HTML, podemos usar CSS. Escribe o copia este código y guárdalo en un archivo de nombre ejemplo3.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
	<head>
		<title>Ejemplo CSS aprenderaprogramar.com</title>
		<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
	<style type="text/css">
		#negrita{font-weight:bold;}
		#italica{font-style:italic;}
		#tachado{text-decoration: line-through;}
		#verde{color:green;}
	</style>
	</head>
	<body>
		<p>Negrita: <span id="negrita">Quiero aprender a programar</span></p>
		<p>Italica: <span id="italica">Quiero aprender a programar</span></p>
		<p>Tachado: <span id="tachado">Quiero aprender a programar</span></p>
		<p>Color fuente:  <span id="verde">Quiero aprender a programar</span></p>
	</body>
</html>
```

Haz doble click sobre cada uno de los archivos para visualizar el resultado en un navegador. El resultado que obtenemos es algo similar a esto:



El único código que debemos entender por el momento es el correspondiente al ejemplo 1 y al ejemplo 3, ya que debemos conocer HTML y CSS para seguir este curso. El código del ejemplo 2 no te preocupes si no lo entiendes ya que el objetivo ahora no es comprender ese código, sino simplemente ver cómo podemos alcanzar un mismo objetivo usando distintos lenguajes como HTML, Javascript ó CSS.

Además si nos fijamos, el código Javascript y el código CSS está dentro de un documento HTML (aunque podrían estar en archivos separados).

Todo esto nos puede llevar a preguntarnos: ¿Por qué se entremezclan unos lenguajes con otros? La respuesta sería histórica y técnica: HTML se convirtió en la forma de crear páginas web, pero tenía muchas limitaciones. En un momento dado, se consideró que entremezclar (embeber) lenguajes entre sí podía ser una buena opción técnica para resolver problemas o hacer cosas que no era posible o conveniente hacer con HTML. Así, podemos embeber Javascript en HTML ó embeber CSS en HTML, o embeber HTML en PHP, etc. Por ello a veces ocurre que no hay una frontera clara entre lenguajes de programación, HTML y CSS. Esto, que puede resultar un tanto confuso inicialmente, se va convirtiendo en "comprensible" a medida que se trabaja y se aprende más sobre estos lenguajes.

Por otro lado, ¿por qué tantas vías distintas para hacer algo cuando quizás que solo hubiera una manera de poner el texto en negrita, o una sola manera de poner un color de fuente, sería más simple?

Para esto podemos citar varios motivos:

a) **Motivos históricos:** a veces las cosas se empezaron a hacer de una manera y luego se pensó que era mejor hacerlas de otra. Sin embargo, para evitar que las páginas web existentes dejaran de funcionar, se siguieron permitiendo formas de hacer las cosas "anticuadas". Por ejemplo la etiqueta <strike> … </strike> en HTML se considera deprecated (obsoleta, de uso no recomendado) en HTML 4.01 y no está admitida en HTML 5. Sin embargo, se sigue usando. Muchas formas de hacer las cosas se admiten aunque no estén recomendadas.

b) **Motivos de independencia de tecnologías:** HTML es una cosa y Javascript es otra, aunque en la práctica nos encontremos con que Javascript se puede "entremezclar" (embeber) en HTML. Podríamos hacer cosas en Javascript y no querer usar otro lenguaje, es decir, podríamos tratar de hacer cosas independientes sin "entremezclar" tecnologías. Por ello lenguajes como Javascript ó PHP incorporan posibilidades para hacer cosas que ya se pueden hacer de otra manera. De esta forma tienen la potencialidad de ser más independientes.

c) **Motivos de políticas de desarrollo y estándares:** quizás no te lo hayas preguntado nunca, pero conviene tener al menos una orientación al respecto: ¿Quién define qué es un lenguaje como HTML, CSS, PHP, cómo se debe escribir, qué resultado debe generar cada etiqueta o instrucción, etc.? En general detrás de los lenguajes, aunque algunos fueron creados por personas individuales, hay empresas, comunidades de desarrollo, asociaciones, consorcios internacionales, comités, etc. En ocasiones un grupo de personas no está de acuerdo con la forma en que se está creando una especificación del lenguaje y forman grupos de trabajo alternativos que definen estándares alternativos. A veces triunfa un estándar y el otro se desecha, pero otras veces conviven distintos estándares que permiten hacer las cosas de distintas maneras. Para los creadores de páginas web esto resulta negativo, pero ¡así es la vida!

d) **Otros motivos:** podríamos abundar en el por qué de que las cosas sean como son, pero con tener una idea general nos basta.

Acostúmbrate a pensar que los desarrollos web no son matemáticas. Las cosas se pueden hacer de muchas maneras, y de hecho muchas veces se hacen "de mala manera" por desconocimiento, por prisas, o por ser más fácil.

Acostúmbrate a pensar que los desarrollos web usan distintos lenguajes que muchas veces se entremezclan entre sí hasta el punto de ser difícil distinguir qué corresponde a un lenguaje y qué corresponde a otro. Hay lenguajes comunes en los desarrollos web como HTML, pero por ejemplo en cuanto a lenguajes de programación no todos los programadores usan el mismo.

Acostúmbrate a encontrarte con que a veces las cosas no funcionan como una esperaría que lo hicieran, no debido a que se haya escrito mal el código o usado mal una instrucción, sino debido a que en el mundo de internet existen distintos estándares y distintas versiones. A veces aunque nos esforcemos porque todo se vea como nosotros esperamos en todos los navegadores o dispositivos, es difícil conseguirlo. Es un poco caótico, pero es así.

En este curso más que aprendernos todas las instrucciones, estándares, etc. vamos a tratar de ser capaces de razonar el por qué de las cosas, y a tratar de esforzarnos por saber cómo generar código JavaScript limpio, bien estructurado y de calidad.

Próxima entrega: CU01106E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT EN APLICACIONES WEB COMO JOOMLA, WORDPRESS, DRUPAL, PRESTASHOP, ETC. MÓDULOS, TEMPLATES O THEMES, ETC. (CU01106E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº6 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

JAVASCRIPT EN APLICACIONES WEB

Ya hemos dicho que JavaScript es un lenguaje de programación que nos permite crear efectos dinámicos y ágiles del lado del cliente. Una de las aplicaciones más ampliamente extendida de JavaScript está en dotar de dinamismo y efectos a las aplicaciones web, dentro de las que destacan los Gestores de Contenidos o CMS (Content Management Systems). Un CMS es software que se instala en el servidor y sirve para publicar contenidos en una página web fácilmente.

El concepto de aplicación web (programa que se aloja en un servidor remoto o hosting y al que accedemos a través de internet) es muy amplio, de hecho con el paso de los años se ha hecho tan amplio como los programas de ordenador o las actividades que realiza el ser humano. Las aplicaciones web se han popularizado en los últimos años gracias a que buena parte de estas aplicaciones se comenzaron a distribuir y utilizar de forma gratuita, con una comunidad de usuarios y desarrolladores de software en torno a ellas.

Muchas de estas aplicaciones sirven para que personas que no tienen conocimientos de informática gestionen páginas web como tiendas de comercio electrónico, foros, portales de contenidos, periódicos digitales, etc.

Las aplicaciones web se podrían clasificar de varias maneras. De hecho es difícil realizar una clasificación debido a que los campos en que se utilizan las distintas aplicaciones muchas veces se solapan. Vamos a hacer una clasificación común, que es basándonos en el tipo de página web para el que son más habitualmente usados:

CLASIFICACIÓN	EJEMPLOS	DESCRIPCIÓN
Gestores de Contenidos	Joomla, Drupal, OpenCMS, Plone, WordPress, b2evolution, Geeklog, Serendipity, Textpattern, CMS Made Simple, concrete5, Contao, ImpressPages, liveSite, Nucleus, PyroCMS, TYPO3, Chamilo, Moodle, phpMyFAQ, e107, Mahara, Mambo, ocPortal, PHP-Fusion, PHP-Nuke, Tiki Wiki, Xoops, Zikula.	Orientados a crear portales web de muy diferentes temáticas, desde un periódico digital hasta una tienda on-line o un blog, página personal, etc.

CLASIFICACIÓN	EJEMPLOS	DESCRIPCIÓN
Foros y libros de visitas	phpBB, SMF, fluxBB, MyBB, Vanilla Forums, XMB Forums, GBook, Lazarus GuestBook.	Pensados para la creación de sistemas de foros donde los usuarios participan intercambiándose mensajes o para libros de visitas
Wikis	MediaWiki, DocuWiki, PmWiki, WikkaWiki, TikiWiki, PikiWiki.	Pensados para mantener un sistema de información entre una comunidad de usuarios. Este sistema puede ser generalista como wikipedia o estar especializado en un área o campo de conocimiento concreto.
Tiendas y comercio electrónico	Magento, PrestaShop, CubeCart, OpenCart, osCommerce, TomatoCart, Zen Cart,	Pensadas para crear tiendas electrónicas y galerías de productos destinadas al comercio electrónico.
Utilidades varias	ExtCalendar, phpScheduleit, WebCalendar, phpFreeChat, phpMyChat, DadaMail, PHPList, SiteRecommender, OpenX, OSClass, QuickSell Classifieds, Help Center Live, Hesk, osTicket	Permiten crear calendarios, galerías de imágenes, Chats, Sistemas de envío de correo electrónico, sistemas de anuncios, sistemas de soporte a usuarios

MILLONES DE WEBS DISTINTAS

Vamos a centrarnos ahora en lo que permite mostrar una aplicación web a los usuarios. Por ejemplo, para un diario digital diremos que existe una parte denominada BackEnd donde escriben los articulistas y otra parte denominada FrontEnd que es la página web en sí del diario. Aplicaciones web que pueden servir para este propósito son Joomla, Drupal o WordPress.

Tanto en el backEnd de las aplicaciones como en el FrontEnd de estas, se utiliza JavaScript. En el backEnd no se trata tanto de crear efectos visuales atractivos, sino de dotar a la página de administración de agilidad y rapidez. En el FrontEnd el principal objetivo será crear efectos visuales atractivos y hacer que la página funcione de forma correcta y rápida.

Para conseguir webs útiles y rápidas estas herramientas incorporan código JavaScript avanzado mediante el que se realizan distintos procesos o se crean distintos efectos. La aplicación web suele contar con una parte para la gestión de contenidos mientras que otra parte denominada plantilla, template, theme, skin, etc. se encarga de controlar el aspecto.

El template o theme actúa como una piel sobre los contenidos y su principal herramienta es CSS, pero también suele estar controlado por un lenguaje de programación del lado del servidor como PHP y en algunos aspectos por un lenguaje del lado del cliente como JavaScript, además de tener “un esqueleto” de HTML.

Fíjate en estas imágenes, que corresponden a themes o plantillas del gestor de contenidos Drupal.

Fíjate en las imágenes, en los menús, en los iconos. Vamos a comentar distintos efectos que se pueden conseguir con JavaScript:

a) Es frecuente que una imagen principal en la cabecera de una página web vaya rotando o alternándose con otras imágenes (a veces con un texto superpuesto). Para hacer esto el usuario no tiene que recargar la página, es un proceso que se puede ejecutar del lado del cliente usando JavaScript.

b) Es frecuente que al pulsar en un menú se vea un efecto de despliegue o desenrollado, a veces con cambios de color, velocidad, despliegue de submenús, etc. Para hacer esto el usuario no tiene que recargar la página, es un proceso que se puede ejecutar del lado del cliente usando JavaScript.

c) Muchas veces hay iconos que permiten activar o desactivar sonidos, cambiar los colores que intervienen en la página web, ser arrastrados y colocados en distintos lugares, etc. Para hacer esto el usuario no tiene que recargar la página, es un proceso que se puede ejecutar del lado del cliente usando JavaScript.

Vemos cómo usando JavaScript se pueden conseguir muy distintos efectos y procesos. Esto ha permitido el éxito de gestores de contenidos como Joomla, Drupal o WordPress, con los que se puede crear desde una página dedicada al comercio electrónico hasta una web de un restaurante o un periódico digital. Gracias a JavaScript estas páginas pueden tener muchos contenidos y ser rápidas a la hora de realizar procesos, desplegar menús, rotar imágenes, etc.

JavaScript interviene en la programación de distintos complementos para las aplicaciones web.

Existen muchos estudios de diseño y programación donde se trabaja en la creación de templates o themes prediseñados. Hay muchos de distribución gratuita, pero la mayoría de los templates o themes de calidad son de pago (cosa lógica, ya que tienen un gran trabajo detrás). Prácticamente todos estos templates usan, entre otros lenguajes, JavaScript.

La mayoría de las aplicaciones web tienen disponibles extensiones (que reciben distintos nombres como módulos, componentes, etc.) que permiten incorporar nuevas funcionalidades. Por ejemplo si queremos incorporar una calculadora a nuestra aplicación web Joomla, Drupal o WordPress posiblemente ya exista un módulo de descarga gratuita o de pago que permita incorporar la calculadora sin tener que programarla partiendo de cero. Estas extensiones usan, entre otros lenguajes, JavaScript.

Hemos querido con esta aproximación al uso de JavaScript en aplicaciones web remarcar la importancia que ha adquirido esta técnica en los desarrollos web. No vamos a entrar de momento en cuestiones relacionadas con aplicaciones web o gestores de contenidos, sino a centrarnos en cuestiones básicas de JavaScript. La realidad en torno a las aplicaciones web es bastante compleja, ya que actualmente se tiende a trabajar con muchos lenguajes, frameworks y herramientas de carácter avanzado. Esto ya supone el uso de herramientas muy específicas, y también suele suponer la participación de distintos especialistas (diseñadores, maquetadores, expertos en CSS, programadores expertos en bases de datos, expertos en programación del lado del servidor, expertos en programación del lado del cliente, etc.) para la creación de las aplicaciones web.

Lo primero es lo primero, así que empecemos con los fundamentos de JavaScript.

Próxima entrega: CU01107E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EMPEZAR A USAR JAVASCRIPT A PARTIR DE HTML Y CSS BÁSICOS (MENÚ, LISTAS, LINKS, FORMULARIOS, ETC.) (CU01107E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº7 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

HTML Y CSS DE PARTIDA

Para ver cómo JavaScript nos sirve para dotar de procesos y efectos dinámicos a un documento HTML vamos a partir de un código HTML + CSS que representa la estructura y estilos básicos de una página web. En esta estructura la página web queda dividida en: cabecera con el título y mensaje breve, menú, texto con algunas imágenes, formulario de contacto y un pie de página con información sobre los autores o el copyright.

Crea un archivo HTML con un editor de texto como Notepad++ o similar con el siguiente contenido:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<!—Código base para el curso de javascript desde cero -->
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title>
<meta charset="utf-8">
<meta name="description" content="Portal web aprenderaprogramar.com">
<meta name="keywords" content="aprender, programar, cursos, libros">
<link rel="stylesheet" type="text/css" href="estilosA.css">
</head>
<!-- Contenido de la página web -->
<body>
<!-- Cabecera de la página web -->
<div id="cabecera">
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>
<!-- Fin de la cabecera de la página web -->
<!-- Contenedor para la parte central -->
<div id="central">
<!-- menu -->
<div id="menu">
<div class="txtPresentaMenu">Menú</div>
<hr/>
<ul class="itemsMenu">
<li><a href="#">Inicio</a></li>
<li> <a href="libros.html">Libros de programación</a> </li>
<li> <a href="cursos.html">Cursos de programación</a> </li>
<li> <a href="humor.html">Humor informático</a> </li>
</ul>
</div>
<!-- fin menu -->
```

```
<!-- cuerpo -->
<div id="cuerpoCentral">
<!-- Texto con imágenes -->
<div class="txtConImg">
<p>Aprender a programar es un objetivo que se plantea mucha gente y que no todos alcanzan.</p>
<p>Hay que tener claro que <a href="http://www.aprenderaprogramar.com">aprender programación</a> no es tarea de un día ni de una semana: aprender programación
requiere al menos varios meses y, si hablamos de programación a nivel profesional, varios años. No queremos
con esto desanimar a nadie: en un plazo de unos pocos días podemos estar haciendo nuestros primeros programas.</p>
<p>Puedes seguir uno de <a
href="http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=57&Itemid=86">nuestros cursos</a> entre los varios disponibles. Cuando haya que utilizar un editor de textos
recomendamos el uso de uno potente y sencillo como Notepad++, aunque son válidas otras alternativas como Crimson Editor.</p>
<a
href="http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=205&catid=57:herramientas-informaticas&Itemid=179">
<img src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
</a>
<img src="http://www.crimsoneditor.com/images/logo.jpg" alt="Crimson" title="Crimson, un editor de texto alternativo">
</div>
<!-- Fin del texto con imágenes -->

<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="accion.html">
<div class="form-content">
<h2>Formulario de contacto</h2>
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" placeholder="Tu nombre..." /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" placeholder="Tus apellidos..."/></label>
<label for="direccion"><span>Dirección:</span> <input id="direccion" type="text" name="direccion" placeholder="Tu dirección..."/></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" placeholder="Tu email..."/></label>
<label for="mensaje"><span>Mensaje:</span> <textarea id="mensaje" name="mensaje" cols=30 rows=2 placeholder="Tu mensaje..."></textarea></label>
<label>
<input type="submit" value="Enviar">
<input type="reset" value="Cancelar">
</label>
</div>
</form>
<!-- Fin del formulario de contacto -->
</div>
<!-- fin cuerpo -->
</div>
<!-- fin contenedor para la parte central -->

<!-- Pie de página o footer -->
<div id="pieDePagina">
<img src="http://lh5.ggpht.com/_PeVwghrmOec/TMkzEonRrcI/AAAAAAAAAHc/IxL8g0fTYtk/an_oliva_png.png" alt="logo apr" title="Logo aprenderaprogramar.com">
Copyright 2006-2038 aprenderaprogramar.com
</div>
<!-- Fin del pie de página o footer -->
</body>
<!-- Fin del contenido de la página web -->
</html>
```

Crea un archivo CSS (cuyo nombre debe ser estilosA.css) con un editor de texto como Notepad++ o similar con el siguiente contenido:

```
/* CSS para curso JavaScript aprenderaprogramar.com*/

body {font-family: sans-serif; background-color: yellow; font-size: 14px;}

h1 {color: blue;}

a {text-decoration: none;}

label {color: maroon; display:block; padding:5px;}

.itemsMenu{ padding: 3px 0px;
border-bottom: 1px solid #778;
font: bold 16px Verdana, sans-serif; }

.itemsMenu li { list-style: none; display: inline; margin-left:10px;}

.itemsMenu li a { padding: 3px 1em;
border: 1px solid #778; border-bottom: none;
background: #DDE; text-decoration: none; }

.itemsMenu li a:link { color: #448; }

.itemsMenu li a:hover {color: #000; background: #AAE; border-color: #227; }

.itemsMenu li a#actual { background: #AAE; }
```

El código anterior es HTML y CSS y lo usaremos a lo largo del curso para poner diferentes ejemplos, por lo que lo denominaremos "código base HTML del curso" o “código base CSS del curso”. Para seguir este curso debes ser capaz de comprender todo el código HTML y CSS que hemos usado, su significado y sintaxis. Si no comprendes el código anterior no continues avanzando, dirígete a la web aprenderaprogramar.com y en la sección cursos busca los cursos “Curso básico del programador web: HTML desde cero” y “Curso básico del programador web: CSS desde cero” y realízalos. Si no lo haces así no entenderás o no le sacarás todo el partido posible a este curso.

Visualiza el documento HTML en un navegador. Debes obtener un resultado similar a este (si te falla alguna imagen puedes cambiar las rutas y poner otra imagen que tú desees):

En este documento tenemos varios elementos como etiquetas de título h1 y h2, links, listas con elementos dentro de las listas, imágenes, formularios, botones, texto, etc. Todo ello nos va a servir para usar JavaScript y ver las posibilidades que nos ofrece JavaScript para crear efectos y procesos dinámicos en nuestras páginas web. En este curso nos vamos a centrar en tratar de aprender los aspectos más importantes de JavaScript y la lógica de JavaScript. El objetivo será saber hacer un buen diseño de código JavaScript, un buen uso de JavaScript y comprender lo que hacemos. Por el contrario, no vamos a tratar de aprender o conocer todas las propiedades, posibilidades o instrucciones de JavaScript ya que si logramos comprender cómo funciona y su lógica, nos bastará con realizar búsquedas en internet para encontrar aquella propiedad o sintaxis que podamos necesitar en un momento dado. “Aprende a pescar, no te conformes con que te den peces”.

Próxima entrega: CU01108E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DIFERENCIAS ENTRE NAVEGADORES EN RESPUESTAS A JAVASCRIPT (FIREFOX, EXPLORER, CHROME, SAFARI…). JAVASCRIPT EN LÍNEA. (CU01108E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº8 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

INCLUIR JAVASCRIPT EN WEBS. RESPUESTA DE LOS NAVEGADORES.

Una página web podemos verla como estructura o armazón, más estilos o apariencia, más acciones o respuestas dinámicas. En esta visión HTML es la estructura, CSS los estilos y JavaScript la respuesta dinámica. El código JavaScript se incluye en las páginas web de forma similar a como se hace con el código CSS: puede ir especificado en un apartado específico dentro del código HTML, en línea, o en un archivo externo.

JavaScript puede usarse con muchos fines, pero el más habitual es responder a una acción del usuario. Cuando el usuario hace click sobre un botón o sobre una imagen, por ejemplo, decimos que se ha producido "un evento" (¿Cuál es el evento? Que el usuario ha hecho click sobre el botón o imagen).

Nos interesa ahora ver cómo podemos incluir JavaScript dentro del código de una página web para que el navegador lo interprete y ejecute.

PRIMERA FORMA DE INCLUIR JAVASCRIPT EN WEBS

Crea un archivo HTML con este contenido, guárdalo con un nombre como ejemplo1.html y visualízalo en el navegador (puedes cambiar la ruta de la imagen si lo deseas):

```
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p>Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Podemos hacer click sobre la imagen y no obtendremos respuesta alguna.

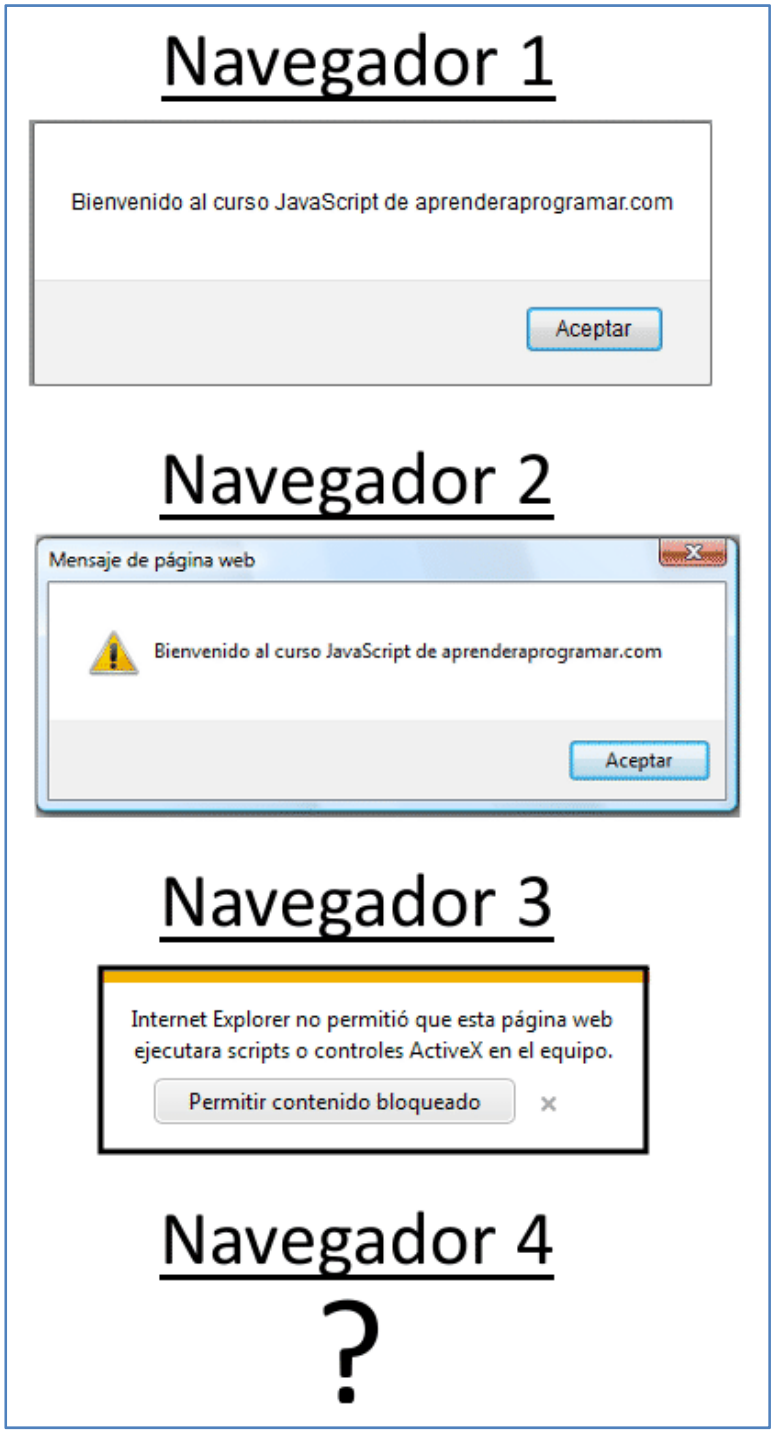
Nuestro objetivo ahora es que cuando se pulse sobre la imagen aparezca un mensaje, por ejemplo: "Bienvenido al curso JavaScript de aprenderaprogramar.com"

Ahora vamos a ver la primera forma de introducir JavaScript: hacerlo en línea. Modifica el código y comprueba el resultado de hacer click sobre la imagen después de introducir esta modificación:

```
<img onclick="alert('Bienvenido al curso JavaScript de aprenderaprogramar.com')"
src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
```

Interpretamos lo que hace este código: le indica al navegador que cuando sobre el elemento img se produzca un click (evento denominado onclick), se muestre un mensaje de alerta al usuario cuyo contenido será “Bienvenido al curso de JavaScript de aprenderaprogramar.com”.

Una vez hacemos click sobre la imagen obtendremos un resultado. ¿Qué resultado? Aquí mostramos algunas posibilidades:



El código onclick="alert('Bienvenido al curso JavaScript de aprenderaprogramar.com')" es código JavaScript, aunque esté dentro de una línea de HTML. El navegador reconoce que se trata de código JavaScript, lo interpreta y se encarga de tener en cuenta ese código y generar las acciones que correspondan.

Si has visualizado la página web en tu navegador y haz hecho click sobre la imagen lo más probable es que te haya aparecido una ventana con un mensaje informativo, pero hemos indicado distintas posibilidades que podrían tener lugar en distintos navegadores o computadores:

Posibilidad 1 (navegador 1): se muestra una ventana informativa con el texto. En este caso no tiene aspa de cierre, ni márgenes ni título.

Posibilidad 2 (navegador 2): se muestra una ventana informativa con el texto. En este caso sí tiene aspa de cierre, márgenes y título.

Posibilidad 3 (navegador 3): al hacer click sobre la imagen no ocurre nada, aunque un mensaje nos avisa de que el navegador no permitió la ejecucion de scripts y nos da opción a permitir el contenido bloqueado.

Posibilidad 4 (navegador 4): al hacer click sobre la imagen no ocurre nada, y no aparece ningún mensaje avisando de nada.

¿Por qué es posible obtener diferentes respuestas ante un código tan simple? (Donde a priori parecería que todos los navegadores deberían mostrar un mismo resultado).

Tenemos que indicar distintos motivos:

1) Cada navegador tiene su propia interfaz o apariencia para ciertos elementos. En este ejemplo, el navegador 1 muestra la ventana de mensaje con una apariencia y el navegador 2 con otra. Ambos resultados los podemos considerar “correctos”, la única diferencia está en la apariencia.

2) La ejecución de JavaScript en un navegador está ligada a la configuración de seguridad del navegador y al propio hecho de que JavaScript esté activado o no en el navegador. Hoy en día la mayor parte de los usuarios navegan con una configuración de seguridad estándar y con JavaScript activado, pero algunos usuarios pueden tener configuraciones de seguridad especiales (o incluso JavaScript desactivado). En el navegador 3 el propio navegador alerta que se está impidiendo la ejecución de un script (por su configuración de seguridad). En el navegador 4 posiblemente JavaScript esté desactivado (con lo cual JavaScript no se ejecuta y no podemos observar resultado alguno).

Nos gustaría que al incluir un código obtuviéramos siempre la misma respuesta, pero la realidad es que no siempre se obtiene la misma respuesta. El navegador que utilicemos (Firefox, Internet Explorer, Chrome, Safari, Opera, etc.) puede influir en los resultados que obtengamos. Y también puede influir la

configuración de seguridad del navegador (que impida la ejecución de ciertas cosas para evitar que un virus, un hacker, o un programa indeseado pueda generarnos algún perjuicio).

También puede influir el sistema operativo (Windows, Macintosh, Linux…) e incluso qué tipo de dispositivo sea el que estemos usando: un computador "tradicional" (pc o portátil), un smartphone, una tablet, una consola de videojuegos, una smartv, etc.

En este curso vamos a centrarnos en aprender JavaScript básico, aplicable al 99 % de los usuarios, el 99 % de los navegadores y el 99 % de los dispositivos. No obstante, acostúmbrate a que siempre pueden existir pequeñas diferencias entre navegadores o a que en algún dispositivo concreto no se obtengan los resultados deseados. Para este curso consideramos que usaremos un computador "tradicional" (pc o portátil), con un navegador como Firefox, Internet Explorer, Chrome, Opera o Safari, y con la configuración de seguridad habitual (99 % de los usuarios).

No podemos (y no tiene sentido) estar considerando todas las posibles configuraciones de seguridad, todos los posibles dispositivos, todos los posibles navegadores, etc. para tratar de tener una página web que funcione correctamente en el 100 % de los dispositivos. Nos resignamos a que nuestra web funcione correctamente en el 99 %. Si tienes algún problema para visualizar los resultados y ejecutar el código con el que iremos trabajando a lo largo del curso te recomendamos que escribas una consulta en los foros de aprenderaprogramar.com (http://aprenderaprogramar.com/foros) para obtener ayuda.

A medida que vayamos avanzando con el curso iremos comentando algunos aspectos concretos relativos a diferencias entre navegadores, cuando consideremos que puede resultar relevante.

Próxima entrega: CU01109E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT INTERNO <SCRIPT TYPE= "TEXT/JAVASCRIPT">. INTÉRPRETE JAVASCRIPT . FUNCIÓN EJEMPLO BÁSICO. (CU01109E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº9 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

INCLUIR JAVASCRIPT EN WEBS.

Hemos visto que podemos incluir JavaScript "embebido" dentro de una línea de HTML. Pero esto será poco práctico cuando tengamos que escribir pequeños programas (sería muy confuso entremezclarlo todo con el HTML). Por eso vamos a ver dos formas adicionales para escribir JavaScript de forma clara y ordenada: JavaScript interno delimitado por etiquetas, o JavaScript en archivos externos.

JAVASCRIPT INTERNO DELIMITADO POR ETIQUETAS (TAGS)

De la misma forma que el código CSS puede aparecer agrupado entre etiquetas <style> … </style>, el código JavaScript irá entre etiquetas <script> … </script> tal y como mostramos a continuación:

```
<head>
…
<script type="text/javascript">
function nombreDeFuncion1() {
… aquí el código de la función
}
function nombreDeFuncion2() {
… aquí el código de la función
}
…
</script>
</head>
```

Dentro de las etiquetas <head> … </head> incluiremos una etiqueta de apertura de declaración de script <script type="text/javascript">, a continuación colocaremos la definición de una, dos o más funciones JavaScript y terminaremos cerrando con la etiqueta </script>.

En muchas páginas web y muchos libros verás que se escribe **<script> … </script>** sin hacer uso de la definición de type. El motivo para esto es que JavaScript se ha convertido en un estándar reconocido por todos los navegadores y ya no se considera necesario especificar type="text/javascript". En versiones antiguas de HTML se consideraba necesario, pero en las actuales no es necesario especificar type ="text/javascript". Todos los navegadores actuales reconocerán JavaScript sin necesidad de la especificación type.

En algunas páginas web antiguas te podrás encontrar incluso otros tipos de sintaxis como <script language="JavaScript"....</script>.

Los motivos por los que se introdujo la especificación type son varios:

a) Existían (y existen) distintos lenguajes de script, es decir, además de javascript nos podemos encontrar otros lenguajes. El uso de type se concibió para indicarle al navegador qué lenguaje era el que usaba el script. Por ejemplo type="text/vbscript" indicaría lenguaje VBscript, type="text/tcl" indicaría lenguaje TCL, etc.

b) Fue parte de la especificación oficial de algunas versiones de HTML.

Desde el momento en que todos los navegadores reconocen JavaScript sin necesidad de la especificación type, usarla o no resulta indistinto y no vamos a preocuparnos más por este asunto. Por otro lado ten en cuenta que la etiqueta <script> y su contenido no es lenguaje JavaScript, sino lenguaje HTML.

Una cuestión importante e interesante que ha surgido al ver cómo se define JavaScript es que el código JavaScript se escribe normalmente dentro de funciones. Hablaremos sobre las funciones con más detalle más adelante. De momento, simplemente consideraremos que una función es un fragmento de código que tiene un nombre, y que puede ser invocado desde algún punto del código HTML para ser ejecutado. Para escribir una función especificaremos de momento: function nombreDeLaFuncionAqui { código … } Las funciones introducen algo interesante: cuando tengamos que hacer algo repetidas veces, no tendremos que escribir el código varias veces, sino simplemente invocar a la función para que se ejecute.

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html:

```
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
alert('Bienvenido al curso JavaScript de aprenderaprogramar.com');
}
function mostrarMensaje2() {
alert('Ha hecho click sobre el párrafo inferior');
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick="mostrarMensaje2()">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza la página. El resultado esperado será que al hacer click sobre la imagen se muestre un mensaje (el mensaje 1), mientras que al hacer click sobre el párrafo inferior se mostrará otro mensaje (el mensaje 2). Aquí vemos cómo hemos definido un script. Nos podemos hacer algunas preguntas como:

¿Puede escribirse un script en otro lugar distinto del comprendido entre las etiquetas <head> … </head>?

La respuesta es que sí. La mayoría de los navegadores aceptarán que el script esté en casi cualquier parte. Prueba a cambiar el código y ponerlo en distintos lugares y comprueba los resultados. Pero poniendo el código fuera de <head> … </head> podemos tener problemas con algunos navegadores, o podemos tener problemas en algunos casos en que el código deba guardar cierto orden. Para no tener problemas pondremos siempre los scripts dentro de las etiquetas <head> … </head>.

¿Pueden escribirse varios scripts en una misma página web?

La respuesta es que sí. Por ejemplo esto sería válido:

```
<script type="text/javascript">
function mostrarMensaje1() {
alert('Bienvenido al curso JavaScript de aprenderaprogramar.com');
}
</script>
<script type="text/javascript">
function mostrarMensaje2() {
alert('Ha hecho click sobre el párrafo inferior');
}
</script>
```

Aquí en vez de un script con dos funciones, tenemos dos scripts cada uno de ellos con una función. ¿Cuál es mejor opción, un script o dos scripts? Recordar que JavaScript es un código que es interpretado por el navegador (en concreto por una parte del navegador a la que se denomina precisamente intérprete JavaScript). Al escribir dos scripts estamos obligando al intérprete a leer más líneas, y considerar dos unidades de scripts en lugar de una. Esto ralentiza la carga de la página web y su ejecución, por tanto normalmente incluiremos el código dentro de un único script, excepto cuando justificadamente necesitemos o sea razonable hacerlo de otra manera. En páginas web avanzadas es posible que encuentres muchos scripts que salen de muchos sitios diferentes.

¿Qué extensión puede tener un script?

La que nosotros queramos. Puede ser desde una línea hasta miles de líneas.

Próxima entrega: CU01110E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ARCHIVOS DE EXTENSIÓN JS. INCLUIR JAVASCRIPT EN WEBS REFERENCIANDO UN ARCHIVO EXTERNO CON SRC (CU01110E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº10 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

JAVASCRIPT EN ARCHIVOS JS

Hemos visto que podemos incluir JavaScript "embebido" dentro de una línea de HTML o dentro de la cabecera del documento HTML. Pero esto será poco práctico cuando tengamos muchas páginas web que necesiten usar las mismas funciones JavaScript. ¿Vamos a estar repitiendo el código en cada página?

JAVASCRIPT EXTERNO EN ARCHIVOS JS

De la misma forma que el código CSS puede estar contenido en archivos externos con extensión css, el código JavaScript puede independizarse del documento HTML introduciéndolo en un archivo de texto con extensión .js y que se invocará desde el código HTML de la forma que mostramos a continuación:

```
<head>
…
…
<script type="text/javascript" src="rutaDelArchivo1.js"></script>
<script type="text/javascript" src="rutaDelArchivo2.js"></script>
<script type="text/javascript" src="rutaDelArchivo3.js"></script>
…
…
</head>
```

Podemos invocar un solo archivo, o dos, tres, cuatro… tantos como resulten necesarios.

En el atributo src tenemos que especificar la ruta del archivo referenciado (si no indicamos nada, se sobreentiende que está en el mismo directorio que el documento HTML). Supongamos que un archivo externo con código JavaScript se llamara functions.js y estuviera dentro de una carpeta denominada jsf. En ese caso escribiríamos: src="/jsf/functions.js"

En general tener el código JavaScript en archivos independientes será lo más práctico desde el punto de vista del mantenimiento de un sitio web. Tendremos las funciones agrupadas y ordenadas, pudiendo ser invocadas desde cualquier parte de nuestra página web. Esto nos ahorrará tiempo de mantenimiento: si tenemos 200 documentos html, no tendremos que hacer 200 modificaciones sino únicamente modificar el archivo o los archivos con las funciones JavaScript que usamos en todo nuestro sitio web.

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html:

```
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript" src="functions.js"></script>
</head>
<body onload="alert('Completada la carga de la página')">
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick="mostrarMensaje2()">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Crea un archivo de nombre functions.js (créalo con un editor de textos como el bloc de notas, Notepad++ o cualquier otro) y escribe el siguiente código:

```
function mostrarMensaje1(){
alert('Bienvenido al curso JavaScript de aprenderaprogramar.com');
}

function mostrarMensaje2(){
alert('Ha hecho click sobre el párrafo inferior');
}
```

El resultado esperado al cargar la página html será el siguiente:

1) Nada más cargarse la página aparecerá una ventana informativa con el texto “Completada la carga de la página”. Esto se debe a que hemos incluido JavaScript en línea asociado a la etiqueta body. El evento onload se produce cuando el elemento body se ha cargado completamente en el navegador. En ese momento se dispara el código JavaScript asociado.

2) Al hacer click sobre la imagen se muestra un mensaje (el mensaje 1), debido a que hemos indicado que cuando se produzca el evento onclick sobre la imagen, debe ejecutarse la función mostrarMensaje1(). Esta función no se encuentra dentro del documento html, pero el navegador ha incorporado el código JavaScript externo porque así se lo hemos indicado al indicar src = “functions.php”. Por tanto el código externo funciona como si fuera código incluido en el propio documento html.

3) Al hacer click sobre el párrafo inferior se mostrará otro mensaje (el mensaje 2), debido a que hemos indicado que cuando se produzca el evento onclick sobre el segundo párrafo debe ejecutarse la función mostrarMensaje2().

Nos podemos hacer algunas preguntas como:

¿Por qué tenemos una función definida en línea y otras definidas en archivos externos?

En general será más interesante tener todo el código organizado en un archivo externo en lugar de tenerlo "desperdigado" en scripts dentro de html, código en línea, etc. No obstante, acostúmbrate a que por múltiples motivos te puedes encontrar con páginas web donde la organización del código no sea buena. El navegador aceptará el código que se encuentre, tanto en línea, como interno, como externo. Como programadores deberemos intentar mantener el código tan ordenado como sea posible. Para ello es útil usar archivos externos y agrupar las funciones de un mismo tipo dentro de carpetas y archivos con nombres descriptivos. Por ejemplo si tenemos un reloj javascript, todas las funciones asociadas al reloj pueden ir dentro de un archivo de nombre reloj.js. Si tenemos una calculadora con diferentes funciones, podemos crear una carpeta calculadora y dentro de ella poner los archivos calculosAritmeticos.js, calculosFinancieros.js y calculosCientificos.js por ejemplo.

¿No hay que incluir las etiquetas <script> … </script> en el archivo js?

No, de hecho si lo hacemos el código JavaScript es probable que no funcione al no ser capaz el intérprete del navegador de entender su significado en ese contexto. En los archivos js se incluye código JavaScript únicamente. La etiqueta <script> … </script> es código html que se utiliza para delimitar el código JavaScript dentro de un documento html. Esas etiquetas no forman parte del lenguaje JavaScript, por tanto no tiene sentido incluirlas en un documento JavaScript. Si el navegador encuentra cosas extrañas en un archivo js posiblemente ignore todo su contenido (con lo cual no ejecutará ninguna función y no veremos resultado alguno).

¿Puedo ver el código JavaScript si visualizo el código fuente de la página web?

Depende. El código JavaScript en línea o interno forma parte del documento HTML, por tanto si visualizas el código fuente de la página sí podrás verlo. En cambio el código en archivos externos no podrás verlo directamente dentro del código HTML de la web, aunque sí podrás ver el archivo al que se referencia. Podrás acceder al código JavaScript escribiendo la ruta correspondiente. Por ejemplo file:///C:/EjemplosCursoJavaScript/functions.js podría ser una ruta en local (en nuestro ordenador) o http://aprenderaprogramar.com/media/system/js/modal.js podría ser una ruta en una página web on-line. De este modo podemos visualizar el código JavaScript asociado a una página web.

¿Entonces me pueden copiar otras personas mi código JavaScript (y yo copiarlo de otras webs)?

Sí. Ten en cuenta que JavaScript se ejecuta del lado del cliente, es decir, en el ordenador del usuario. Lo mismo que el código HTML y el código CSS.

¿Puede robarse información del usario o introducir virus usando JavaScript?

En general no, ya que los navegadores y sistemas operativos incluyen medidas de seguridad que impiden que a través de código JavaScript se pueda acceder al ordenador del usuario. No obstante,

siempre hay “mentes perversas” que buscan fallos de seguridad en los sistemas e intentan hacer cosas de este tipo. Es muy difícil, pero a veces lo consiguen (y seguidamente son perseguidos por la policía).

¿Deshabilitar JavaScript puede ser una buena medida de seguridad?

Podría serlo, pero no tiene demasiado sentido. Es como no salir de casa para evitar que te roben. Puedes hacerlo, pero te perderías todo lo que hay fuera de tu casa y nadie quiere vivir sin disfrutar de lo que nos puede ofrecer la vida.

¿Es posible ejecutar código JavaScript sin estar asociado a una respuesta a un evento?

Sí, escribe este código en un archivo html y visualiza el resultado en tu navegador.

```
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
</head>
<body>
<div>
<p>Hola</p>
<script type="text/javascript">
alert('Estamos cargando los contenidos de la web...');
</script>
<p>Aquí estamos</p>
</div>
</body>
</html>
```

El resultado esperado es que se cargue en pantalla el texto “Hola”. A continuación se ejecuta el código JavaScript que al no estar incluido dentro de una función se ejecuta directamente. El resultado es que se detiene la carga de la página web y aparece una ventana con el mensaje “Estamos cargando los contenidos de la web…”. Una vez pulsamos aceptar, se terminará la carga de la página y aparecerá el texto “Aquí estamos”. En este caso hemos hecho cosas extrañas (¡sólo para ver un ejemplo de cosas que te puedes encontrar!): hemos incluido un script fuera de las etiquetas <head> … </head> y hemos dado lugar a la ejecución de un código JavaScript secuencialmente con el progreso de la carga del html, en lugar de dar lugar a su ejecución en respuesta a un evento (que sería lo más normal).

Próxima entrega: CU01111E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

COMENTARIOS JAVASCRIPT EN LÍNEA O MULTILÍNEA. INSERCIÓN AUTOMÁTICA DE PUNTO Y COMA ; FINAL (CU01111E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº11 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

COMENTARIOS JAVASCRIPT

JavaScript permite insertar comentarios en el código, al igual que la mayoría de los lenguajes de programación. En concreto hay dos tipos de comentarios permitidos, los comentarios en línea que comienzan con una doble barra: //, y los comentarios multilínea, que comienzan con /* y terminan con */.

COMENTARIOS EN JAVASCRIPT

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html:

```
<html> <head> <title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
/* Funciones JavaScript
  Versión 0.1
  Autor: César Krall
  Curso: Tutorial básico del programador web: JavaScript desde cero
*/
//Función que muestra mensaje de bienvenida
function mostrarMensaje1() {
alert('Bienvenido al curso JavaScript de aprenderaprogramar.com');
}
function mostrarMensaje2() {
//Mensaje si se hace click sobre párrafo
alert('Ha hecho click sobre el párrafo inferior');
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p style="background-color:yellow;" onclick="mostrarMensaje2()">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad.

Los comentarios son parte del código JavaScript. El navegador los recibe y los detecta pero los ignora al no constituir instrucciones que hayan de ejecutarse. Sin embargo, los comentarios pueden ser visualizados si accedemos al código fuente de la página web (cosa que puede hacer cualquier usuario). Por lo tanto en los comentarios no debe figurar nada que pueda considerarse indebido (como "este código lo he copiado a mi compañero de trabajo sin permiso", ó "Para acceder a la base de datos usar como datos usuario: cesar y contraseña: aprenderaprogramar.com").

Los comentarios deben usarse para describir aspectos importantes. Por ejemplo, contenido de un archivo, cometido de una función, versión, licencia, autor, copyright, aspectos que permitan una mejor comprensión del código, avisos importantes, etc.

Obviamente los comentarios JavaScript tienen que encontrarse dentro de código JavaScript, no pueden insertarse en cualquier parte. Por tanto deberán estar dentro de las etiquetas <script> … </script> o bien dentro de un archivo js, no pueden encontrarse en el código HTML.

Los comentarios multilínea no se pueden anidar (es decir, no puede haber un comentario multilínea dentro de otro comentario multilínea). Los comentarios multilínea pueden dar lugar a errores cuando se mezclan con expresiones regulares (hablaremos de expresiones regulares más adelante).

INSERCIÓN AUTOMÁTICA DE PUNTO Y COMA

En general las sentencias JavaScript deben terminar con un punto y coma que delimita el final de una instrucción. No obstante, en caso de que "se olvide" insertar el punto y coma delimitador, el intérprete JavaScript lo insertará automáticamente siempre que le sea posible, facilitando que el código se ejecute.

En el código anterior, elimina los punto y coma al final se las sentencias JavaScript:

```
function mostrarMensaje1() {
alert('Bienvenido al curso JavaScript de aprenderaprogramar.com')
}

function mostrarMensaje2() {
alert('Ha hecho click sobre el párrafo inferior')
}
```

Visualiza la página web en tu navegador y comprueba que JavaScript sigue funcionando. ¿Por qué? Porque el intérprete del navegador, al encontrar que faltan los ; de cierre, los ha introducido automáticamente para permitir que se ejecute el código. Aunque esto puede parecer una facilidad, recomendamos siempre el cierre de toda instrucción mediante punto y coma. Esto evitará errores o que ocurran cosas indeseadas.

Próxima entrega: CU01112E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

TIPOS DE DATOS EN JAVASCRIPT. TIPOS PRIMITIVOS Y OBJETO. SIGNIFICADO DE UNDEFINED, NULL, NAN (CU01112E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº12 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

TIPOS DE VARIABLES EN JAVASCRIPT

JavaScript permite trabajar con variables, como es habitual en los lenguajes de programación. Además permite trabajar con objetos, un tipo de información más compleja que una variable simple. De forma general podríamos referirnos a tipos de datos en JavaScript, y dentro de esos tipos de datos tenemos dos grupos fundamentales: los tipos primitivos y los tipos objeto.

TIPOS DE DATOS EN JAVASCRIPT

Los tipos de datos JavaScript se dividen en dos grupos: tipos primitivos y tipos objeto.

Los tipos primitivos incluyen: cadenas de texto (**String**), variables booleanas cuyo valor puede ser true o false (**Boolean**) y números (**Number**). Además hay dos tipos primitivos especiales que son **Null** y **Undefined**. Es en los tipos primitivos donde vamos a centrarnos por el momento.

Los tipos objeto son datos interrelacionados, no ordenados, donde existe un nombre de objeto y un conjunto de propiedades que tienen un valor. Un objeto puede ser creado específicamente por el programador. No obstante, se dice que todo aquello que no es un tipo primitivo es un objeto y en este sentido también es un objeto, por ejemplo, una función. Lo estudiaremos más adelante.

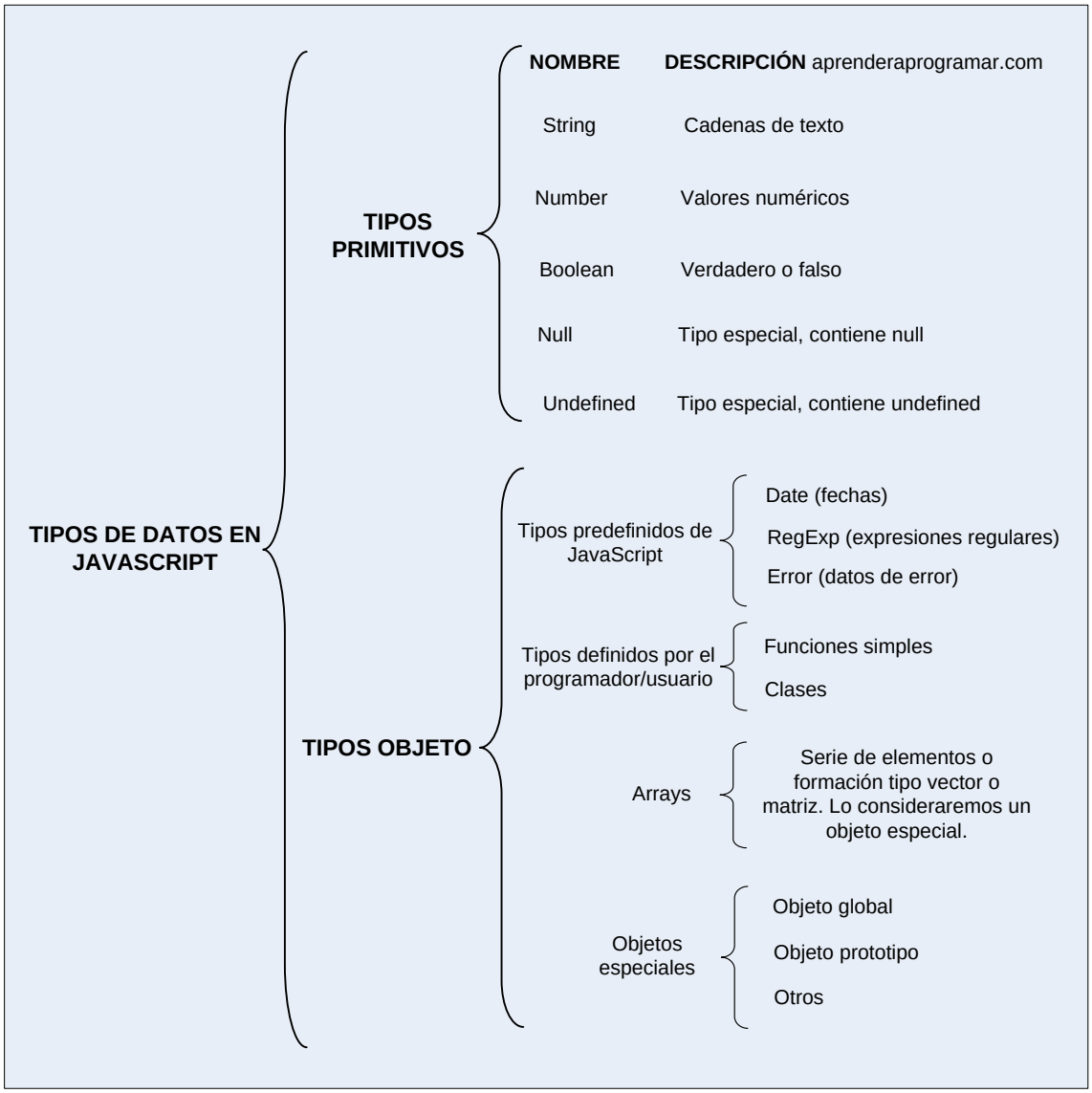
Existen algunos **objetos especiales** con un comportamiento que estudiaremos por separado más adelante: el "objeto global", el "objeto prototipo", los arrays, las funciones, las clases definidas por el programador, las clases predefinidas JavaScript (como la clase Date para manejo de fechas, la clase RegExp para manejo de expresiones regulares y búsqueda de patrones en texto, y la clase Error para almacenar información relacionada con los errores) y otros.

Los primeros lenguajes de programación no usaban objetos, solo variables. Una variable podríamos decir que es **un espacio de la memoria** del ordenador a la que asignamos un contenido que puede ser básicamente un valor numérico (sólo números, con su valor de cálculo) o de tipo carácter o cadena de caracteres (valor alfanumérico que constará sólo de texto o de texto mezclado con números).

Como ejemplo podemos definir una variable a que contenga 32 y esto lo escribimos como a = 32. Posteriormente podemos cambiar el valor de a y hacer a = 78. O hacer "a" equivalente al valor de otra variable "b" así: a = b.

Dado que antes hemos dicho que un objeto también ocupa un espacio de memoria: ¿en qué se parecen y en qué se diferencia un objeto de una variable? Consideraremos que las variables son entidades elementales: un número, un carácter, un valor verdadero o falso… mientras que los objetos son entidades complejas que pueden estar formadas por mucha información. Pero ambas cosas ocupan lo mismo: un espacio de memoria (que puede ser más o menos grande).

El siguiente esquema es un resumen sobre tipos de datos en JavaScript.



No te preocupes ahora por conocer todos los posibles tipos de datos de JavaScript ni su significado. Los iremos estudiando y trabajando con ellos poco a poco.

Frente a otros lenguajes fuertemente tipados (las variables tienen un tipo declarado y no pueden cambiar el tipo de contenido que almacenan) como Java, JavaScript es un un lenguaje **débilmente tipado**: las variables pueden no ser de un tipo específico y cambiar el tipo de contenido que almacenan.

CONCEPTOS BÁSICOS SOBRE VARIABLES

Una variable se declara con la palabra clave **var**. Por ejemplo **var precio;** constituye la declaración de una variable denominada precio. En la declaración no figura qué tipo de variable es (por ejemplo si es texto tipo String o si es numérica tipo Number). Entonces, ¿cómo se sabe de qué tipo es una variable? JavaScript decide el tipo de la variable **por inferencia**. Si detecta que contiene un valor numérico, la considerará tipo Number. Si contiene un valor de tipo texto la considerará String. Si contiene true ó false la considerará booleana.

El nombre de una variable deberá estar formado por letras, números, guiones bajos o símbolos dólar ($), **no siendo admitidos otros símbolos**. El nombre de la variable no puede empezar por un número: obligatoriamente ha de empezar con una letra, un signo dólar o un guión bajo. Por tanto son nombres de variables válidos precio, $precio, _precio_ , _$dato1, precio_articulo, etc. y no son nombres válidos 12precio ni precio# ó pre!dato1.

Una variable se inicializa cuando se establece su contenido o valor por primera vez. Por ejemplo **precio = 22.55;** puede ser una forma de inicializar una variable.

Una variable se puede declarar e inicializar al mismo tiempo. Por ejemplo podríamos escribir **var precio = 22.55;** con lo que la variable ha sido declarada e inicializada en una misma línea.

JavaScript **no requiere declaración del tipo** de las variables, e incluso permite que una variable almacene contenido de distintos tipos en distintos momentos. Por ejemplo podríamos usar precio = 22.55; y en un lugar posterior escribir precio = 'muy caro';. Esto, que en otros lenguajes generaría un error, es aceptado por JavaScript.

JavaScript **distingue entre mayúsculas y minúsculas** (no sólo para las variables): por tanto no es lo mismo precio = 22.55 que Precio = 22.55. Precio es una variable y precio otra.

JavaScript permite hacer uso de una variable **sin que haya sido declarada**. En muchos lenguajes de programación es necesario declarar una variable antes de poder hacer uso de ella, pero JavaScript no obliga a ello. Cuando JavaScript se encuentra una variable no declarada, crea automáticamente una variable y permite su uso. Hablaremos de ello más adelante, de momento usaremos variables declaradas.

El uso inadecuado de la palabra clave var puede generar un error ante el que JavaScript deja de funcionar, es decir, la página web se visualizará correctamente pero **JavaScript no se ejecuta**. Un ejemplo de código erróneo podría ser este:

```
var precio1=44.20; alert ('La variable ahora es ' + var precio1);
```

var se usa únicamente para declarar una variable. No puede usarse para otra cosa. Una vez declarada la variable, ya se hará uso de ella sin precederla de la palabra clave var. Si se declara una variable estando ya declarada, JavaScript intentará continuar (y posiblemente lo consiga), pero esto puede considerarse una mala práctica excepto si se sabe muy bien lo que se está haciendo.

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html>
<head>
<title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
alert('La variable precio vale: ' + precio);
dato1 = null;
alert('La variable dato1 vale: ' + dato1);
var precio = 22.55;
precio = precio + 10;
alert('La variable precio vale: ' + precio);
alert('El doble de precio es: ' + (precio*2));
cantidad = 10;
alert('El importe resultante de multiplicar precio por cantidad es: ' + (precio*cantidad));
precio = 'muy caro';
alert('La variable precio vale ahora: ' + precio);
alert('El doble de precio es ahora: ' + (precio*2));
var precio = 99.55
alert('La variable precio ha sido declarada por segunda vez y ahora vale: ' + precio);
var $descuento_aplicado = 0.55;
alert('La variable $descuento_aplicado vale : ' + $descuento_aplicado);
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++,
un útil editor de texto">
<p style="background-color:yellow;" onclick="mostrarMensaje2()">Aquí otro párrafo de texto. JavaScript es
un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente:

La variable precio vale: undefined (Aceptar)
La variable dato1 vale: null (Aceptar)
La variable precio vale: 32.55 (Aceptar)
El doble de precio es: 65.1 (Aceptar)
El importe resultante de multiplicar precio por cantidad es: 325.5 (Aceptar)
La variable precio vale ahora: muy caro (Aceptar)
El doble de precio es ahora: NaN (Aceptar)
La variable precio ha sido declarada por segunda vez y ahora vale: 99.55 (Aceptar)
La variable $descuento_aplicado vale: 0.55 (Aceptar)

SIGNIFICADO DE UNDEFINED, NULL Y NAN

De este ejemplo debemos destacar lo siguiente:

a) Se cumplen los **conceptos básicos** explicados.
b) El contenido de una variable no inicializada es **undefined**. En este caso decimos que la variable es de tipo Undefined.
c) El contenido de una variable puede ser **null** y en ese caso decimos que la variable es de tipo Null.
d) Si intentamos realizar una operación matemática con una variable cuyo contenido no es numérico sino texto, la variable toma el valor **NaN**. Para JavaScript NaN (abreviatura de "Not-a-Number") es un valor numérico especial, que representa "número ilegal o no representable". La asignación de NaN que realiza JavaScript automáticamente cuando se intentan realizar operaciones numéricas ilegales evita la aparición de errores explícitos o que el código JavaScript deje de ejecutarse.

EJERCICIO

Crea un código JavaScript donde se genere un error por un mal uso de la palabra clave var y comprueba la respuesta del navegador.

Crea una variable y asignale los contenidos true y false, e intenta multiplicarlas por un número (por ejemplo por 2). ¿Qué resultados obtienes? ¿A qué crees que se deben estos resultados?

Para comprobar si es correta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01113E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

TIPOS DE VARIABLES JAVASCRIPT: NUMÉRICAS O NUMBER (INT , INTEGER, SINGLE, DOUBLE, FLOAT…). RESULTADOS NAN E INFINITY (CU01113E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº13 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

VARIABLES NUMÉRICAS EN JAVASCRIPT

A diferencia de muchos otros lenguajes de programación (como Java, C, C++, Visual Basic, etc.), JavaScript no diferencia distintos tipos de variables numéricas. Todos los números en JavaScript son tratados como un tipo numérico único, el tipo Number, que "agrupa" a los tipos int, integer, double, float, single, etc. que se utilizan en otros lenguajes.

VARIABLES NUMÉRICAS JAVASCRIPT

JavaScript permite representar cualquier número que sea necesario. Ten en cuenta que esto no significa que se puedan usar números "descomunales" porque un computador siempre ha de tener un límite. Pero el límite de JavaScript es del orden de $\pm 1.7976931348623157 \times 10^{308}$ para números grandes(esto son millones de trillones, un número tan grande que nunca tendremos problemas porque nunca vamos a tener que usar un número superior a esta cifra) y del orden de $\pm 5 \times 10^{-324}$ para números pequeños (esto supone que se puede trabajar con números con cientos de decimales, y en la práctica nunca vamos a necesitar tanta precisión, con lo cual nunca tendremos problemas por usar números muy pequeños o con gran número de decimales).

Para indicar que un número es negativo se precede del signo menos.

Los números decimales se escriben utilizando el punto (.) como separador. Si la parte entera de un número es el cero, se admite omitir el cero. Es decir, 0.55 y .55 son ambos admitidos.

También se admite la notación basada en indicar un número seguido de E y la potencia de 10 a la que se debe elevar. También se admite usar la e minúscula. Por ejemplo 1.2E2 ó 1.2e2 equivale a $1.2*10^2$ = 1.2*100 = 120. Esta notación permite escribir números muy grandes o muy pequeños sin tener que escribir todas las cifras. Por ejemplo 3.2345234565E20 equivale a 323452345650000000000 ó 3.2345234565e-10 equivale a 0.00000000032345234565

El uso de número decimales puede presentar en ocasiones problemas debido al redondeo decimal, como veremos. Este problema no es exclusivo de JavaScript, sino que es algo que afecta a numerosos lenguajes de programación.

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var bacterias = 3.55; var texto = 'bacterias en la probeta'; var numeroInfinito = Infinity;
alert('La variable bacterias vale: ' + bacterias);
bacterias = 3.55E5; alert('La variable bacterias vale (multiplicamos por 100000): ' + bacterias);
bacterias = 3.55E-5; alert('La variable bacterias ahora es un número muy pequeño: ' + bacterias);
alert('La variable bacterias ahora es (operación sin sentido): ' + bacterias*texto);
bacterias = 3.55e100000000000000000; alert('La variable bacterias ahora es demasiado grande: ' +
bacterias);
bacterias = 3.55E-100000000000000000; alert('La variable bacterias ahora es demasiado pequeña: ' +
bacterias);
alert('Un numero positivo dividido entre cero (indeterminación matemática) devuelve: ' + (4/0));
alert('Un numero negativo dividido entre cero (indeterminación matemática) devuelve: ' + (-4/0));
alert('Cero dividido entre cero devuelve: ' + (0/0));
alert('La variable numeroInfinito vale: ' + numeroInfinito);
var diezCentimos = .1; var veinteCentimos = .2; var treintaCentimos = .3;
alert('Esperamos 0.1 y lo obtenemos: ' + (veinteCentimos-diezCentimos));
alert('Esperamos 0.1 y no lo obtenemos: ' + (treintaCentimos-veinteCentimos));
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++,
un útil editor de texto">
<p style="background-color:yellow;" onclick="mostrarMensaje2()">Aquí otro párrafo de texto. JavaScript es
un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente:

La variable bacterias vale: 3.55 (Aceptar)
La variable bacterias vale (multiplicamos por 100000): 355000 (Aceptar)
La variable bacterias ahora es un número muy pequeño: 0.0000355 (Aceptar)
La variable bacterias ahora es (operación sin sentido): NaN (Aceptar)
La variable bacterias ahora es demasiado grande: Infinity
La variable bacterias ahora es demasiado pequeña: 0
Un numero positivo dividido entre cero (indeterminación matemática) devuelve: Infinity
Un numero negativo dividido entre cero (indeterminación matemática) devuelve: -Infinity
Cero dividido entre cero devuelve: NaN
La variable numeroInfinito vale: Infinity
Esperamos 0.1 y lo obtenemos: 0.1
Esperamos 0.1 y no lo obtenemos: 0.09999999999999998

SIGNIFICADO DE NAN, INFINITY, -INFINITY Y PROBLEMAS DE REDONDEO

De este ejemplo debemos destacar lo siguiente:

a) Se cumplen los **conceptos básicos** explicados.
b) Una operación sin sentido devuelve el valor NaN (Not-a-Number), valor equivalente a "valor numérico no válido".
c) Un valor numérico positivo excesivamente grande (fuera de los límites admisibles) es representado como **Infinity**. Infinity puede considerarse como equivalente a "valor numérico positivo excesivamente grande o tendente a infinito". A una variable puede asignársele valor Infinity.
d) Un valor numérico excesivamente próximo a cero (fuera de los límites admisibles) es representado como cero.
e) Un valor numérico negativo excesivamente grande (fuera de los límites admisibles) es representado como **-Infinity**. -Infinity puede considerarse como equivalente a "valor numérico negativo excesivamente grande o tendente a menos infinito".
f) Algunas indeterminaciones matemáticas se representan como Infinity ó –Infinity y otras como **NaN** (Not-a-Number).
g) Pueden surgir problemas de redondeo al operar con decimales. Por ejemplo 0.3 – 0.2 sería de esperar que diera el mismo resultado que 0.2 – 0.1 y es posible que no dé el mismo resultado. ¿Por qué? Este problema se debe a que JavaScript (al igual que muchos otros lenguajes) no trabaja directamente con los decimales, sino que transforma esos decimales en una representación interna que al realizar operaciones puede dar lugar a resultados inesperados debido al **redondeo** de esa representación interna. Esta situación puede cambiar según el navegador o la versión de JavaScript que se utilice, pero para evitar problemas, se recomienda operar con números enteros y usar los decimales a la hora de visualizar por pantalla. En el ejemplo visto usaríamos 10, 20 y 30 y operaríamos con estos valores enteros. A la hora de mostrar por pantalla, haríamos la división entre 10.

EJERCICIO

Crea un código JavaScript para evitar el problema del redondeo que hemos tenido en el código anterior. Para ello define los decimales como enteros y realiza la operación para mostrar el decimal sólo en el momento de mostrar el resultado por pantalla.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01114E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

VARIABLES STRING JAVASCRIPT. ¿DEBEN USARSE COMILLAS SIMPLES O DOBLES? CARACTERES DE ESCAPE \N \T.LENGTH (CU01114E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº14 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

CARACTERES Y CADENAS DE TEXTO EN JAVASCRIPT

A diferencia de otros lenguajes que diferencian un tipo de dato “carácter” (char) y otro cadena de texto (string), en JavaScript existe un único tipo para englobar tanto a caracteres como a cadenas de texto como a una cadena vacía: el tipo String.

VARIABLES STRING EN JAVASCRIPT

JavaScript permite definir el texto tanto dentro de comillas dobles como dentro de comillas simples. Por ejemplo es válido: var cadenaTexto; cadenaTexto = "aprenderaprogramar.com" ;

Y también es válido:

cadenaTexto = 'aprenderaprogramar.com' ;

HTML también permite usar comillas simples o dobles indistintamente. Esto puede dar lugar a problemas. Por ejemplo:

onclick ="alert("Alerta JavaScript")" sería una construcción incorrecta porque el navegador no sabe interpretar qué comillas son de apertura y cuáles de cierre.

onclick ="alert('Alerta JavaScript')" sería una construcción correcta porque el navegador determina que las comillas dobles son las externas y las comillas simples las internas.

Nosotros preferiremos usar comillas dobles para HTML y comillas simples para JavaScript, aunque no hay nada que obligue a que esto tenga que ser así.

En JavaScript siempre que se quiera indicar la presencia de un texto se puede optar por comillas dobles o simples, según se prefiera. No obstante, habrá situaciones en las que queramos que existan comillas dentro del propio texto, y en este caso tenemos la opción a usar un tipo de comillas como delimitadoras externas y otro tipo como comillas internas del texto. Por ejemplo:

cadenaTexto = '"aprenderaprogramar.com"' ; supone que el texto está delimitado por comillas simples y que la cadena de texto contiene las comillas dobles.

cadenaTexto = " 'aprenderaprogramar.com' " ; supone que el texto está delimitado por comillas dobles y que la cadena de texto contiene las comillas simples.

Aún así, existirán casos en que queramos que un texto contenga simultáneamente comillas simples y dobles, con lo que la solución anterior no nos resulta satisfactoria. Para resolver estas situaciones, se usa el denominado carácter backslah o carácter de escape, que es el símbolo \.

El símbolo \ se usa para resolver la representación de símbolos que no pueden ser incluidos de forma normal dentro de un texto. Cabe destacar las siguientes secuencias de escape:

Secuencia de escape	Resultado
\'	Comilla simple
\"	Comilla doble
****	Símbolo \
\n	Nueva línea
\t	Tabulador

Existen más caracteres de escape pero estos son los más usuales. También es posible introducir caracteres (de escape o no) usando la codificación Latin-1 o Unicode en que se basa JavaScript, por ejemplo \u0041 representa la letra A, \u00F3 representa la letra ó, \u005C representa el carácter \ y \xA9 representa el símbolo de copyright ©, pero esta codificación en general no la utilizaremos salvo en casos muy excepcionales. Si necesitas comprobar el código de los caracteres puedes hacerlo buscando en internet “List of Unicode characters”.

El símbolo \ incluido dentro de un texto será ignorado. Para introducir el símbolo \ siempre hemos de hacerlo escapándolo usando \\.

LONGITUD DE UNA CADENA DE CARACTERES

La longitud de una cadena de caracteres expresada como un valor numérico puede obtenerse escribiendo la cadena (o el nombre de la variable que la representa) seguido de .length.

Por ejemplo "extraordinario".length devuelve 14 por contener la palabra 14 letras y "pitón".length devuelve 5 por contener la palabra 5 letras.

textoUsuario.length devolverá el número de caracteres que contenga la variable textoUsuario.

"".length devuelve cero porque "" representa una cadena vacía o con cero caracteres.

En general cada carácter suma una unidad a la propiedad length del texto (aunque algunos caracteres extraños suman 2 unidades por representarse concatenando dos códigos Unicode, pero esto es algo a lo que no le prestaremos atención porque en la práctica casi nunca tendremos que trabajar con este tipo de caracteres extraños).

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var textoUsuario;
var cadenaTexto;
cadenaTexto =  '\xA9 aprenderaprogramar.com' ; alert ('cadena texto vale ' + cadenaTexto);
cadenaTexto =  '"aprenderaprogramar.com"' ; alert ('cadena texto es ' + cadenaTexto);
cadenaTexto = "Letra A: \u0041, o con tilde: \u00F3 sigue Caracter \\ igualmente \u005C seguido de tres saltos de línea \n\n\n Prosigue comilla simple \' y doble \"\n\n\n"
alert ('cadena texto ahora contiene ' + cadenaTexto);
textoUsuario = prompt("Introduzca un texto por favor:");
alert ("El texto introducido fue " + textoUsuario + " con longitud de " + textoUsuario.length + " caracteres");
alert ("La longitud de extraordinario es de " + ("extraordinario".length) + " caracteres");
alert ("La longitud de la cadena vacía es de " + (""".length) + " caracteres");
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="background-color:yellow;">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo):

cadena texto vale © aprenderaprogramar.com
cadena texto es "aprenderaprogramar.com"
cadena texto ahora contiene Letra A: A, o con tilde: ó sigue Caracter \ igualmente \ seguido de tres saltos de línea

Prosigue comilla simple ' y doble "
Introduzca un texto por favor: andamio
El texto introducido fueandamio con longitud de 7 caracteres
La longitud de extraordinario es de 14 caracteres
La longitud de la cadena vacía es de 0 caracteres

EJERCICIO

Consulta en internet cuál es el código unicode que corresponde al carácter π (símbolo matemático Pi) y usando el código muestra por pantalla el mensaje “El número π vale 3.1416”.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01115E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DECLARAR VARIABLES JAVASCRIPT E INICIALIZARLAS. VARIABLES BOOLEANAS Y VALORES TRUE Y FALSE. TOSTRING() (CU01115E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº15 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DECLARACIÓN DE VARIABLES JAVASCRIPT. VARIABLES BOOLEANAS.

Aunque no es obligatorio, se recomienda declarar siempre las variables antes de usarlas en JavaScript. Como sabemos esta declaración se hace escribiendo la palabra clave var seguida del nombre de variable. También se pueden declarar múltiples variables en una sola línea escribiendo var y a continuación los nombres de las variables separados por comas.

Por ejemplo podemos declarar:

```
var coches;
var motos;
var trenes;
```

O bien hacerlo en una sola línea de esta manera: var coches, motos, trenes;

También se permite la inicialización de variables en línea. Por ejemplo podríamos escribir:

```
var coches = 32, motos = 9, trenes = 12;
```

De este modo quedan las variables inicializadas en el mismo momento de declaración.

Aunque la declaración de variables no es obligatoria, se recomienda siempre declarar las variables. Si se trata de hacer uso de un nombre de variable que no existe para asignárselo a una variable sí existente, se produce un error y el código JavaScript deja de funcionar. Ejemplo:

```
function mostrarMensaje1() {
var mayorDe25 = false;
mayorDe25 = variableInexistente; //Genera un error y el código no se ejecuta
alert ('Mostrar un mensaje no se ejecuta');
```

Importante: un “pequeño error” al escribir un nombre de variable puede generar un error (no visualizado al no aparecer mensaje de error) y hacer que nuestro código JavaScript deje de funcionar.

VARIABLES BOOLEANAS

En JavaScript una variable es booleana si se le asigna como contenido true ó false. Estas variables se usan para almacenar información del tipo sí / no, cumple / no cumple, verdadero / falso, existe / no existe, es decir, información que únicamente puede tener dos estados.

Ejemplo: var casado = true;

true y false son palabras clave JavaScript. Por tanto no podemos usarlas como nombres de variables. Si escribiéramos true = 25; el intérprete del navegador interpretaría que existe un error, por lo que el

código JavaScript no se ejecutará (sin embargo, no veremos ningún mensaje de error, simplemente veremos que JavaScript no funciona).

Las variables booleanas normalmente se emplean para evaluar si se cumplen condiciones mediante instrucciones if else, como veremos más adelante.

Un razonamiento usando variables booleanas puede ser <<Si la variable casado contiene verdadero entonces mostrar por pantalla el mensaje 'El usuario está casado'>>.

Las variables booleanas toman valor true o false. Puede considerarse que existe un equivalente numérico (1 para true y 0 para false), pero debemos decidir si queremos operar con números, usar una variable numérica, y si queremos operar con los valores true y false, operar con variables booleanas.

En general toda variable tiene un equivalente numérico y un equivalente en forma de texto. Por ejemplo el equivalente en forma de texto del valor true sería 'true' y se puede obtener escribiendo el nombre de la variable seguido de toString(). Por ejemplo casado.toString() devuelve la cadena de texto 'true', que es diferente del valor booleano true.

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var a=1, b=2, c=3;
var casado = true;
var mayorDe25 = false;
var texto1 = 'En un lugar de la Mancha...';
//mayorDe25 = variableInexistente; //Genera un error y el código no se ejecuta
alert ('La variable casado vale: ' + casado);
alert ('La variable mayorDe25 vale: ' + mayorDe25);
alert ('La variable casado vale (forzamos mostrar equivalente numérico): ' + casado*1);
alert ('La variable mayorDe25 vale (forzamos mostrar equivalente numérico): ' + mayorDe25*1);
alert ('La variable casado vale ahora: ' + casado);
alert ('La suma de a, b y c vale: ' + (a+b+c));
alert ('Operación sin sentido, obtenemos: ' + (texto1*1));
alert ('Mostramos equivalente de texto de la variable casado: ' + casado.toString());
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="background-color:yellow;">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo):

La variable casado vale: true (Aceptar)
La variable mayorDe25 vale: false (Aceptar)
La variable casado vale (forzamos mostrar equivalente numérico): 1 (Aceptar)
La variable mayorDe25 vale (forzamos mostrar equivalente numérico): 0 (Aceptar)
La variable casado vale ahora: true (Aceptar)
La suma de a, b y c vale: 6 (Aceptar)
Operación sin sentido, obtenemos: NaN (Aceptar)
Mostramos equivalente de texto de la variable casado: true (Aceptar)

EJERCICIO

Crea un código JavaScript y declara dos variables booleanas. Asigna a una de ellas valor true y a otra valor false. Intenta mostrar por pantalla el resultado de dos operaciones a priori carentes de lógica: el resultado de la suma de las dos variables booleanas y el resultado de la suma de la conversión de las variables booleanas en String mediante el uso de toString(). Visualiza los resultados e intenta razonar el por qué de cada uno de ellos.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01116E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

OPERADORES ARITMÉTICOS JAVASCRIPT. %, MOD O RESTO DE UNA DIVISIÓN ENTRE ENTEROS (MÓDULO). (CU01116E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº16 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

OPERADORES ARITMÉTICOS EN JAVASCRIPT. RESTO DE DIVISIÓN.

En JavaScript disponemos de los operadores aritméticos habituales en lenguajes de programación como son suma, resta, multiplicación, división y operador que devuelve el resto de una división entre enteros (en otros lenguajes denominado operador mod o módulo de una división):

OPERADORES BÁSICOS	DESCRIPCIÓN
+	Suma
–	Resta
*	Multiplicación
/	División
%	Resto de una división entre enteros (en otros lenguajes denominado mod)

Operadores aritméticos en JavaScript

Existen otros operadores no básicos como ++ y -- que comentaremos más adelante.

Destacar que el operador % es de uso exclusivo entre enteros. 7%3 devuelve 1 ya que el resto de dividir 7 entre 3 es 1. 8%2 devuelve 0 ya que el resto de dividir 8 entre 2 es cero. Al valor obtenido lo denominamos módulo (en otros lenguajes en vez del símbolo % se usa la palabra clave *mod*) y a este operador a veces se le denomina "operador módulo".

Aunque en otros lenguajes existe un operador de exponenciación para calcular potencias, en JavaScript no es así. Para calcular una potencia podemos hacer varias cosas:

a) Recurrir a multiplicar n veces el término. Por ejemplo min^3 lo podemos calcular como min*min*min. Obviamente esto no es práctico para potencias de exponentes grandes.

b) Usar un bucle que dé lugar a la repetición de la operación multiplicación n veces, o usar una función que ejecute la operación. Estas opciones las comentaremos más adelante.

c) Usar herramientas propias del lenguaje que permiten realizar esta operación. Esta opción la comentaremos más adelante.

Las operaciones con operadores siguen un **orden de prelación o de precedencia** que determinan el orden con el que se ejecutan. Con los operadores matemáticos la multiplicación y división tienen precedencia sobre la suma y la resta. Si existen expresiones con varios operadores del mismo nivel, la operación se ejecuta de izquierda a derecha. Para evitar resultados no deseados, en casos donde pueda existir duda se recomienda el uso de paréntesis para dejar claro con qué orden deben ejecutarse las operaciones. Por ejemplo, si dudas si la expresión 3 * a / 7 + 2 se ejecutará en el orden que tú deseas, especifica el orden deseado utilizando paréntesis: por ejemplo 3 * ((a / 7) + 2).

DOBLE SENTIDO DEL OPERADOR +

En JavaScript el operador + se usa para realizar sumas pero también para concatener cadenas, es decir, realiza una operación u otra según el tipo de variables a las que se aplique. Así si tenemos dos variables denominadas adcon1 y adcon2 podríamos ejecutar código de este tipo:

```
var adcon1, adcon2, result;
adcon1=3;
adcon2 = 4;
result = adcon1 + adcon2; // result es de tipo numérico y contiene 7
adcon1= 'Amanecer'
adcon2 = ' cálido';
result = adcon1 + adcon2; // result es ahora de tipo String y contiene “Amanecer cálido”
```

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html> <head> <title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var adcon1, adcon2, adcon3;
var cadenaTexto1; var cadenaTexto2; var cadenaTexto3; var espacioTxt = ' ';
adcon1=3; adcon2 = 4; result = adcon1 + adcon2; alert ('Ahora result vale ' + result);
adcon1= 'Amanecer'; adcon2 = ' cálido'; result = adcon1 + adcon2; alert ('Ahora result vale ' + result);
cadenaTexto1 = 'Tomate'; cadenaTexto2 = 'frito'; cadenaTexto3 = cadenaTexto1+espacioTxt+cadenaTexto2;
alert ('La concatenación del texto es: \"' + cadenaTexto3 +'\"');
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="background-color:yellow;">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo):

Ahora result vale 7
Ahora result vale Amanecer cálido
La concatenación del texto es: "Tomate frito"

EJERCICIO

Crea un código JavaScript para pedir al usuario que introduzca dos números (mediante el uso de prompt como hemos visto en anteriores epígrafes del curso) y devuelva mensajes informativos con:

a) El resto de dividir el primer número entre 5.

b) El resultado de dividir el primer número entre el segundo.

c) El resultado de sumar los dos números.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01117E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

OPERADORES LÓGICOS JAVASCRIPT. EJEMPLOS. RELACIONALES MAYOR, MENOR, IGUAL, DISTINTO. AND, OR, NOT. CORTO-CIRCUITO (CU01117E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº17 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

OPERADORES LÓGICOS PRINCIPALES EN JAVASCRIPT

En JavaScript disponemos de los operadores lógicos habituales en lenguajes de programación como son “es igual”, “es distinto”, menor, menor o igual, mayor, mayor o igual, and (y), or (o) y not (no). La sintaxis se basa en símbolos como veremos a continuación y cabe destacar que hay que prestar atención a no confundir == con = porque implican distintas cosas.

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

Operadores lógicos y relacionales principales en JavaScript

La sintaxis coincide con la empleada en otros lenguajes de programación.

Además de los operadores habituales existen los operadores === que se interpreta como “es estrictamente igual” y !== que se interpreta como “no es estrictamente igual”. Estos operadores resultan un poco más complejos de comprender por lo que volveremos a hablar de ellos más adelante. De momento tener en cuenta que si una variable contiene texto1= “1” y hacemos la comparación texto1 === 1 , obtendremos false, es decir, que no es igual (porque un texto no es igual a un número). Sin embargo una comparación como texto == 1 devolverá true ya que esta comparación no es estricta y

trata de realizar automáticamente conversiones para comprobar si se puede establecer una equivalencia entre los dos valores. En este caso se busca el equivalente numérico del texto y luego se hace la comparación, motivo por el cual se obtiene true.

Las expresiones donde se utilizan operadores lógicos y relacionales devuelven un valor booleano, es decir, verdadero (true) o falso (false). Por ejemplo si a = 7 y b = 5 la expresión a < b devuelve false (es falsa). Si a = true y b = false la expresión a && b devuelve false (es falsa porque no se cumple que a y b sean verdaderas). Si a = true y b = false la expresión a || b devuelve true porque uno de los dos operandos es verdadero. Si a = true la expresión !a devuelve false (el opuesto o contrario).

El operador || se obtiene en la mayoría de los teclados pulsando ALT GR + 1, es decir, la tecla ALT GR y el número 1 simultáneamente.

Los operadores && y || se llaman **operadores en cortocircuito** porque si no se cumple la condición de un término no se evalúa el resto de la operación. Por ejemplo: (a == b && c != d && h >= k) tiene tres evaluaciones: la primera comprueba si la variable a es igual a b. Si no se cumple esta condición, el resultado de la expresión es falso y no se evalúan las otras dos condiciones posteriores.

En un caso como (a < b || c != d || h <= k) se evalúa si a es menor que b. Si se cumple esta condición el resultado de la expresión es verdadero y no se evalúan las otras dos condiciones posteriores.

El operador ! recomendamos no usarlo hasta que se tenga una cierta destreza en programación. Una expresión como (!esVisible) devuelve false si (esVisible == true), o true si (esVisible == false). En general existen expresiones equivalentes que permiten evitar el uso de este operador cuando se desea.

OPERADOR DE NEGACIÓN APLICADO SOBRE NÚMEROS O TEXTO

Si a = true su negación!a devuelve false. ¿Pero qué ocurre si a es un número o un texto? Si a es un número se considera que equivale a false si su valor numérico es 0, o que equivale a true si su valor numérico es distinto de cero. Seguidamente se aplica la negación. Por tanto si a = 7, a se considera equivalente a true y !a es false. Si a = 0, a se considera equivalente a false y !a es true.

Para cadenas de texto, la cadena vacía se considera equivale a false y cualquier otra cadena se considera que equivale a true. Si texto1 = “” (cadena vacía) entonces !texto1 vale true.

COMPARACIÓN DE CADENAS DE TEXTO CON OPERADORES RELACIONALES

Dos cadenas de texto se pueden comparar resultando que se comparan letra a letra por el valor del equivalente numérico de cada letra. Cada letra tiene un número asociado: por ejemplo la a es el número 97, la b el 98, etc.

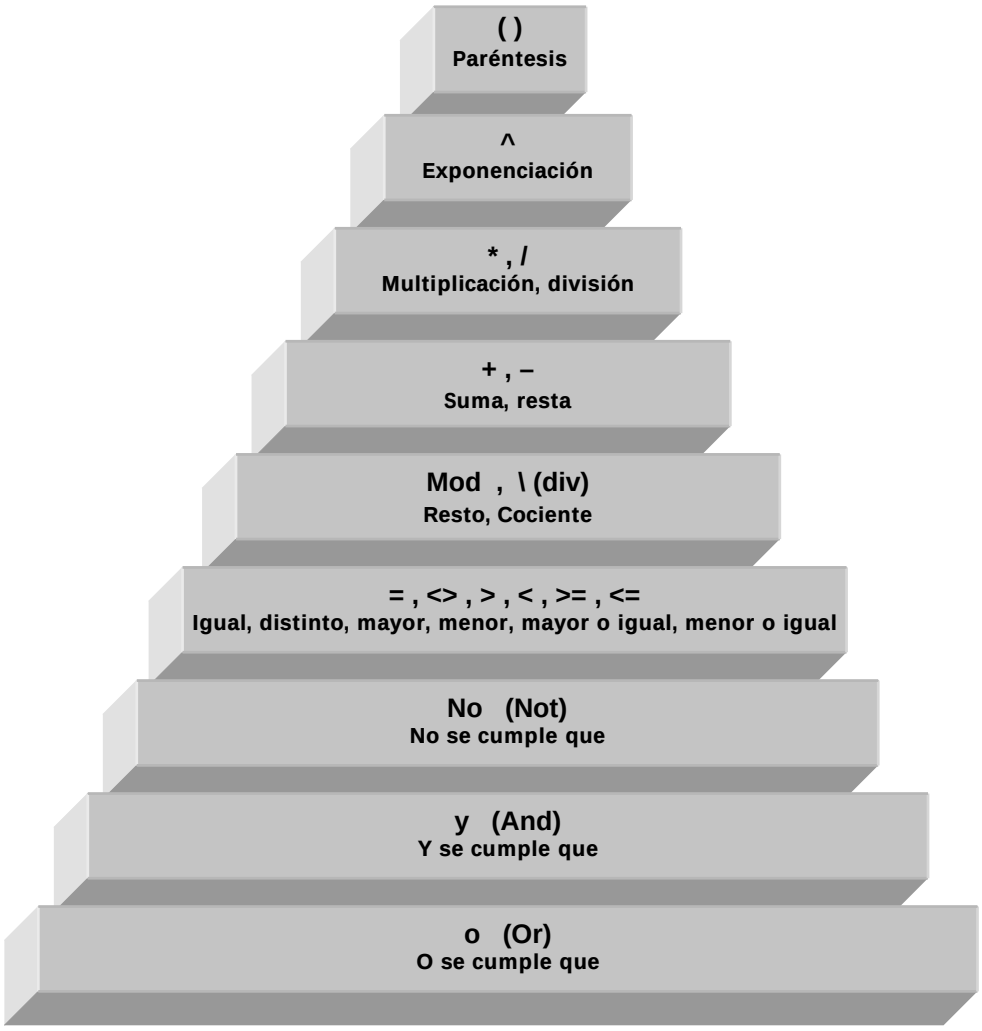
Si comparamos ‘avellana’ < ‘sandia’ obtenemos true.

Sin embargo, los códigos numéricos pueden generar resultados no previstos. Por ejemplo, ¿qué código numérico es menor, el de la a ó el de la A? Aún más, resulta que todos los códigos numéricos de mayúsculas son menores que los de minúsculas, con lo cual podemos obtener que 'Zulú' < 'avellano' devuelve true (cosa que a priori nos resultará ciertamente extraña).

Para comparar cadenas en base a un orden alfabético necesitaremos usar entonces otras técnicas que comentaremos más adelante.

ORDEN DE PRIORIDAD, PRELACIÓN O PRECEDENCIA

Los operadores lógicos y matemáticos tienen un orden de prioridad o precedencia. Este es un esquema general que indica el orden en que deben evaluarse en la mayoría de los lenguajes de programación:



Una expresión como A+B == 8 && A-B == 1 siendo A = 3 y B = 5 supondrá que se evalúa primero A+B que vale 8, luego se evalúa A-B que vale -2. Luego se evalúa si se cumple que la primera operación es cierta y luego si la segunda también es cierta, resultando que no, por lo que la expresión es falsa.

EJEMPLO

Veamos un ejemplo. Escribe este código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si quieres):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var a, b, c, d, eTexto; a = 3; b = 5; c = true; d = false; eTexto = '1';
alert ('Valor de verdad para a+b == 8 && a-b ==1 es: ' + (a+b == 8 && a-b ==1));
alert ('Valor de verdad para a+b == 8 && a-b ==-2 es: ' + (a+b == 8 && a-b ==-2));
alert ('Valor de verdad para c == d es: ' + (c==d));
alert ('Valor de verdad para c&&d es: ' + (c&&d));
alert ('Valor de verdad para c||d es: ' + (c||d));
alert ('Valor de verdad para !a es: ' + (!a));
alert ('Valor de verdad para eTexto === 1: ' + (eTexto === 1));
alert ('Valor de verdad para eTexto == 1: ' + (eTexto == 1));
alert ('Valor de verdad para Zapato < avellano es: ' + ('Zapato'<'avellano'));
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++,
un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="background-color:yellow;">Aquí otro párrafo de texto.
JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando pulsas sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo):

Valor de verdad para a+b == 8 && a-b ==1 es: false (Aceptar)
Valor de verdad para a+b == 8 && a-b ==-2 es: true (Aceptar)
Valor de verdad para c == d es: false (Aceptar)
Valor de verdad para c&&d es: false (Aceptar)
Valor de verdad para c||d es: true (Aceptar)
Valor de verdad para !a es: false (Aceptar)
Valor de verdad para eTexto === 1: false (Aceptar)
Valor de verdad para eTexto == 1: true (Aceptar)
Valor de verdad para Zapato < avellano es: true (Aceptar)

Importante: recordar que cuando se quieran realizar comparaciones de igualdad hay que usar el operador == y no el operador =.

EJERCICIO 1

Dadas las variables de tipo entero con valores A = 5, B = 3, C = -12 indicar si la evaluación de estas expresiones daría como resultado verdadero o falso:

a) A > 3

b) A > C

c) A < C

d) B < C

e) B != C

f) A == 3

g) A * B == 15

h) A * B == -30

i) C / B < A

j) C / B == -10

k) C / B == -4

l) A + B + C == 5

m) (A+B == 8) && (A-B == 2)

n) (A+B == 8) || (A-B == 6)

o) A > 3 && B > 3 && C < 3

p) A > 3 && B >= 3 && C < -3

Crea un script donde declares estas variables, les asignes valores y muestres por pantalla el valor de verdad que tienen cada una de las expresiones antes indicadas. ¿Coincide lo que se muestra con pantalla con lo que tú esperarías que se mostrara? Puedes comprobar si tus resultados son correctos consultando en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un script donde declares dos variables a y b y fuerces a que ambas contengan valor NaN. Ahora realiza las comparaciones a == b, a === b, a !== a ¿Qué resultados obtienes? ¿A qué crees que se debe este resultado?

Próxima entrega: CU01118E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EJEMPLOS JAVASCRIPT OPERADORES INCREMENTO Y DECREMENTO. ASIGNACIÓN Y ASIGNACIÓN COMPUESTA. (CU01118E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº18 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

OPERADORES EXTRA EN JAVASCRIPT

En JavaScript disponemos además de la asignación tracional basada en = de algunos operadores menos habituales como incremento unitario (++), decremento unitario (--), asignación compuesta de suma (+=) , asignación compueta de resta (-=) y asignación compuesta de multiplicación .

OPERADORES	DESCRIPCIÓN	EJEMPLO	EQUIVALE A
=	Asignación	a = 5	
+=	Suma lo indicado	a += b	a = a + b
-=	Resta lo indicado	a -= b	a = a - b
*=	Multiplica por lo indicado	a *= b	a = a * b
%=	Calcula el módulo por lo indicado	a %= b	a = a % b
++ (anterior)	Incremento unitario antes de operar	++a * 2	a = a + 1 a * 2
++ (posterior)	Incremento unitario después de operar	a++ * 2	a * 2 a = a + 1
-- (anterior)	Decremento unitario antes de operar	--a * 2	a = a – 1 a * 2
-- (posterior)	Decremento unitario después de operar	a-- * 2	a * 2 a = a – 1

Operadores en JavaScript

El operador = es el operador de asignación y hay que tener bien claro que no sirve para realizar comparaciones. Para realizar comparaciones ha de usarse == (es igual a) ó === (es estrictamente igual a). La asignación a = b se lee: “asigna a a el contenido de b”. Si b es una operación o expresión lógica, a almacenará el valor numérico resultado de la operación o el valor booleano resultado de evaluar la expresión lógica. Por ejemplo a = 3 > 5 implicará que a vale false porque 3 > 5 es falso.

++ y -- son sólo válidos para variables numéricas y sirven para incrementar una unidad el valor de la variable. Dependiendo de dónde se coloquen (antes o después de la variable) el resultado del cálculo puede diferir debido al momento en que se ejecuta la adición de la unidad.

Los operadores +=, -= y *= son formas abreviadas de escribir operaciones habituales.

Tener en cuenta que ++, --, +=, -= y *= son expresiones que siempre se aplican sobre variables. Por ejemplo no es válido escribir 2++ porque 2 no es una variable. Todas estas operaciones pueden sustituirse por otra equivalente más evidente. Muchos programadores prefieren no usar estos operadores porque hacen menos legible el código. A otros programadores les gusta usarlos porque les ahorra escribir. Nosotros preferimos no usarlos, pero es cierto que los puedes encontrar cuando tengas que revisar el código escrito por otra persona.

EJEMPLO

Veamos un ejemplo. Vamos a usar dos pequeñas novedades.

Una vez pedido un número al usuario usaremos nombreVariable = Number (nombreVariable); para indicarle a JavaScript que considere la variable como de tipo numérico y no de tipo texto. De esta forma al usar el operador + se ejecutará la operación de suma en lugar de la operación de concatenación de cadenas de texto.

Además usaremos la sentencia document.write('cadena de texto a introducir como HTML en el documento’); para introducir HTML en el documento. Hablaremos de esta sentencia más adelante, de momento sólo nos interesa saber que sirve para introducir código HTML en el documento.

Fíjate cómo en este ejemplo se usan tres scripts cuya ejecución no está en función de la ocurrencia de un evento como un click de un usuario, sino que se ejecutan automáticamente cuando carga la página web, en un determinado orden.

Escribe el siguiente código y guárdalo en un archivo de extensión html:

```
<html> <head> <title>Curso JavaScript aprenderaprogramar.com</title> <meta charset='utf-8'>
<style type='text/css'>
body {background-color: #FAEBD7; font-family: sans-serif; line-height: 1.3;}
div {border-style: solid; margin: 15px; padding: 10px; float: left;}
</style>
<script type='text/javascript'>
var a = prompt('Introduzca un número entero'); var b = prompt('Introduzca otro número entero');
a = Number(a) //Si no hacemos esto a es un texto y a+b concatena el texto
b = Number(b)
</script>
</head>
<body>
<div> <h2>Operadores de incremento</h2>
<script type='text/javascript'>
var valorInicial_a = a; document.write('Valores iniciales: a = ' + a + ', b = ' + b + '<br/> <br/>');
aumentar = ++a * b; document.write ('Operador ++ (anterior): ++a * b  == ' + aumentar +  '<br/>');
document.write ('(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = valorInicial_a; aumentar = a++ * b;
document.write ('Operador ++ (posterior): a++ * b  == ' + aumentar +  '<br/>');
document.write ('(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = valorInicial_a; //Restablecemos el valor que inicialmente tenía a
disminuir = --a * b; document.write ('Operador -- (anterior): --a * b == ' + disminuir +  '<br/>');
document.write ('(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = valorInicial_a;
disminuir = a-- * b;
document.write ('Operador -- (posterior): a-- * b == ' + disminuir +  '<br/>');
document.write ('(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = valorInicial_a;
</script>
</div>
<div>
<h2>Operadores de asignación compuestos</h2>
<script type='text/javascript'>
document.write('Valores iniciales: a = ' + a + ', b = ' + b + '<br/> <br/>');
inicio = a;
a += b;
document.write ( 'Asignación compuesta de suma: a += b equivale a = a + b<br/>');
document.write ( '(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = inicio; a -= b; document.write ( 'Asignación compuesta de resta: a -= b equivale a = a - b<br/>');
document.write ( '(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = inicio; a *= b;
document.write ( 'Asignación compuesta de multiplicación: a *= b equivale a = a * b<br/>');
document.write ( '(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = inicio; a /= b;
document.write ( 'Asignación compuesta de división: a /= b equivale a = a / b<br/>');
document.write ( '(Ahora el valor de a es: ' + a + ')<br/><br/>');
a = inicio; a %= b;
document.write ( 'Asignación compuesta de módulo: a %= b equivale a = a % b<br/>');
document.write ( '(Ahora el valor de a es: ' + a + ')<br/><br/>');
</script> </div> </body> </html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad.

El resultado esperado es que se muestre lo siguiente (compruébalo introduciendo 4 y 2 como números):

Operadores de incremento

Valores iniciales: a = 4, b = 2
Operador ++ (anterior): ++a * b == 10
(Ahora el valor de a es: 5)
Operador ++ (posterior): a++ * b == 8
(Ahora el valor de a es: 5)
Operador -- (anterior): --a * b == 6
(Ahora el valor de a es: 3)
Operador -- (posterior): a-- * b == 8
(Ahora el valor de a es: 3)

Operadores de asignación compuestos

Valores iniciales: a = 4, b = 2
Asignación compuesta de suma: a += b equivale a = a + b
(Ahora el valor de a es: 6)
Asignación compuesta de resta: a -= b equivale a = a - b
(Ahora el valor de a es: 2)
Asignación compuesta de multiplicación: a *= b equivale a = a * b
(Ahora el valor de a es: 8)
Asignación compuesta de división: a /= b equivale a = a / b
(Ahora el valor de a es: 2)
Asignación compuesta de módulo: a %= b equivale a = a % b
(Ahora el valor de a es: 0)

EJERCICIO 1

a) Describe paso a paso lo que hace el código anterior, indicando en qué orden se carga cada cosa (instrucciones HTML, instrucciones CSS, instrucciones JavaScript). Por ejemplo, ¿se carga el html <h2>Operadores de incremento</h2> en el instante en que solicitamos al navegador que muestre la página web? ¿En qué orden se ejecutan los scripts?

b) Introduce errores en los scripts de modo que estos no se ejecuten ¿Qué se visualiza en la página web?

c) Introduce 0 y 0 como números en el ejemplo anterior. ¿Cuántas veces el resultado de las operaciones es NaN? ¿Por qué?

Para comprobar tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01119E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

IF ELSE, IF ELSE IF JAVASCRIPT. CONDICIONALES DEL FLUJO O ESTRUCTURAS DE DECISIÓN. EJEMPLOS. EJERCICIOS. (CU01119E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº19 del Tutorial básico “JavaScript desde cero”.

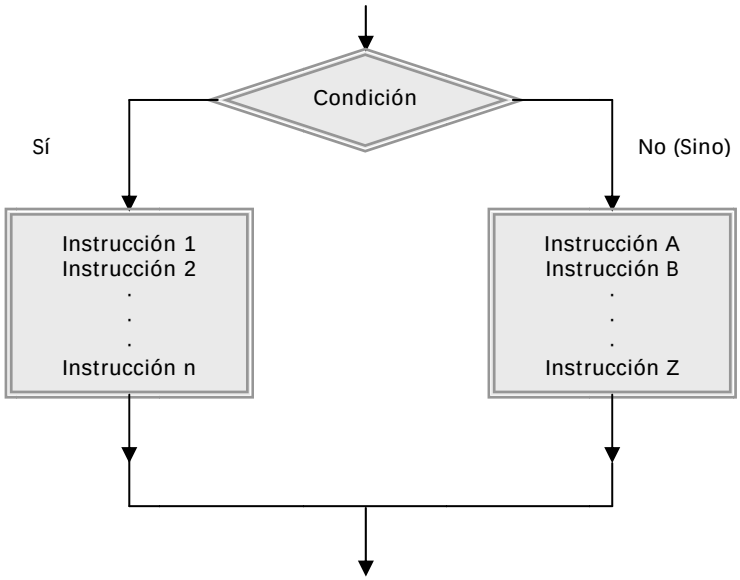
Autor: César Krall

ESTRUCTURA O ESQUEMA DE DECISIÓN EN JAVASCRIPT. IF ELSE , IF ELSE IF.

La instrucción if … else funciona de forma análoga a como lo hace en otros lenguajes de programación. Permite controlar qué procesos tienen lugar, típicamente en función del valor de una o varias variables, de un valor de cálculo o booleano, o de las decisiones del usuario. La sintaxis a emplear es:

```
if (condición) {
      instrucciones
} else {
      instrucciones
}
```

Esquemáticamente en forma de diagrama de flujo:



La cláusula else (no obligatoria) sirve para indicar instrucciones a realizar en caso de no cumplirse la condición. **JavaScript admite escribir un else y dejarlo vacío:** else { }. El else vacío se interpreta como que contemplamos el caso pero no hacemos nada en respuesta a él. Un else vacío no tiene ningún efecto y en principio carece de utilidad, no obstante a veces es usado para remarcar que no se ejecuta ninguna acción cuando se alcanza esa situación.

Cuando se quieren evaluar distintas condiciones una detrás de otra, se usa la expresión *else if { }*. En este caso no se admite *elseif* todo junto como en otros lenguajes. De este modo, la evaluación que se produce es: si se cumple la primera condición, se ejecutan ciertas instrucciones; si no se cumple, comprobamos la segunda, tercera, cuarta… n condición. Si no se cumple ninguna de las condiciones, se ejecuta el else final en caso de existir.

```
//if sencillo
if ( admitido == true) { alert ("Se ha admitido el valor"); }

//if else sencillo
      if ( admitido == true) {
      alert ("Se ha admitido el valor");
   } else {
      alert ("No se ha admitido el valor");
   }
//if con else if y cláusula final else
if (DesplazamientoX == 0 && DesplazamientoY == 1) {
      alert ("Se procede a bajar el personaje 1 posición");
   }
    else if (DesplazamientoX == 1 && DesplazamientoY == 0) {
      alert ("Se procede a mover el personaje 1 posición a la derecha"); }

   else if (DesplazamientoX == -1 && DesplazamientoY == 0) {
      alert ("Se procede a mover el personaje 1 posición a la izquierda");
   }
    else {
      alert ("Los valores no son válidos");
   }
```

La expresión dentro de paréntesis es una expresión booleana. **Llamamos expresión booleana a una expresión que solo tiene dos valores posibles:** verdadero (true) o falso (false).

Es importante distinguir la comparación que realizamos con el operador == de la asignación que realizamos con el operador =. Confundirlos nos generará errores que harán que el código JavaScript no se ejecute o problemas de lógica en el código. Recuerda que siempre que tengas que comparar con un operador, has de usar == ó === en lugar de =.

La condición a evaluar puede ser un simple nombre de variable. Por ejemplo:

if (antiop) {alert ('nombre se evaluó a verdadero');}

En este caso se comprueba el valor booleano (o equivalente booleano) de antiop. Si nombre es de tipo String y es la cadena vacía, su valor equivalente es falso y no se ejecutarán las instrucciones dentro del if. Si antiop es un número se considera que equivale a false si su valor numérico es 0, o que equivale a true si su valor numérico es distinto de cero (incluido NaN). Para cadenas de texto, la cadena vacía se considera equivale a false y cualquier otra cadena se considera que equivale a true.

Se admite omitir las llaves después de la condición si solo se va a incluir una sentencia a ejecutar. Por ejemplo: if (nombre) alert ('nombre se evaluó a verdadero');

Sin embargo, recomendamos incluir las llaves siempre después de un if porque hace el código más fácil de seguir y más claro.

EJEMPLO

Escribe el siguiente código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si lo deseas):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var nombre, edad;
nombre = prompt ('Por favor introduce tu nombre:');
if (nombre =="") { alert ('No has introducido ningún nombre'); }
else { alert ('Hola '+nombre + '. Bienvenido a esta página web.'); }
edad = prompt ('¿Cuál es tu edad?');
edad = Number(edad);
if (edad >3 && edad < 10) {alert ('Eres un niño.');}
else if (edad>=10 && edad <18) {alert ('Eres un jovencito.');}
else if (edad >=18 && edad < 90) {alert ('Eres mayor de edad.');}
else if (edad >=90) { alert ('Tienes muchos años encima...');}
else {alert ('No has introducido un valor válido de edad ( '+edad+')');}
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="background-color:yellow;">Aquí otro párrafo de texto. JavaScript es un lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando haces click sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo introduciendo Juan y 96 como entradas de usuario):

Por favor introduce tu nombre: Juan (Aceptar)
Hola Juan. Bienvenido a esta página web. (Aceptar)
¿Cuál es tu edad? : 96 (Aceptar)
Tienes muchos años encima... (Aceptar)

EJERCICIO

Crea un script que pida al usuario el diámetro de una rueda y su grosor (en metros) y a través de condicionales if realice las siguientes operaciones:

a) Si el diámetro es superior a 1.4 debe mostrarse el mensaje “La rueda es para un vehículo grande”. Si es menor o igual a 1.4 pero mayor que 0.8 debe mostrarse el mensaje “La rueda es para un vehículo mediano”. Si no se cumplen ninguna de las condiciones anteriores debe mostrarse por pantalla el mensaje “La rueda es para un vehículo pequeño”.

b) Si el diámetro es superior a 1.4 con un grosor inferior a 0.4, ó si el diámetro es menor o igual a 1.4 pero mayor que 0.8, con un grosor inferior a 0.25, deberá mostrarse el mensaje “El grosor para esta rueda es inferior al recomendado”

Ejecuta el código y comprueba sus resultados. Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01120E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

SWITCH JAVASCRIPT. SELECCIÓN ENTRE ALTERNATIVAS. DIAGRAMA DE FLUJO Y EJEMPLO DE APLICACIÓN. (CU01120E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº20 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

CONDICIONAL DE SELECCIÓN SWITCH EN JAVASCRIPT. EJEMPLO DE APLICACIÓN.

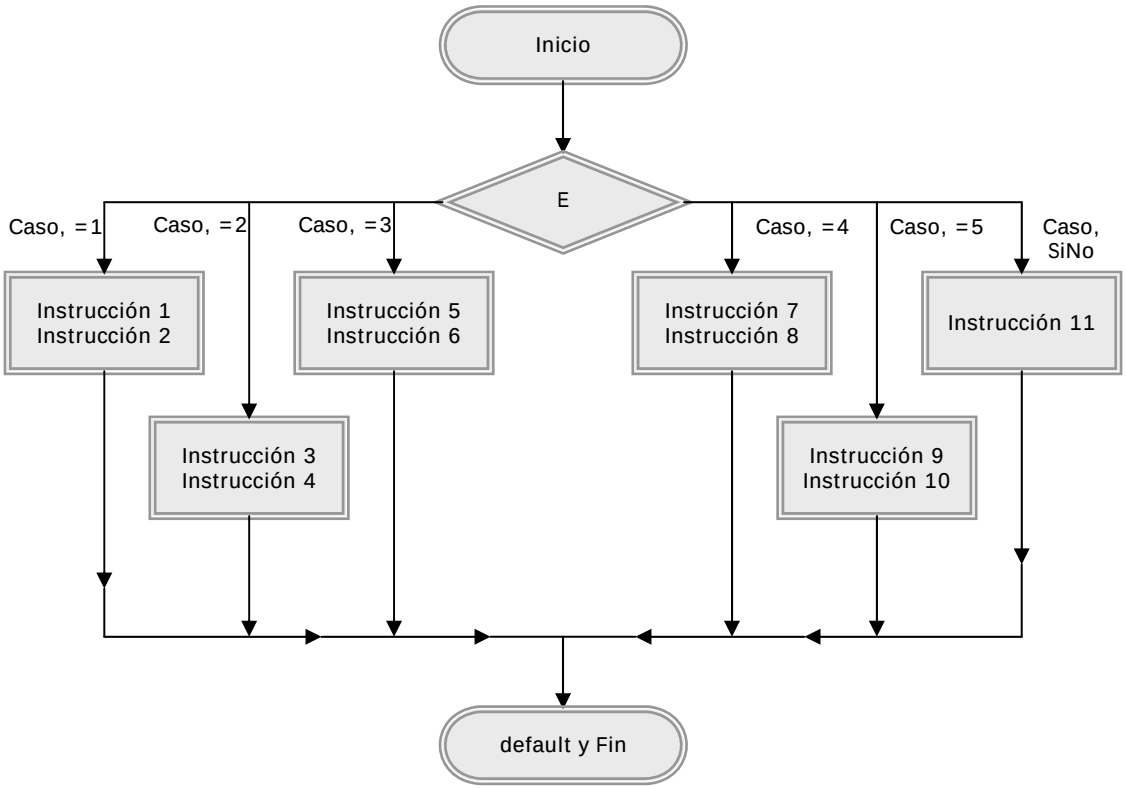
La instrucción switch es una forma de expresión de un anidamiento múltiple de instrucciones if ... else. Su uso no puede considerarse, por tanto, estrictamente necesario, puesto que siempre podrá ser sustituida por el uso de if. No obstante, a veces resulta útil al introducir eficiencia y mayor claridad en el código.

La sintaxis será (expresión será normalmente una variable cuyo contenido queremos evaluar, pero puede ser una operación matemática, una expresión booleana, etc.):

```
switch (expresión) {

    case valor1:
    instrucciones;
    break;

    case valor2:
    instrucciones;
    break;
    .
    .
    .
    default:
    sentencias;
    break;

}
```

```
switch (expresión) {

    case valor1:
    case valor2:
    case valor3:
    instrucciones;
    break;

    case valor4:
    instrucciones;
    break;
    .
    .
    .
    default:
    sentencias;
    break;

}
```

Esquemáticamente a modo de diagrama de flujo:



break es opcional (pero en general recomendable) y provoca que una vez encontrada una coincidencia termine la evaluación de casos. Si no se incluye, se evaluarían el resto de casos (ejecutándose si fueran ciertos). De no incluirse break, podría producirse que se ejecuten dos o más casos.

La cláusula default es opcional y representa las instrucciones que se ejecutarán en caso de que no se verifique ninguno de los casos evaluados. El último *break* dentro de un *switch* (en *default* si existe esta cláusula, o en el último caso evaluado si no existe *default*) también es opcional, pero lo incluiremos siempre para ser metódicos.

switch realiza las comparaciones usando el operador === (igual estricto). Se permite evaluar tanto números como cadenas de texto o valores booleanos. Switch solo permite evaluar valores concretos de la expresión: no permite evaluar intervalos (pertenencia de la expresión a un intervalo o rango) ni expresiones compuestas. Código de ejemplo:

```
//Ejemplo de uso switch JavaScript aprenderaprogramar.com
Function mostrarMensaje1() {
      switch (mes) {
         case 1:
         alert ("El mes es enero");
         break;
         case 2: alert ("El mes es febrero"); break;
         case 10: alert ("El mes es octubre"); break;
         default: alert ("El mes no es enero, febrero ni octubre"); break;
      }
   }
```

En algunos casos escribimos varias instrucciones en una línea y en otros una sola instrucción por línea. Ambas posibilidades son válidas.

El anterior código usa valores numéricos. También se admiten situaciones como switch (edad+1>18) donde la expresión devuelve un booleano, o case 'armario': alert ("Ha introducido armario"); break; donde se evalúa la igualdad con un texto.

EJEMPLO

Escribe el siguiente código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si lo deseas):

```
<html>
<head>
<title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var edad;
edad = prompt ('Por favor introduce edad:');
if (edad =="") { alert ('No has introducido edad'); }
edad = Number(edad);
      switch (edad) {
         case 0: alert ("Acaba de nacer hace poco. No ha cumplido el año"); break;
         case 18: alert ("Está justo en la mayoría de edad"); break;
         case 65: alert ("Está en la edad de jubilación"); break;
         default: alert ("La edad no es crítica"); break;
      }
}
</script>
</head>
<body>
<div>
<p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++,
un útil editor de texto">
<p onclick ="alert('Alerta JavaScript')" style="color: #D2691E;">Aquí otro párrafo de texto. JavaScript es un
lenguaje utilizado para dotar de efectos dinámicos a las páginas web.
</p>
</div>
</body>
</html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando haces click sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo introduciendo 33 como entrada del usuario):

Por favor introduce edad: 33 (Aceptar)
La edad no es crítica (Aceptar)

EJERCICIO

Considera estás desarrollando un script donde trabajas con tipos de motor (suponemos que se trata del tipo de motor de una bomba para mover fluidos). Crea una función denominada dimeTipoMotor() donde pidas el tipo de motor al usuario (indicando que los valores posibles son 1, 2, 3, 4) y a través de un condicional switch hagas lo siguiente:

a) Si el tipo de motor es 0, mostrar un mensaje indicando “No hay establecido un valor definido para el tipo de bomba”.

b) Si el tipo de motor es 1, mostrar un mensaje indicando “La bomba es una bomba de agua”.

c) Si el tipo de motor es 2, mostrar un mensaje indicando “La bomba es una bomba de gasolina”.

d) Si el tipo de motor es 3, mostrar un mensaje indicando “La bomba es una bomba de hormigón”.

e) Si el tipo de motor es 4,mostrar un mensaje indicando “La bomba es una bomba de pasta alimenticia”.

f) Si no se cumple ninguno de los valores anteriores mostrar el mensaje “No existe un valor válido para tipo de bomba”.

Ejecuta el código y comprueba sus resultados. Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01121E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ARRAYS JAVASCRIPT (ARREGLOS). DECLARACIÓN, INICIALIZACIÓN. ARRAY VACÍO O CON ELEMENTOS UNDEFINED. EJEMPLOS. (CU01121E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº21 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

ARRAYS JAVASCRIPT (ARREGLOS).

Al igual que en la mayor parte de los lenguajes de programación, en JavaScript podemos trabajar con arrays (también llamados arreglos, vectores o matrices). Los arrays son de gran importancia ya que permiten organizar series de datos que comparten el mismo nombre pero se diferencian por un índice.

DECLARACIÓN E INICIALIZACIÓN DE ARRAYS

La declaración de un array se hace de misma forma que se declara cualquier variable: var nombreDelArray;

El array adquiere condición de tal cuando la variable se inicializa con forma de array, bien como array con un contenido inicial o bien como array vacío:

```
var pais;
pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
```

O podemos hacerlo todo en una sola línea:

```
var pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
```

En este ejemplo decimos que hemos declarado un array de 8 elementos. Cada elemento tiene un índice, comenzando los índices por el número 0. Por tanto los 8 elementos del array anterior son: pais[0], pais[1], pais[2], pais[3], pais[4], pais[5], pais[6] y pais[7].

También podemos inicializar un array vacío de dos formas distintas:

```
var fruta = [];
var fruta = new Array();
```

Ambas expresiones tienen el mismo efecto: crean un array vacío. En este caso se entiende que se añadirán contenidos a posteriori. Por ejemplo fruta[3] = 'manzana';

ARRAYS CON ELEMENTOS SIN DEFINIR

Podemos dar valor a un elemento de un array sin que los anteriores elementos estén definidos como hemos hecho en el ejemplo, declarando el elemento de índice 3 sin haber definido los índices 0, 1 y 2. Los elementos no definidos toman valor “undefined”. En este ejemplo fruta[0] no ha sido definido por lo que si intentamos invocarlo su valor es undefined.

Para que JavaScript comprenda que una variable es un array hay que indicárselo explícitamente. Es válido: `var pais = []; pais[0] = 'Mexico';`

Sin embargo no es válido: var pais; pais[0] = 'Mexico'; ¿Por qué? Porque en este código no hemos declarado explícitamente que pais sea un array. Si no lo hemos declarado, no podemos empezar a usar índices asociados a la variable como si se tratara de un array. Si lo hacemos se entiende como un error en el código, lo que dará lugar a que JavaScript no se ejecute.

Un array puede inicializarse dejando ciertos índices sin un contenido definido. Para ello, se deja un espacio separado por comas como en este ejemplo:

```
var ciudad = ["Buenos Aires", , "Madrid"];
```

Esta declaración supone que el array tiene 3 elementos. Ciudad[0] que tiene valor Buenos Aires, ciudad[1] que tiene valor undefined, y ciudad [2] que tiene valor Madrid.

```
var capital = [ , 'Mexico D.F.', , 'Santiago'];
```

Esta declaración supone que el array capital tiene 4 elementos que son ciudad[0] con valor undefined, ciudad[1] con valor México D.F., ciudad[2] con valor undefined y ciuedad[3] con valor Santiago.

Una coma final no genera un nuevo elemento. Por ejemplo:

```
var capital = [ , 'Mexico D.F.', , 'Santiago', ];
```

En este caso la coma final es ignorada y el array sigue teniendo 4 elementos. Si queremos definir el array con un quinto elemento vacío se recomienda hacerlo así:

```
var capital = [ , 'Mexico D.F.', , 'Santiago', undefined];
```

También sería posible hacerlo dejando una coma final libre pero esto es menos recomendable y en algunos navegadores puede dar lugar a errores:

```
var capital = [ , 'Mexico D.F.', , 'Santiago', ,];
```

ACCESO A ÍNDICES NO EXISTENTES

En otros lenguajes de programación, intentar acceder a un índice de array inexistente devuelve un error, pero en JavaScript no es así. Si escribimos pais[40] cuando sólo hemos definido hasta el índice 7, el resultado es que pais[40] devuelve undefined. A diferencia de en otros lenguajes, los arrays en JavaScript no tienen un número fijo de elementos, sino que el número de elementos del array se ajusta dinámicamente según las necesidades.

USO DE LOS ARRAYS

Los arrays son de gran utilidad para automatizar los cálculos y procesos, por lo que siempre algo pueda expresarse con un nombre seguido de un índice, será preferible usar un array a usar variables independientes.

Los arrays por defecto siempre empiezan por el índice 0, pero en determinadas ocasiones algunos programadores prescinden de ese índice. Por ejemplo var mes = []; mes[0] = undefined; mes[1] = 'enero'; mes[2]='febrero'; mes[3]='marzo'; mes[4] = 'abril'; mes[5]='mayo'; mes[6] = 'junio';

```
mes[7]='julio'; mes[8]='agosto'; mes[9]='septiembre'; mes[10]='octubre'; mes[11]='noviembre'; mes[12]='diciembre';
```

Hemos definido mes[0] como undefined. ¿Por qué? Porque en general es preferible dejar constancia de que si mes[0] tiene valor undefined es porque el programador ha decidido que sea así, de esta manera no hay duda respecto a que pueda ser un error o un olvido.

También sería posible definir doce variables como mes1, mes2, mes3, mes4, mes5, mes6, mes7, mes8, mes9, mes10, mes11, mes12. Sin embargo esto es algo que desde el punto de vista de la programación es en general indeseable, ya que estas doce variables funcionan como variables que no tienen relación entre sí. Por tanto cuando tengamos que recorrer los meses no podremos hacerlo de forma automatizada usando índices, ya que aquí no existen índices (aunque el nombre de la variable lleve un número eso no significa que ese número sea un índice).

TIPADO DE LOS ARRAYS

En otros lenguajes de programación un array contiene un tipo de datos y se dice que el array es de ese tipo. Por ejemplo un array puede ser un array de enteros, o un array de cadenas de texto. Pero no pueden existir arrays que contengan indistintamente elementos de distinto tipo. Esto sí es posible en JavaScript, y por ello se dice que los arrays en JavaScript no tienen tipo. Por ejemplo se admite algo como esto: `var datos = ['Frío', 33, false, 'nube', -11.22, true, 3.33, 'variado'];`

En este array el elemento de índice cero es de tipo texto, el de índice 1 es un valor numérico, el elemento de índice tres es un valor booleano, etc.

Normalmente los arrays contendrán elementos de un tipo, por ejemplo valores numéricos, pero en ocasiones nos interesará que contengan elementos de distintos tipos.

JavaScript admite que los elementos de un array sean elementos de naturaleza compleja (objetos), o incluso que un elemento de un array sea otro array.

PROPIEDAD LENGTH DE LOS ARRAYS

La propiedad length de un array indica el número máximo de elementos en el array de acuerdo con el índice máximo existente (independientemente de que los elementos del array tengan contenido o no). Por ejemplo si definimos `var oficina = []; oficina[25] = 'Oficial José Vargas Corononado';` la propiedad oficina.length devuelve 26, número de elementos para el array (de 1 a 25 más el correspondiente al 0).

MÁS SOBRE LOS ARRAYS

Los arrays son elementos de gran importancia dentro de JavaScript y aún queda mucho que estudiar sobre ellos. De momento el conocimiento adquirido nos sirve para seguir avanzando, pero más adelante volveremos a explicar más cosas relacionadas con los arrays.

EJEMPLO

Escribe el siguiente código y guárdalo en un archivo de extensión html (puedes cambiar la ruta de la imagen si lo deseas):

```
<html> <head> <title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje1() {
var mes; mes =[]; mes[0] = undefined; mes[1] = 'enero'; mes[2]='febrero'; mes[3]='marzo'; mes[4] = 'abril'; mes[5]='mayo';
mes[6] = 'junio'; mes[7]='julio'; mes[8]='agosto'; mes[9]='septiembre'; mes[10]='octubre'; mes[11]='noviembre';
mes[12]='diciembre';
var pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
var msg, msg2; msg = 'El país de índice 2 es: ' + pais[2] + '\n\n';
var datos = ['Frío', 33, false, 'nube', -11.22, true, 3.33, 'variado'];
msg = msg + 'En el indice 1 de datos tenemos: ' + datos[1] + ' (numérico), p.ej. multiplica por 2: ' + (datos[1]*2) + '\n\n';
msg = msg + 'En el indice 2 de datos tenemos: ' + datos[2] + ' y es booleano\n\n';
msg = msg + 'En el índice 3 de datos tenemos: ' + datos[3] + ' y es un texto \n\n';
msg = msg + 'En el índice 40 de datos tenemos: ' + datos[40] + '\n\n';
var fruta = [];
msg = msg + 'En el índice 0 de fruta tenemos: ' + fruta[0] + ' y en el índice 30 '+  fruta[30] + '\n\n';
fruta[1] = 'pera'; fruta[2] = undefined; fruta[30] = 'manzana';
msg = msg + 'En el índice 0 de fruta tenemos: ' + fruta[0] + ' y en el índice 30 '+  fruta[30] + '\n\n';
alert (msg);
msg2 = 'Mostramos el array país: ' + pais + '\n\n';
msg2 = msg2 + 'Mostramos el array fruta: ' + fruta + '\n\n';
msg2 = msg2 + 'Mostramos el array datos: ' + datos + '\n\n';
msg2 = msg2 + 'Mostramos el array mes: ' + mes + '\n\n';
msg2 = msg2 + 'Intentamos sumar o concatenar arrays: ' + (pais + fruta) +'\n\n';
msg2 = msg2 + 'Valor length en el array pais es: ' + (pais.length) + ' y en el array fruta es ' + fruta.length +'\n\n';
alert (msg2);
var ejemplo = new Array(); alert('Contenido de ejemplo: '+ ejemplo); ejemplo [0]= 1; ejemplo [2]= 44;
alert('Contenido de ejemplo: '+ ejemplo);
}
</script>
</head>
<body> <div> <p>Aquí un párrafo de texto situado antes de la imagen, dentro de un div contenedor</p>
<img onclick="mostrarMensaje1()" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de
texto">
<p onclick ="alert('Alerta JavaScript')" style="color: #D2691E;">Aquí otro párrafo de texto. JavaScript es un lenguaje
utilizado para dotar de efectos dinámicos a las páginas web. </p> </div> </body> </html>
```

Visualiza el resultado y comprueba que la página web se muestra con normalidad y que JavaScript se ejecuta con normalidad cuando haces click sobre la imagen.

El resultado esperado es que se muestre lo siguiente (compruébalo):

El país de índice 2 es: Argentina
En el indice 1 de datos tenemos: 33 (numérico), p.ej. multiplica por 2: 66
En el indice 2 de datos tenemos: false y es booleano
En el índice 3 de datos tenemos: nube y es un texto
En el índice 40 de datos tenemos: undefined
En el índice 0 de fruta tenemos: undefined y en el índice 30 undefined
En el índice 0 de fruta tenemos: undefined y en el índice 30 manzana

Mostramos el array país: Mexico,España,Argentina,Chile,Colombia,Venezuela,Perú,Costa Rica
Mostramos el array fruta: ,pera,,,,,,,,,,,,,,,,,,,,,,,,,manzana
Mostramos el array datos: Frío,33,false,nube,-11.22,true,3.33,variado
Mostramos el array mes:
,enero,febrero,marzo,abril,mayo,junio,julio,agosto,septiembre,octubre,noviembre,diciembre
Intentamos sumar o concatenar arrays: Mexico,España,Argentina,Chile,Colombia,Venezuela,Perú,Costa Rica,pera,,,,,,,,,,,,,,,,,,,,,,,,,manzana
Valor length en el array pais es: 8 y en el array fruta es 31
Contenido de ejemplo:
Contenido de ejemplo: 1,,44

Fíjate en las siguientes cuestiones: un array en JavaScript puede contener elementos de distintos tipos. El contenido de un elemento no definido es undefined. Un array puede tener elementos intermedios no definidos. Cuando tratamos de mostrar por pantalla un array, se produce una conversión automática a texto. Esto no significa que el array se un texto ni un tipo String, sino simplemente que el intérprete hace una conversión automática para tratar de ofrecer un resultado. En el caso de elementos no definidos, al mostrarse el array se muestran espacios separados por comas.

EJERCICIO

Crea un script donde declares un array vacío denominado nombres. Pide al usuario tres nombres usando la sentencia prompt de JavaScript y almacena esos nombres como elementos 0, 1 y 2 del array. A continuación muestra el contenido del array por pantalla.

Ejecuta el código y comprueba sus resultados. Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01122E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FUNCIONES JAVASCRIPT. CONCEPTO. PARÁMETROS O ARGUMENTOS Y TIPOS. PASO POR VALOR. RETURN. EJEMPLOS. (CU01122E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº22 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

FUNCIONES JAVASCRIPT

Una función JavaScript es un fragmento de código que puede ser invocado para realizar tareas o devolver un resultado. Si has trabajado con otros lenguajes de programación el concepto te resultará familiar. Las funciones JavaScript son similares a lo que en otros lenguajes se denomina procedimientos, funciones o métodos.

Programar, usando un símil, podemos verlo como realizar un viaje por carretera. Cuando realizamos un viaje, aparte de la necesidad de definir el objetivo y estudiar la ruta del viaje (estructura del programa) podemos decir que: “En general, pero sobre todo para viajes complicados, conviene dividir el problema en subapartados”.

La estrategia del “divide y vencerás”... Es una de las estrategias más usadas en programación de ordenadores y, una vez más, abordaremos aquí el uso de esta filosofía compañera de viaje del programador. El concepto de función aplicado a la programación JavaScript es muy similar al aplicable a distintas facetas de la vida: un escritor divide su curso en capítulos y apéndices. Un profesor divide el contenido de la asignatura en temas. Un ingeniero divide el proyecto en partes como Memoria, Anejos, Pliego de Condiciones, Presupuesto y Planos. En una fábrica, organizan el trabajo dividiendo las áreas funcionales en recepción de materias primas, área de pre-proceso, área de proceso, área de post-proceso y área de carga y despacho de producto terminado.

De cara a la programación JavaScript, usaremos la división del código en funciones por ser una estrategia efectiva para resolver problemas complejos. Cada función será llamada para realizar su cometido en un orden establecido.

Además una función se puede llamar tantas veces como se desee, lo cual evita tener que repetir código y por otro lado permite que cuando haya que realizar una corrección únicamente tengamos que hacerla en la función concreta que se ve afectada.

Las funciones pueden recibir información para realizar su cometido, por ejemplo function suma (a, b) recibe dos elementos de información: a y b, o no recibirla por realizar un proceso que no necesita recibir información, por ejemplo function dibujarCirculo().

Otra característica interesante de las funciones es que permite abstraer los problemas. Supongamos que necesitamos una función que devuelva para un importe de una compra sin impuestos el importe con impuestos, y que a su vez el porcentaje de impuestos a aplicar depende del tipo de producto. Si un compañero nos facilita la función function obtenerImporteConImpuestos (importeSinImpuestos) no tenemos que preocuparnos del código de la función. Unicamente sabemos que invocando a la función obtendremos el importe con impuestos. De esta forma, podemos utilizar funciones que han creado otros programadores o funciones disponibles en librerías sin necesidad de conocer el código de las mismas. Decimos que las funciones son “cajas negras” que facilitan la abstracción porque no necesitamos ver en su interior, sólo nos interesan sus resultados.

De hecho, es posible que un programador use un código para una función function obtenerImporteConImpuestos (importeSinImpuestos) y otro programador use otro código para esa misma función sin que esto suponga ningún problema. Lo importante es que la función realice su cometido, no cómo lo realice ya que es frecuente que haya distintas maneras de hacer algo (aunque ciertamente hacer las cosas de diferente manera no debe significar que unas veces se hagan bien y otras mal: siempre deberían hacerse las cosas bien).

Una función en general debe tener un nombre descriptivo de cuál es su cometido y tener un cometido claro y único. No deben mezclarse tareas que no tengan relación entre sí dentro de una función.

FUNCIONES CON PARÁMETROS Y SIN PARÁMETROS

Una función JavaScript puede requerir ser llamada pasándole cierta información o no requerir información.

Definición de una función sin parámetros (no requiere información):

```
//Comentario descriptivo de qué hace la función
    function nombreDeLaFunción () {
//Código de la función
}
```

Definición de una función con parámetros (requiere información):

```
//Comentario descriptivo de qué hace la función
    function nombreDeLaFunción (param1, param2, …, paramN) {
//Código de la función
}
```

Una función puede recibir tantos parámetros como se deseen, aunque no sería demasiado razonable que una función reciba más de cuatro o cinco parámetros.

Los parámetros que se le pasan a la función pueden ser:

a) Valores simples a los que se denomina literales: por ejemplo 554, true ó ‘aldea’.

b) Variables que contienen un número, un texto o un valor booleano.

c) Objetos de naturaleza compleja, como arrays y otros tipos de objetos que veremos más adelante.

Cuando una función recibe un parámetro dicho parámetro funciona como si se tratara de una variable disponible para la función inicializada con el valor que se le pasa a la función.

Veamos un ejemplo:

```
function mostrarImporteConImpuestos(importeSinImpuestos) {
var importeConImpuestos; importeConImpuestos = importeSinImpuestos * 1.21;
msg = 'Importe antes de impuestos: '+ importeSinImpuestos + '\n\n';
alert(msg + 'Importe con impuestos: '+ importeConImpuestos + '\n\n');
}
```

Aquí vemos dos cosas de interés: el parámetro que recibe la función no tiene un tipo de datos explícito. El tipo de datos es "inferido" por el intérprete JavaScript.

Por otro lado, el parámetro está disponible dentro de la función con el valor con el que haya sido invocado. Por ejemplo onclick="mostrarImporteConImpuestos(100)" hará que importeSinImpuestos valga 100 porque ese es el valor con el que se invoca.

Cuando una función tiene varios parámetros, se debe invocar escribiendo su nombre seguido de los parámetros en el orden adecuado.

FUNCIONES QUE DEVUELVEN UN RESULTADO. RETURN.

Una función JavaScript puede devolver un resultado si se introduce la sentencia return resultado; donde resultado es aquello que queremos devolver (normalmente una variable que contiene un valor numérico, de texto o booleano, pero también podrían ser objetos con mayor complejidad como un array).

Una vez se llega a la sentencia return se produce la devolución del resultado y se interrumpe la ejecución de la función. Por ello la sentencia return será normalmente la última instrucción dentro de una función.

Definición de una función sin parámetros que devuelve un resultado:

```
//Comentario descriptivo de qué hace la función
    function nombreDeLaFunción () {
//Código de la función
return resultado;
}
```

Definición de una función con parámetros que devuelve un resultado:

```
//Comentario descriptivo de qué hace la función
    function nombreDeLaFunción (param1, param2, …, paramN) {
//Código de la función
return resultado;
}
```

Una función sólo devolverá un resultado y normalmente sólo tendrá una sentencia return, aunque si hay sentencias condicionales como if, puede haber varias sentencias return: una sentencia return para cada sentencia condicional.

Si además del resultado la función incluye código que implique acciones como mostrar un mensaje por pantalla, se ejecutará el código a la vez que se devuelve el resultado.

Veamos un ejemplo:

```
function obtenerImporteConImpuestos(importeSinImpuestos) {
var importeConImpuestos; importeConImpuestos = importeSinImpuestos * 1.21;
return importeConImpuestos;
}
```

Un ejemplo de uso de esta función sería:

onclick="alert('Calculado lo siguiente para producto de precio 100: importe con impuestos vale ' + obtenerImporteConImpuestos(100));"

Aquí vemos cómo al invocar la función ésta devuelve un resultado que se coloca allí donde se encuentra la llamada a la función (en este ejemplo el resultado se coloca dentro de una sentencia para mostrar un mensaje por pantalla. Aquí el resultado es numérico, pero el intérprete lo transformará automáticamente en texto para mostrarlo por pantalla).

También será frecuente almacenar el resultado en una variable, por ejemplo: var importeConImp = obtenerImporteConImpuestos(100);

El resultado que devuelve una función puede ser:

a) Un valor simple (literal): por ejemplo 554, ó true ó 'aldea'.

b) Una variable que contienen un número, un texto o un valor booleano.

c) Un objeto de naturaleza compleja, como arrays y otros tipos de objetos que veremos más adelante.

El resultado que devuelve una función no tiene un tipo de datos explícito. El tipo de datos es "inferido" por el intérprete JavaScript.

LLAMADAS A FUNCIONES DESDE OTRAS FUNCIONES

Una función puede llamar a otra función simplemente escribiendo su nombre y los parámetros que sean necesarios. Ejemplo:

```
function mostrarImporteConImpuestos2(importeSinImpuestos) {
var msg; msg = 'Ejemplo. Importe antes de impuestos: '+ importeSinImpuestos + '\n\n';
alert(msg + 'Importe con impuestos: '+ obtenerImporteConImpuestos(importeSinImpuestos) + '\n\n');
}
```

En esta función en vez de realizarse el cálculo del importe con impuestos, se invoca otra función que es la que se encarga de realizar el cálculo y devolver el valor correspondiente.

PASO DE PARÁMETROS A FUNCIONES

Hay dos formas comunes de pasar parámetros a funciones en programación: por valor, que implica que si se pasa una variable sus cambios sólo son conocidos dentro de la función, o por variable, que implica que si se pasa una variable ésta puede ser modificada por la función y sus cambios ser conocidos fuera de la función. JavaScript trabaja con paso de parámetros por valor, lo que implica que la variable pasada como parámetro funciona como una variable local a la función: si el parámetro sufre cambios, estos cambios sólo son conocidos dentro de la función. La variable “verdadera” no puede ser modificada.

PASO DE UN NÚMERO DE PARÁMETROS INCORRECTO

Si se pasan más parámetros de los necesarios, JavaScript ignorará los parámetros sobrantes. Si se pasan menos parámetros de los necesarios, JavaScript asignará valor undefined a los parámetros de los que no se recibe información y se ejecutará sin que surja ningún mensaje de error (aparte de los posibles resultados extraños que esto pudiera ocasionar).

EJERCICIOS

1. Crea un script donde declares una función obtenerImporteConImpuestos que reciba dos parámetros: el importe sin impuestos (numérico) y el tipo de producto (numérico entero). La función debe mostrar por pantalla el importe sin impuestos más el 21% si el tipo de producto es 1, ó el importe sin impuestos más el 10% si el tipo de producto es 2, ó el importe sin impuestos más el 5% si el tipo de producto es 3.

Ejemplo: obtenerImporteConImpuestos(100, 1) debe mostrar: Para un importe sin impuestos de 100 y tipo de producto 1 el resultado de importe con impuestos es 121. obtenerImporteConImpuestos(100, 2) debe mostrar: Para un importe sin impuestos de 100 y tipo de producto 2 el resultado de importe con impuestos es 110.

2. Crea un script donde declares una función obtenerImporteConImpuestos2 que reciba un parámetro: el importe sin impuestos (numérico). La función debe devolver un array con valor undefined para el índice 0, el importe sin impuestos más el 21% para el índice 1, el importe sin impuestos más el 10% para el índice 2, ó el importe sin impuestos más el 5% para el índice 3. Invoca la función haciendo que se muestre el contenido del array por pantalla.

obtenerImporteConImpuestos(100) debe devolver: resultado[0] = undefined, resultado[1] = 121, resultado[2] = 110, resultado[3] = 105. Por pantalla se debe mostrar: Para precio sin impuestos 100 si el producto es tipo 1 el importe es 121, si el producto es tipo 2 el importe es 110 y si el producto es tipo 3 el importe es 105.

Ejecuta el código y comprueba sus resultados. Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01123E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DOM O DOCUMENT OBJECT MODEL JAVASCRIPT. ¿QUÉ ES? ¿PARA QUÉ SIRVE? EL W3C. ARBOL DE NODOS. PARENT Y CHILD. EJEMPLOS (CU01123E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº23 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DOM. DOCUMENT OBJECT MODEL.

Una de las mayores potencialidades de JavaScript es que puede manipular cualquier elemento de una página web y modificar su contenido, su tamaño, su color, su posición... e incluso hacer aparecer y desaparacer elementos. Para hacer esto posible necesitamos identificar con precisión cada elemento de una página web para poder indicarle a JavaScript sobre qué elemento debe operar.

Para permitir que los lenguajes de programación pudieran extraer información o manipular cualquier elemento de una página web era necesario definir de alguna manera qué tipos de elementos conforman una página web, cómo nombrarlos y cómo se relacionan entre sí. Inicialmente esto resultaba bastante problemático, ya que no existía un estándar o especificación oficial sobre cómo debía realizarse esto.

Para evitar los problemas de falta de estandarización, un organismo internacional (el W3C) definió un estándar denominado DOM ó Document Object Model (Modelo de objetos para representar documentos) que define qué elementos se considera que conforman una página web, cómo se nombran, cómo se relacionan entre sí, cómo se puede acceder a ellos, etc.

EL WORLD WIDE WEB CONSORTIUM O W3C

DOM es un estándar cuya definición ha sido creada por el World Wide Web Consortium, cuyas siglas son W3C. Se trata de un consorcio internacional que produce especificaciones, recomendaciones, manuales y herramientas en relación al desarrollo de internet. El W3C es un organismo que cuenta entre sus principales promotores al MIT (Massachussets Institute of Technology, USA), el ERCIM (European Research Consortium for Informatics and Mathematics, participado por numerosos países), la Keio University (Japón) y la Beihang University (China).

Uno de los objetivos principales del W3C es generar estándares: documentos donde se definen las sintaxis de lenguajes y protocolos que intervienen en el desarrollo de internet. El objetivo es promover que las empresas, instituciones y personas que participan o trabajan en desarrollos web utilicen un mismo lenguaje y se pongan de acuerdo a la hora de generar software y productos relacionados con internet. ¿Por qué decimos que “la raíz” de una página web es document y webPage ó internetDocument? Porque esta forma de nombrar y organizar las páginas web ha sido definida por el W3C de esta manera y todas (o casi todas) las empresas, instituciones y personas lo han aceptado.

No siempre lo que propone el W3C es aceptado. El W3C emite propuestas, no leyes de obligado cumplimiento porque no tiene capacidad legal para ello. Hay otras instituciones o grandes empresas que también hacen propuestas o tienen criterios propios, y en ocasiones esas propuestas o criterios alternativos hacen que haya distintos grupos de trabajo y distintos estándares o forma de funcionamiento del software.

Por último, hay que tener en cuenta que el W3C está formado por un equipo de personas que también cometen errores y que "lo que dice el W3C" no siempre tiene por qué ser lo mejor ni lo más usado. No obstante, hoy día el W3C es la principal institución de referencia a la hora de crear y difundir estándares relacionados con los desarrollos web, entre ellos el estándar CSS y el estándar DOM.

LAS VERSIONES DE DOM

El W3C trabaja continuamente para mejorar el DOM, corrigiendo errores e incorporando nuevas funcionalidades. Antes de llegar a una especificación o recomendación oficial se trabajan numerosos borradores que son sometidos a revisión y corrección. Cuando se alcanza un relativo grado de acuerdo entre los miembros del W3C se libera lo que se denomina una recomendación oficial de DOM ó versión a modo de propuesta para su uso y aplicación por todas las empresas, instituciones y personas.

Las versiones de DOM a lo largo de la historia han sido*:

- Dom Level 1: publicada en 1998.
- Dom Level 2: publicada en 2000.
- Dom Level 3: publicada en 2004.
- Dom Level 4: se esperaba como recomendación oficial en 2016.

*Las fechas indicadas son sólo orientativas, la realidad es que una version no aparece un día, sino que tiene un largo proceso de desarrollo que a veces dura años.

Nosotros nos atendremos a la versión de DOM de más amplia difusión en cada momento. No nos interesa tanto conocer la especificación oficial al completo, sino entender cómo está concebido y para qué sirve el DOM.

Por otro lado, hay que tener en cuenta que "seguir con exactitud" una especificación oficial no significa que nuestra web vaya a funcionar perfectamente, debido a que no todos los navegadores reconocen todas las propiedades o sintaxis que se definen como recomendación oficial. También puede suceder que un navegador sí reconozca la sintaxis pero no ofrezca el mismo resultado que otro, lo cual da lugar a problemas en la visualización de páginas web.

Conseguir buenos resultados con JavaScript pasa por estar al día de la especificación del W3C pero también por seguir las novedades de la web, de los navegadores y siendo prácticos, por hacer muchas pruebas y comprobaciones con distintos navegadores o herramientas específicas para este fin.

REPRESENTACIÓN DOM DE UNA PÁGINA WEB

Decimos que una página web es un documento HTML. Este documento puede ser representado de diferentes maneras:

a) Representación web: como una página web en un navegador donde vemos imágenes, texto, colores, etc.

b) Representación texto: como un texto plano (código HTML) que podemos visualizar en cualquier editor de textos como el bloc de notas de Windows ó Notepad++ ó cualquier otro.

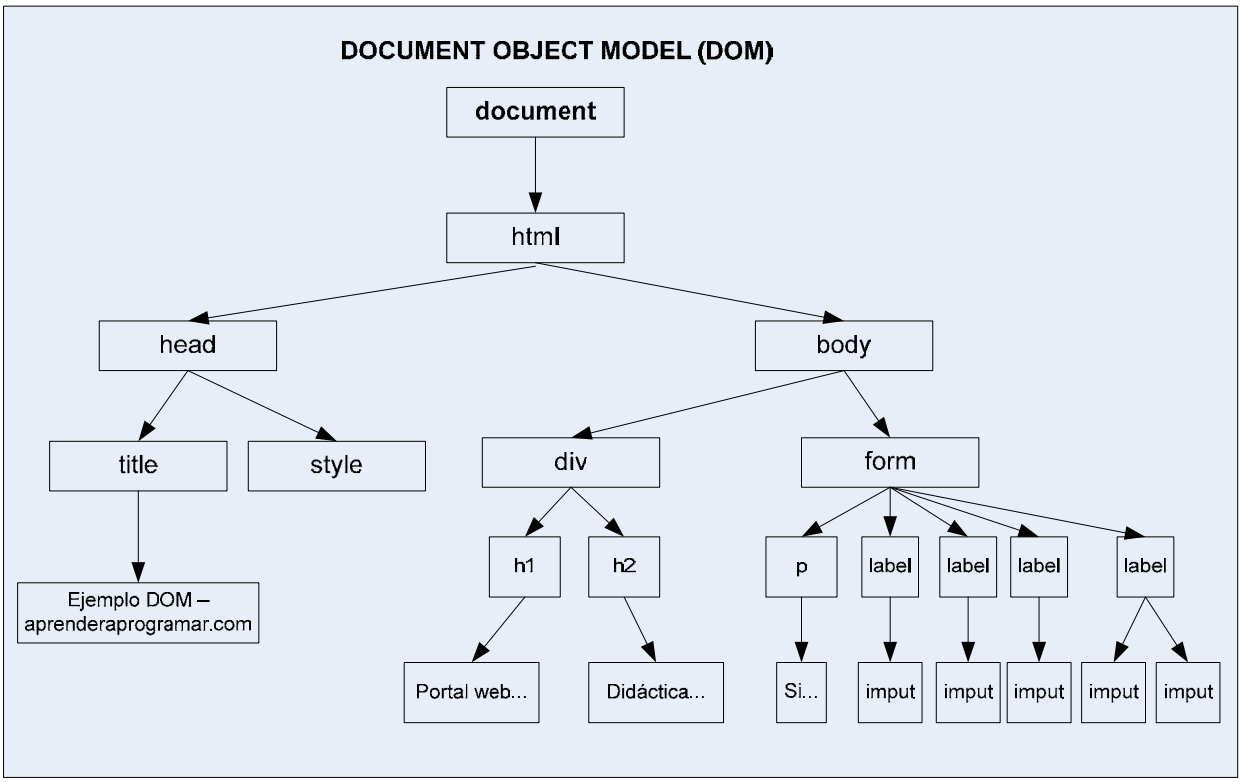
c) Representación DOM: como un árbol donde los elementos de la página web están organizados jerárquicamente, con nodos superiores (nodos padre o parent) y nodos que derivan de los nodos padre (nodos hijo o child).

Veamos un ejemplo de representación DOM. Escribe el siguiente código y guárdalo en un archivo de extensión html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>

<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {background-color:yellow; font-family: sans-serif;}
label {color: maroon; display:block; padding:5px;}
</style>
</head>

<body>
<div id="cabecera">
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>

<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="accion.html">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<label>
<input type="submit" value="Enviar">
<input type="reset" value="Cancelar">
</label>
</form>

</body>

</html>
```

La anterior representación se corresponde con la representación del documento como texto. Visualiza el resultado. La imagen que vemos en el navegador se corresponde con la representación del documento como página web en un navegador.

La representación del documento conforme al estándar del DOM sería (de forma aproximada) esta:



La representación anterior es solo aproximada: no nos va a interesar representar una página web conforme al DOM, simplemente queremos conocer cómo se estructura una página web conforme al DOM para saber cómo podemos acceder a sus elementos y manipularlos usando JavaScript (u otro lenguaje).

El DOM no es parte de JavaScript, de hecho puede ser utilizado por otros lenguajes de programación. No obstante, el DOM está íntimamente ligado a JavaScript ya que JavaScript lo utilizará con profusión para acceder y modificar las páginas web dinámicamente.

Decimos que conforme al DOM la página web se representa como un árbol de nodos, interconectados y relacionados de acuerdo con una jerarquía.

Como nodo principal, inicial o padre de todos los nodos en el árbol de nodos de una página web se encuentra un nodo especial cuyo nombre es document.

El nodo raíz tiene como nodo hijo al nodo html, que representa a todo lo contenido dentro de las etiquetas <html> … <html>.

El nodo html tiene dos hijos: head y body. Sucesivamente cada nodo tiene uno o varios hijos hasta que se llega a un punto terminal donde no existen más hijos.

La construcción del árbol de nodos es realizada automáticamente por los navegadores de acuerdo con las reglas internas que tienen definidas. Por ejemplo una regla interna dice que una etiqueta define un nodo, por ejemplo una etiqueta <p> define un nodo, y a su vez el contenido de texto de esa etiqueta define otro nodo.

Una página web avanzada supone la construcción de un árbol de nodos que puede constar de miles de nodos y cambiar continuamente a medida que el usuario realiza la navegación a través de diferentes páginas web. Todo este proceso es realizado “en segundo plano” por el navegador y dada la potencia de los computadores actuales, es un proceso muy rápido.

No todos los navegadores construyen el árbol de nodos de igual manera, ni siquiera todos construyen el mismo árbol de nodos (lo cual puede resultar problemático al dar lugar a distintos resultados cuando ejecutemos JavaScript). La mayor parte de los navegadores siguen el estándar W3C para construir el árbol de nodos, pero algunos navegadores no lo siguen con exactitud o introducen algunas extensiones adicionales específicas de ese navegador.

Un nodo constituye un elemento complejo que suele llevar información asociada como veremos más adelante.

A través de JavaScript podremos acceder a nodos con instrucciones del tipo document.getElementById("menu"); (acceder a un elemento por su id), pero también podremos crear nodos (por ejemplo document.createElement("h1"); ó document.body.appendChild(heading);). Todo esto es posible gracias a la existencia del DOM.

EJERCICIO

Crea una página web html que conste de las etiquetas html, head, body. Dentro de body incorpora dos div: uno que contenga una etiqueta h1 con el texto “Curso JavaScript aprenderaprogramar.com” y otro que contenga tres párrafos que contengan: Párrafo 1, Párrafo 2 y Párrafo 3. Crea la representación del árbol de nodos conforme al DOM para este documento. Para comprobar si tu respuesta es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01124E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

TIPOS DE NODOS DEL DOM: DOCUMENT, ELEMENT, TEXT, ATTRIBUTE, COMMENT. ARBOL DE NODOS PARA JAVASCRIPT (CU01124E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº24 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

TIPOS DE NODOS EN EL DOM

Un navegador web crea una representación interna del código HTML como un árbol de nodos y a esto lo denominamos DOM (Document Object Model). Los nodos se pueden clasificar en diferentes tipos, dentro de los cuales los principales tipos son: document, element, text y "otros".

Los principales tipos de nodos en el DOM son:

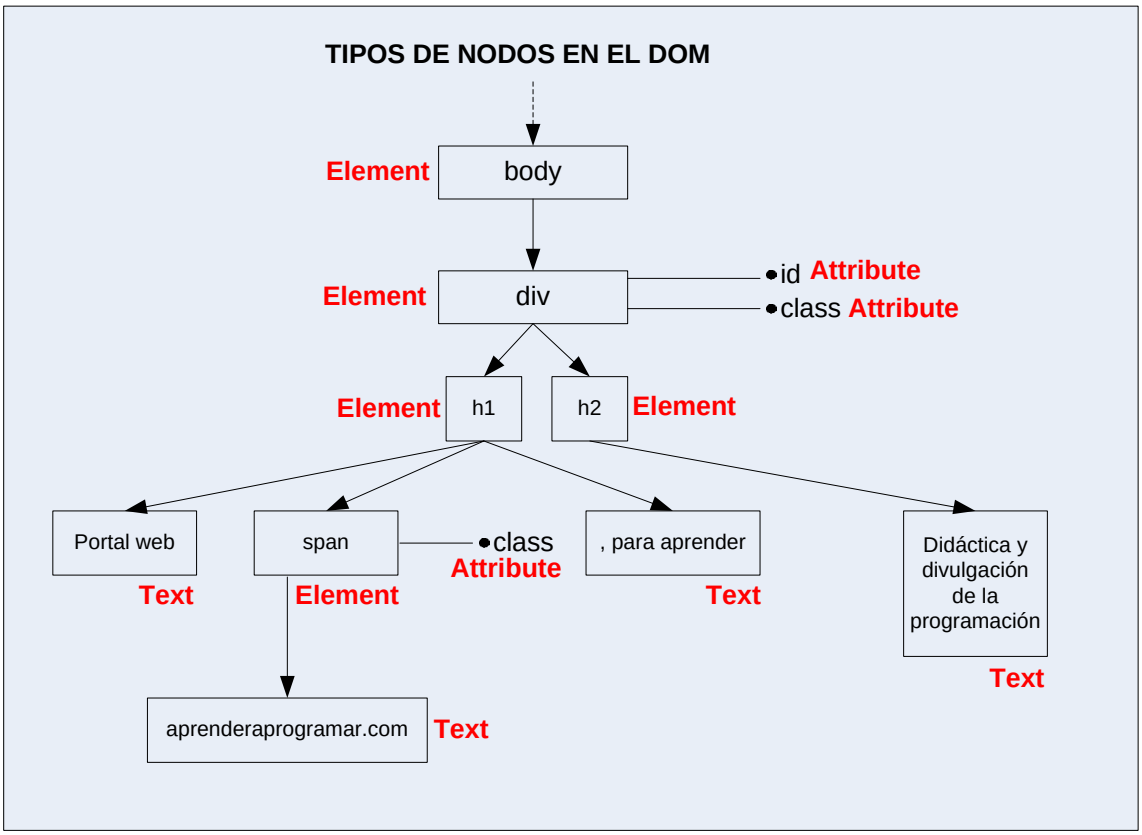
- **Document**: el nodo document es el nodo raíz, a partir del cual derivan el resto de nodos.
- **Element**: son los nodos definidos por etiquetas html. Por ejemplo una etiqueta div genera un nodo. Si dentro de ese div tenemos tres etiquetas p, dichas etiquetas definen nodos hijos de la etiqueta div.
- **Text**: el texto dentro de un nodo element se considera un nuevo nodo hijo de tipo text (texto). Los navegadores también crean nodos tipo texto sin contenido para representar elementos como saltos de línea o espacios vacíos.
- **Attribute**: los atributos de las etiquetas definen nodos, aunque trabajando con JavaScript no los veremos como nodos, sino que lo consideramos información asociada al nodo de tipo element.
- **Comentarios y otros**: los comentarios y otros elementos como las declaraciones doctype en cabecera de los documentos HTML generan nodos.

Existen más tipos de nodos en el DOM, pero de uso más infrecuente.

Veamos un ejemplo de un fragmento de código:

```
<body>
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>
</body>
```

Su representación indicando los tipos de nodos sería:



En la terminología de árboles, diremos que en este ejemplo el nodo body es padre (parent) y tiene un hijo (child) que es el nodo div. El nodo div tiene 2 hijos: h1 y h2. Por tanto h1 es el primer hijo (firstChild) del nodo div. A su vez el nodo h2 es hermano (sibling) de h1. El nodo h1 tiene 4 nodos hijos. El nodo de tipo texto aprenderaprogramar.com es un nodo hoja o nodo que no tiene hijos.

EJERCICIO

Crea el árbol de nodos DOM indicando los tipos de nodos y atributos para este código:

```
<body>
<div id="menu" class="tenue">
<p> Bienvenidos </p>
<h1>Portal web aprenderaprogramar.com</h1>
<img name ="lagarto" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++ editor de texto">
</div>
</body>
```

Para comprobar si tu respuesta es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01125E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT CHILDNODES, PARENTNODE, FIRSTCHILD, LASTCHILD, NEXTSIBLING, CHILDREN.LENGTH , CHILDELEMENTCOUNT (CU01125E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº25 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

ACCESO A NODOS DEL DOM

Los navegadores web representan la página web en forma de árbol de nodos, por tanto tendremos que saber cómo acceder a dichos nodos para crear efectos dinámicos (cambios en los nodos) con JavaScript. Para ello JavaScript usa términos como childNodes, nodeType, nodeName, nodeValue, firstChild, lastChild, parentNode, nextSibling, previousSibling, etc. que vamos a estudiar.

Algunas de las palabras clave de las que disponemos en JavaScript para acceder a los nodos del DOM son las que enumeramos a continuación (veremos un ejemplo de código después):

Palabra clave	Significado	Ejemplo aprenderaprogramar.com
parentNode	Nodo padre de un nodo	...childNodes[1].childNodes[3].parentNode
childNodes	Array conteniendo los hijos de un nodo	document.childNodes[1].childNodes[1]
firstChild	Primer hijo de un nodo (empezando por la izquierda)	document.firstChild
lastChild	Ultimo hijo de un nodo (el más a la derecha)	document.childNodes[1].lastChild
nextSibling	Próximo nodo hermano (situado a la derecha)	document.childNodes[1].nextSibling
previousSibling	Anterior nodo hermano (situado a la izquierda)	...childNodes[2].childNodes[0].previousSibling

Palabra clave	Significado	Ejemplo aprenderaprogramar.com
nodoAccedido.children.length	Permite conocer el número de nodos hijo de un nodo	document.childNodes[1].children.length
nodoAccedido.atributo	Permite acceder al atributo de un nodo	...childNodes[1].childNodes[3].name

Palabra clave	Significado	Ejemplo aprenderaprogramar.com
nodeName	Etiqueta del nodo, como texto en mayúsculas (por ejemplo H1, DIV, SPAN…)	…childNodes[2].childNodes[0].nodeName
nodeType	Número que identifica el tipo de nodo (9 para document, 1 para element, 3 para text, 8 para comment)	document.nodeType
nodeValue	Contenido en forma de texto de un nodo de tipo text o de un nodo de tipo comment	…childNodes[2].childNodes[0].nodeValue

Para los nodos de tipo elemento hay invocaciones equivalentes (mismo efecto que la invocación general para un nodo cualquiera, pero sólo aplicables a nodos tipo elemento):

Invocación general	Equivalente para nodos tipo element
firstChild	firstElementChild
lastChild	lastElementChild
nextSibling	nextElementSibling
previousSibling	previousElementSibling
children.length	childElementCount

La mayor parte de los navegadores reconocen todas estas palabras claves, pero pueden existir navegadores (en especial los más antiguos) que no reconozcan algunas de ellas.

Como veremos más adelante, podemos acceder a un nodo usando JavaScript para hacer cambios dinámicos en el nodo. También podemos acceder a un nodo simplemente para mostrar algo por pantalla o recuperar alguna información. Por ejemplo:

```
Alert ('El nombre del nodo es: ' + document.childNodes[1].nodeName +' y el tipo del nodo es: '+ document.childNodes[1].nodeType + '\n\n');
```

Esta instrucción nos mostraría por pantalla el valor de nodeName y nodeType para el segundo nodo hijo de document (dado que el primero es document.childNodes[0]). En caso de que tratemos de invocar un nodo que no exista, se producirá un error y JavaScript no se ejecutará. No aparecerá ningún mensaje de error por pantalla, simplemente no se ejecuta JavaScript.

También podemos almacenar la referencia a un nodo en una variable. Por ejemplo:

```
var nodoBody = document.childNodes[1].childNodes[2];
```

Aquí estamos asignando el nombre de variable “nodoBody” a un nodo (suponemos que document.childNodes[1].childNodes[2] es una referencia válida a un nodo). Ahora podremos hacer uso de este nombre de variable como equivalente al nodo, por ejemplo:

```
alert ('Segundo hijo de nodo body es: ' + nodoBody.childNodes[1].nodeName +'\n\n');
```

¿Qué tipo de datos corresponde a la variable nodoBody? Cuando hablamos de tipos de datos en JavaScript dijimos que existían datos primitivos y objetos. En este caso, la variable es de tipo objeto. Los objetos podemos verlos como "algo" (estructura de datos) de naturaleza compleja. Ya hemos visto que los arrays podían ser considerados objetos especiales y lo mismo podemos decir de los nodos del DOM.

En los próximos epígrafes del curso veremos ejemplos de código para acceder a los nodos y modificarlos, de momento nos es suficiente con irnos familiarizando con la sintaxis y los conceptos básicos relacionadso con el DOM.

EJERCICIO

Crea el árbol de nodos DOM para el siguiente código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo DOM - aprenderaprogramar.com</title><meta charset="utf-8">
</head>
<body>Texto en body
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<img name ="lagarto" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
</div>
<!-- Final del código--></body></html>
```

Para comprobar si tu respuesta es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01126E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ACCEDER A NODOS HIJOS JAVASCRIPT Y SUS ATRIBUTOS. NODENAME, NODETYPE Y NODEVALUE O TEXTO DEL NODO. EJEMPLOS (CU01126E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº26 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DIFERENCIAS ENTRE NAVEGADORES

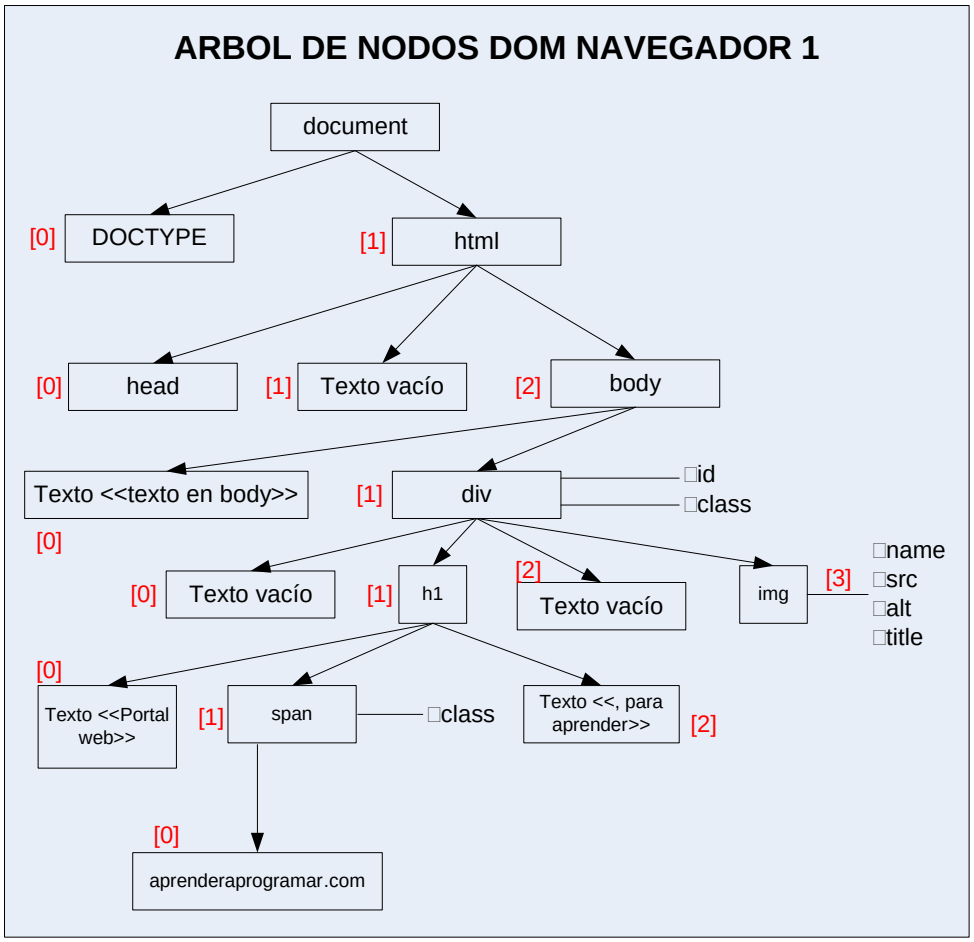
Con cierta frecuencia nos tendremos que enfrentar a que una página web se muestra de cierta forma en un navegador y de otra forma en otro navegador. ¿Por qué estas diferencias entre navegadores? Pueden existir diferentes motivos, pero vamos a ver cómo la representación interna del DOM puede estar relacionado con esto.

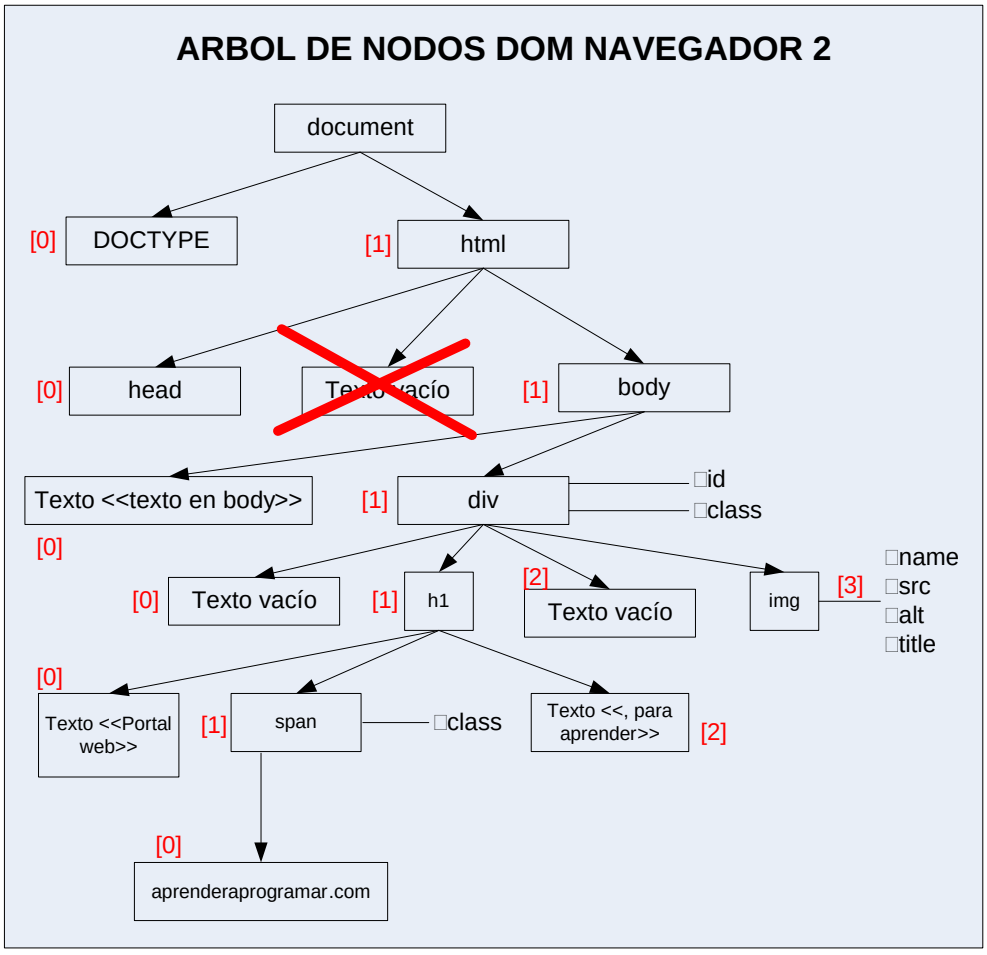
EJEMPLO: ACCESO A NODOS DEL DOM

Vamos a trabajar con un código sencillo y trataremos de acceder a los nodos del DOM a través de código JavaScript. El código que emplearemos es este:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title><meta charset="utf-8">
</head>
<body>Texto en body
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<img name ="lagarto" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
</div>
<!-- Final del código-->
</body>
</html>
```

La primera cuestión que queremos destacar es que distintos navegadores pueden crear distintas representaciones de árbol de nodos. En concreto nosotros hemos probado dos navegadores a los que denominaremos navegador 1 y navegador 2. Hemos numerado los nodos conforme aparecen de izquierda a derecha con 0, 1, 2, 3, etc. y nos hemos encontrado con esto:





Hay distintas cuestiones que merece la pena comentar:

a) Aparecen una serie de nodos que podemos denominar “auxiliares”. En los árboles gráficamente los hemos denominado ***Texto vacío***, y se corresponden con la representación interna que le da el navegador a elementos como saltos de línea. Por ejemplo:

```
<div id="cabecera" class="brillante">

<h1>
```

Puede ser interpretado como que div tiene un hijo que es texto vacío (el salto de línea) y a su derecha aparece el hijo h1. En cambio:

```
<div id="cabecera" class="brillante"><h1>
```

Sería interpretado como que div tiene un hijo que es h1 (no aparece el texto vacío asociado al salto de línea).

Esto no es demasiado importante para nosotros y en general ignoraremos a nodos como estos (texto vacío). Además distintos navegadores pueden usar nodos auxiliares de distinta manera. La idea con que debemos de quedarnos es que la representación de árbol de nodos que usa un navegador **es compleja** y en general no nos interesará conocer los detalles de cómo trabaja el navegador, sino simplemente ser prácticos y poder acceder de forma fácil a los nodos que nos interesen.

b) En el árbol de nodos del navegador 2 hemos señalado que hay un nodo que no es considerado por el navegador 2, pero que en cambio sí era considerado por el navegador 1. ¿Por qué? Porque cada navegador tiene **su propia forma** de construir el árbol de nodos. Ciertamente distintos navegadores pueden construir árboles de nodos parecidos, pero quizás no sean exactamente iguales (y esto en algunas ocasiones nos podrá generar problemas). Si probáramos otro navegador quizás nos encontráramos algún cambio adicional.

c) Dado que estamos representando un código muy muy sencillo, podemos extraer dos conclusiones rápidas: la primera, que si para este código encontramos algunas diferencias en el árbol de nodos, para un código extenso y complejo como suele tener cualquier página web existirán muchas **diferencias internas**. La segunda, que el árbol de nodos de una página web real será muy extenso y muy complejo, tanto que no nos interesará visualizarlo ni conocerlo. Unicamente nos interesa tener los conceptos claros para poder trabajar con el DOM y con JavaScript.

A continuación mostramos el código para acceder a los nodos del navegador 1 y mostrar algunos mensajes por pantalla:

CÓDIGO PARA EL NAVEGADOR 1:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title><meta charset="utf-8">
</head>
<body>Texto en body
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<img name ="lagarto" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
</div>
<!-- Final del código-->
<script type = "text/javascript">
var msg = '';
msg = '¿Quién es el nodo padre de document? '+ document.parentNode + '\n\n';
msg = msg + 'Para el nodo document el nodeType vale: ' + document.nodeType +' , y el nodeName vale '+ document.nodeName + '\n\n';
msg = msg+ 'El valor de nodeValue para el nodo document es: ' + document.nodeValue +'\n\n';
msg = msg + 'Nodo hijo del nodo raíz es la declaración DOCTYPE con nodeName: ' + document.childNodes[0].nodeName +'\n\n';
msg = msg + 'Nodo hijo del nodo raíz es etiqueta html con nodeName: ' + document.childNodes[1].nodeName +' y nodeType: '+
document.childNodes[1].nodeType + '\n\n';
msg = msg + 'Número de hijos de nodo etiqueta html: ' + document.childNodes[1].childElementCount +' ( ' +
document.childNodes[1].children.length +')\n\n';
msg = msg + 'Nodo hijo de etiqueta html es etiqueta head con nodeName: ' + document.childNodes[1].childNodes[0].nodeName +' y
nodeType: '+ document.childNodes[1].childNodes[0].nodeType +'\n\n';
msg = msg + 'Nodo hijo de etiqueta html es tipo texto vacío (salto de línea entre terminación de head y comienzo de body): ' +
document.childNodes[1].childNodes[1].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta html es etiqueta body con nodeName: ' + document.childNodes[1].childNodes[2].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeName: ' +
document.childNodes[1].childNodes[2].childNodes[0].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: ' +
document.childNodes[1].childNodes[2].childNodes[0].nodeValue +'\n\n';
var nodoBody = document.childNodes[1].childNodes[2];
msg = msg + '(Repetimos) Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: ' + nodoBody.childNodes[0].nodeValue
+'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es div con nodeName: ' + nodoBody.childNodes[1].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: ' +
nodoBody.childNodes[1].childNodes[0].nodeName +' y node value: '+ nodoBody.childNodes[1].childNodes[0].nodeValue + '\n\n';
msg = msg + 'Nodo hijo de etiqueta div es etiqueta H1 con nodeName: ' + nodoBody.childNodes[1].childNodes[1].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es texto con nodeName: ' + nodoBody.childNodes[1].childNodes[1].childNodes[0].nodeName
+'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es etiqueta span con nodeName: ' +
nodoBody.childNodes[1].childNodes[1].childNodes[1].nodeName +'\n\n';
var nodoSpan = nodoBody.childNodes[1].childNodes[1].childNodes[1];
msg = msg + 'Nodo hijo de etiqueta span es texto con nodeName: ' + nodoSpan.childNodes[0].nodeName +' y nodeValue: '+
nodoSpan.childNodes[0].nodeValue +'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es texto con nodeName: ' + nodoBody.childNodes[1].childNodes[1].childNodes[2].nodeName +' y
nodeValue: '+ nodoBody.childNodes[1].childNodes[1].childNodes[2].nodeValue +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: ' +
nodoBody.childNodes[1].childNodes[2].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es img con nodeName: ' + nodoBody.childNodes[1].childNodes[3].nodeName +'\n\n';
msg = msg + 'Valor del atributo name de la imagen: ' + nodoBody.childNodes[1].childNodes[3].name +'\n\n';
var nodoImg = nodoBody.childNodes[1].childNodes[3];
msg = msg + 'Valor del atributo src de la imagen: ' + nodoImg.src +', valor de alt: ' + nodoImg.alt + '\n\n';
msg = msg + 'Valor del atributo title de la imagen: '+ nodoImg.title+'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto vacío (salto de línea) con nodeName: ' + nodoBody.childNodes[2].nodeName +' y
nodeType: '+nodoBody.childNodes[2].nodeType+' \n\n';
msg = msg + 'Nodo hijo de etiqueta body es comentario con nodeName: ' + nodoBody.childNodes[3].nodeName +' y nodeType:
'+nodoBody.childNodes[3].nodeType+' y nodeValue: '+nodoBody.childNodes[3].nodeValue+' \n\n';
alert (msg);
</script>
</body>
</html>
```

El resultado esperado en este navegador al tratar de mostrar la página web es el siguiente:

¿Quién es el nodo padre de document? null

Para el nodo document el nodeType vale: 9 , y el nodeName vale #document

El valor de nodeValue para el nodo document es: null

Nodo hijo del nodo raíz es la declaración DOCTYPE con nodeName: html

Nodo hijo del nodo raíz es etiqueta html con nodeName: HTML y nodeType: 1

Número de hijos de nodo etiqueta html: 2 (2)

Nodo hijo de etiqueta html es etiqueta head con nodeName: HEAD y nodeType: 1

Nodo hijo de etiqueta html es tipo texto vacío (salto de línea entre terminación de head y comienzo de body): #text

Nodo hijo de etiqueta html es etiqueta body con nodeName: BODY

Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeName: #text

Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: Texto en body

(Repetimos) Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: Texto en body

Nodo hijo de etiqueta body es div con nodeName: DIV

Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: #text y node value:

Nodo hijo de etiqueta div es etiqueta H1 con nodeName: H1

Nodo hijo de etiqueta h1 es texto con nodeName: #text

Nodo hijo de etiqueta h1 es etiqueta span con nodeName: SPAN

Nodo hijo de etiqueta span es texto con nodeName: #text y nodeValue: aprenderaprogramar.com

Nodo hijo de etiqueta h1 es texto con nodeName: #text y nodeValue: , para aprender

Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: #text

Nodo hijo de etiqueta div es img con nodeName: IMG

Valor del atributo name de la imagen: lagarto

Valor del atributo src de la imagen: http://i.imgur.com/afC0L.jpg, valor de alt: Notepad++

Valor del atributo title de la imagen: Notepad++, un útil editor de texto

Nodo hijo de etiqueta body es texto vacío (salto de línea) con nodeName: #text y nodeType: 3

Nodo hijo de etiqueta body es comentario con nodeName: #comment y nodeType: 8 y nodeValue: Final del código

A continuación mostramos el código para acceder a los nodos del navegador 2 y mostrar algunos mensajes por pantalla:

CÓDIGO PARA EL NAVEGADOR 2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title><meta charset="utf-8">
</head>
<body>Texto en body
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<img name ="lagarto" src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto">
</div>
<!-- Final del código-->
<script type = "text/javascript">
var msg = '';
msg = '¿Quién es el nodo padre de document? '+ document.parentNode + '\n\n';
msg = msg + 'Para el nodo document el nodeType vale: ' + document.nodeType +' , y el nodeName vale '+ document.nodeName +
'\n\n';
msg = msg+ 'El valor de nodeValue para el nodo document es: ' + document.nodeValue +'\n\n';
msg = msg + 'Nodo hijo del nodo raíz es la declaración DOCTYPE con nodeName: ' + document.childNodes[0].nodeName +'\n\n';
msg = msg + 'Nodo hijo del nodo raíz es etiqueta html con nodeName: ' + document.childNodes[1].nodeName +' y nodeType: '+
document.childNodes[1].nodeType + '\n\n';
msg = msg + 'Número de hijos de nodo etiqueta html: ' + document.childNodes[1].childElementCount +' ( ' +
document.childNodes[1].children.length +')\n\n';
msg = msg + 'Nodo hijo de etiqueta html es etiqueta head con nodeName: ' + document.childNodes[1].childNodes[0].nodeName +' y
nodeType: '+ document.childNodes[1].childNodes[0].nodeType +'\n\n';
msg = msg + 'Nodo hijo de etiqueta html es etiqueta body con nodeName: ' + document.childNodes[1].childNodes[1].nodeName
+'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeName: ' +
document.childNodes[1].childNodes[1].childNodes[0].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: ' +
document.childNodes[1].childNodes[1].childNodes[0].nodeValue +'\n\n';
var nodoBody = document.childNodes[1].childNodes[1];
msg = msg + '(Repetimos) Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: ' +
nodoBody.childNodes[0].nodeValue +'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es div con nodeName: ' + nodoBody.childNodes[1].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: ' +
nodoBody.childNodes[1].childNodes[0].nodeName +' y nodeValue ' + nodoBody.childNodes[1].childNodes[0].nodeValue + '\n\n';
msg = msg + 'Nodo hijo de etiqueta div es etiqueta H1 con nodeName: ' + nodoBody.childNodes[1].childNodes[1].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es texto con nodeName: ' + nodoBody.childNodes[1].childNodes[1].childNodes[0].nodeName
+'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es etiqueta span con nodeName: ' +
nodoBody.childNodes[1].childNodes[1].childNodes[1].nodeName +'\n\n';
var nodoSpan = nodoBody.childNodes[1].childNodes[1].childNodes[1];
msg = msg + 'Nodo hijo de etiqueta span es texto con nodeName: ' + nodoSpan.childNodes[0].nodeName +' y nodeValue: '+
nodoSpan.childNodes[0].nodeValue +'\n\n';
msg = msg + 'Nodo hijo de etiqueta h1 es texto con nodeName: ' + nodoBody.childNodes[1].childNodes[1].childNodes[2].nodeName +'
y nodeValue: '+ nodoBody.childNodes[1].childNodes[1].childNodes[2].nodeValue +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: ' +
nodoBody.childNodes[1].childNodes[2].nodeName +'\n\n';
msg = msg + 'Nodo hijo de etiqueta div es img con nodeName: ' + nodoBody.childNodes[1].childNodes[3].nodeName +'\n\n';
msg = msg + 'Valor del atributo name de la imagen: ' + nodoBody.childNodes[1].childNodes[3].name +'\n\n';
var nodoImg = nodoBody.childNodes[1].childNodes[3];
msg = msg + 'Valor del atributo src de la imagen: ' + nodoImg.src +', valor de alt: ' + nodoImg.alt + '\n\n';
msg = msg + 'Valor del atributo title de la imagen: '+ nodoImg.title+'\n\n';
msg = msg + 'Nodo hijo de etiqueta body es texto vacío (salto de línea) con nodeName: ' + nodoBody.childNodes[2].nodeName +' y
nodeType: '+nodoBody.childNodes[2].nodeType+' \n\n';
msg = msg + 'Nodo hijo de etiqueta body es comentario con nodeName: ' + nodoBody.childNodes[3].nodeName +' y nodeType:
'+nodoBody.childNodes[3].nodeType+' y nodeValue: '+nodoBody.childNodes[3].nodeValue+' \n\n';
alert (msg);
</script>
</body>
</html>
```

El resultado esperado en este navegador al tratar de mostrar la página web es el siguiente:

¿Quién es el nodo padre de document? null
Para el nodo document el nodeType vale: 9 , y el nodeName vale #document
El valor de nodeValue para el nodo document es: null
Nodo hijo del nodo raíz es la declaración DOCTYPE con nodeName: **HTML**
Nodo hijo del nodo raíz es etiqueta html con nodeName: HTML y nodeType: 1
Número de hijos de nodo etiqueta html: 2 (2)
Nodo hijo de etiqueta html es etiqueta head con nodeName: HEAD y nodeType: 1
Nodo hijo de etiqueta html es etiqueta body con nodeName: BODY
Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeName: #text
Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: Texto en body
(Repetimos) Nodo hijo de etiqueta body es texto <<Texto en body>> con nodeValue: Texto en body
Nodo hijo de etiqueta body es div con nodeName: DIV
Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: #text y node value:
Nodo hijo de etiqueta div es etiqueta H1 con nodeName: H1
Nodo hijo de etiqueta h1 es texto con nodeName: #text
Nodo hijo de etiqueta h1 es etiqueta span con nodeName: SPAN
Nodo hijo de etiqueta span es texto con nodeName: #text y nodeValue: aprenderaprogramar.com
Nodo hijo de etiqueta h1 es texto con nodeName: #text y nodeValue: , para aprender
Nodo hijo de etiqueta div es texto vacío (salto de línea) con nodeName: #text
Nodo hijo de etiqueta div es img con nodeName: IMG
Valor del atributo name de la imagen: lagarto
Valor del atributo src de la imagen: http://i.imgur.com/afC0L.jpg, valor de alt: Notepad++
Valor del atributo title de la imagen: Notepad++, un útil editor de texto
Nodo hijo de etiqueta body es texto vacío (salto de línea) con nodeName: #text y nodeType: 3
Nodo hijo de etiqueta body es comentario con nodeName: #comment y nodeType: 8 y nodeValue: Final del código

¿Por qué usamos distinto código para distintos navegadores si estamos trabajando con un mismo código HTML?

El motivo es que cada navegador trabaja, de forma interna, de distinta manera. En este caso, al navegador 2 usar distinto número de nodos que el navegador 1, los índices de acceso a nodos no son los mismos para distintos navegadores. En consecuencia, si tratamos de ejecutar el código para el navegador 2 en el navegador 1 posiblemente lo que ocurra es que “no se muestra nada”, debido a que al haber errores en los índices de acceso JavaScript detecta el error y simplemente no se ejecuta (sin mostrar ningún aviso).

¿Significa esto que tendremos que crear distinto código JavaScript para los distintos navegadores?

No, existen métodos más estándares y sencillos para acceder a nodos que veremos más adelante, con los que no cabe esperar diferencias entre distintos navegadores. No obstante, es importante recordar que cada navegador puede ofrecer distintos resultados ante un mismo código (o incluso no ejecutar algo que sí ejecuta otro). Nos tendremos que enfrentar a esta situación en diversas ocasiones y conviene tener conocimiento de ello. Muchas veces te preguntarán (o te preguntarás): ¿Por qué esta página web se muestra de esta manera en un navegador y de otra manera en otra navegador? Los motivos pueden ser varios, pero esto que estamos comentando es uno de ellos.

EJERCICIO

Comprueba cuál de los códigos que hemos empleado se ejecuta en tu navegador. Si no se ejecuta ninguno de ellos, corrígelo para que se ejecute (una estrategia puede ser ir añadiendo línea a línea para saber dónde surgen problemas).

Ambos códigos (el código para el navegador 1 y el código para el navegador 2) son muy similares. ¿Dónde podemos decir que se encuentra la principal diferencia entre uno y otro código? ¿Por qué?

Una vez tengas el código ejecutándose, modifica la forma de acceso a los nodos de modo que en vez de childNodes uses firstChild y nextSibling para acceder a todos los nodos.

Por ejemplo, en vez de: document.childNodes[0].nodeName

Usaríamos: document.firstChild.nodeName

Otro ejemplo. En vez de: document.childNodes[1].childNodes[0].nodeName

Usaríamos: document.firstChild.nextSibling.firstChild.nodeName

Vete probando el código línea a línea. Ten en cuenta que tu navegador podría no reconocer alguna palabra clave. Ten en cuenta que una referencia a un nodo no válida hará que no se ejecute tu código.

Para comprobar si tu respuesta es correcta puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01127E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT: ACCEDER A ELEMENTOS POR ID. GETELEMENTBYID. DOCUMENT.ALL. CAMBIAR IMAGEN IMG SRC (CU01127E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº27 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

ACCEDER A ELEMENTOS JAVASCRIPT

Podemos acceder a los nodos del DOM (Document Object Model) usando JavaScript y los índices de array y relaciones padre – hijo – hermanos (parent, chidNodes, siblings), pero en general nos será más útil poder acceder directamente a aquel nodo que nos resulte de interés. JavaScript proporciona varias funciones para el acceso directo a nodos.

ACCESO DIRECTO A NODOS

Las formas de acceder directamente a nodos con JavaScript podemos resumirlas así:

Acceso directo a nodos	Sintaxis	Ejemplo
Elegido por su atributo id	document.getElementById('valorId');	document.getElementById (menu1);
Elegido por su atributo name	document.getElementsByName('valorName');	document.getElementsBy Name('peticionDatos');
Elegido por el tipo de etiqueta HTML (tag)	document.getElementsByTagName('valorTag');	document.getElementsByTag Name(span);
Elegido por class CSS	document.getElementsByClassName('valorClase');	document.getElementsByClass Name('destacado');
Elegido por selector CSS	document.querySelectorAll('selectorCSS');	document.querySelectorAll ('#menu1 p');
Elegido por selector CSS	document.querySelector('selectorCSS');	document.querySelector ('#menu1 p');

Hace unos años se usaba document.all para realizar selección de nodos en un documento, por ejemplo document.all.tags(“div”) devolvía todos los elementos div en un documento, pero esta sintaxis se considera obsoleta por lo que no vamos a usarla.

Una vez que tenemos acceso directo a los nodos podemos empezar a hacer cosas interesantes como modificar dinámicamente los nodos, como veremos en el ejemplo de código que mostramos más abajo.

GET ELEMENT BY ID

Un atributo que pueden tener las etiquetas HTML es el valor de id, que permite entre otras cosas establecer estilos CSS para dicho elemento y sus descendientes. Por ejemplo establece que el id para este elemento div es “contenedor”.

El atributo id nos va a permitir acceder a un nodo del DOM con dicho valor de atributo. Recordar que en un documento HTML no debería existir más de un elemento con un mismo id. Es decir, los ids deben ser únicos (identificadores únicos de un elemento). En caso de tener dos elementos dentro de nuestro documento HTML que tengan el mismo id, pueden surgir errores (seguramente JavaScript devuelva el primer elemento con el id especificado que encuentre, pero no deberíamos trabajar teniendo ids repetidos porque esto es incorrecto).

El nombre de la función document.getElementById('valorId') nos informa sobre su cometido: nos devuelve un solo nodo de “tipo elemento” del DOM cuyo id coincide con el pasado como argumento.

Una vez tenemos el nodo con el id deseado, podemos modificar dinámicamente sus propiedades. En el siguiente código vemos un ejemplo donde el objetivo es mostrar una imagen con dos botones en su parte inferior: “<<<Atrás” para ir hacia atrás y mostrar una imagen anterior o “Adelante>>>” para ir hacia delante y mostrar una imagen posterior. Si ya estamos en la primera imagen o en la última, aparecerá un mensaje indicando que no es posible realizar ese movimiento. Escribe el código, ejecútalo y razona sobre él. Ten en cuenta que podría surgir algún problema de compatibilidad con tu navegador. Si es así, localiza dónde se encuentra ese problema y corrígelo (si tienes problemas para hacerlo escribe una consulta en los foros aprenderaprogramar.com).

```
<html><head><title>Curso JavaScript aprenderaprogramar.com</title> <meta charset="utf-8">
<style type="text/css">
body {text-align: center; font-family: sans-serif;}
div {margin:20px;}
#contenedor {width:405px;margin:auto;}
#adelante, #atras {padding:15px; width: 130px; float: left;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#adelante:hover, #atras:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
var numeroImagenActual =9;
function moverImagen(movimiento) {
if (numeroImagenActual == 6 && movimiento == 'atras' || numeroImagenActual == 11 && movimiento == 'adelante') {
alert ('No es posible hacer ese movimiento');
}
if (numeroImagenActual == 11 && movimiento == 'atras' || numeroImagenActual == 6 && movimiento =='adelante') {
valorNuevoNumeroImagen = 9;
document.getElementById('imgCarrusel').src =
'http://aprenderaprogramar.com/images/thumbs_portada/thumbs_camisetas/camiseta_9_humor_informatico_foto.jpg';
document.getElementById('imgCarrusel').alt = 'Camiseta 9 aprenderaprogramar.com';
document.getElementById('imgCarrusel').title = 'Diálogo entre informáticos';
}
if (numeroImagenActual == 9 && movimiento == 'atras') {
valorNuevoNumeroImagen = 6;
document.getElementById('imgCarrusel').src =
'http://aprenderaprogramar.com/images/thumbs_portada/thumbs_camisetas/camiseta_6_humor_informatico_foto.jpg';
document.getElementById('imgCarrusel').alt = 'Camiseta 6 aprenderaprogramar.com';
document.getElementById('imgCarrusel').title = 'Desbordado por los números';
document.getElementById('numeracion').nodeValue = '99';
}
if (numeroImagenActual == 9 && movimiento == 'adelante') {
valorNuevoNumeroImagen = 11;
document.getElementById('imgCarrusel').src =
'http://aprenderaprogramar.com/images/thumbs_portada/thumbs_camisetas/camiseta_11_humor_informatico_foto.jpg';
document.getElementById('imgCarrusel').alt = 'Camiseta 11 aprenderaprogramar.com';
document.getElementById('imgCarrusel').title = 'Estudiando programacion';
}
numeroImagenActual = valorNuevoNumeroImagen;
document.getElementById('numeracion').firstChild.nodeValue = 'Camiseta ' + numeroImagenActual;
}
</script>
</head>
<body>
<div >
<p>Pulsa adelante o atrás</p>
<h1 id="numeracion">Camiseta 9</h1>
<img id="imgCarrusel"
src="http://aprenderaprogramar.com/images/thumbs_portada/thumbs_camisetas/camiseta_9_humor_informatico_foto.jpg"
alt="Camiseta 9 aprenderaprogramar.com" title="Diálogo entre informáticos">
<div id="contenedor">
<div id ="atras" onclick="moverImagen('atras')"> <<< Atrás </div>
<div id="adelante" onclick="moverImagen('adelante')" >Adelante >>></div>
</div>
</div></body></html>
```

EJERCICIO

1.Comprueba que el código anterior te permite cambiar la imagen que se muestra haciendo uso de los botones.

2. En el código anterior hay una declaración de variables (var numeroImagenActual =9;) que no está dentro de una función. ¿Por qué crees que se ha hecho esto así? Prueba a colocarla dentro de la función, comprueba qué ocurre y razona el por qué.

3. Modifica el código anterior para que en lugar de definirse src, alt y title para cada imagen dentro de los if, se definan estos valores usando arrays declarados en cabecera de la función. Por ejemplo tendremos que:

valorSrc[0] = 'http://aprenderaprogramar.com/images/thumbs_portada/thumbs_camisetas/camiseta_6_humor_informatico_foto.jpg';

Las asignaciones dentro de los if deberán hacer referencia a los elementos del array y el resultado de ejecución deberá ser el mismo que se obtenía con el código original.

4. Duplica todo el código HTML existente dentro de la etiqueta body, de modo que se muestren dos veces el texto, dos veces la imagen y dos veces los botones. ¿Cuándo pulsas un botón situado debajo a la imagen inferior qué ocurre? ¿Por qué ocurre esto?

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01128E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

GETELEMENTS BYTAGNAME JAVASCRIPT. ACCEDER A ELEMENTOS DE UN FORMULARIO. CAMBIAR ESTILOS CSS CON JAVASCRIPT. (CU01128E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº28 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

GETELEMENTSBYTAGNAME

El acceso a un nodo concreto del DOM usando getElementById es muy frecuente, pero hay otras maneras de acceder a los nodos del DOM. Una de ellas es usando getElementsByTagName('etiquetaBuscada'). Esta función nos devuelve un array conteniendo todos los nodos DOM cuya etiqueta coincide con etiquetaBuscada.

El orden en que aparecerán los elementos en el array (comenzando con índice cero) es el mismo en el que aparezcan en el código de la página web. Ejemplo:

```
var elementosDiv = document.getElementsByTagName('div');
```

Nos devolverá **un array con todos los nodos** de tipo element cuya etiqueta sea div, empezando con índice cero: elementosDiv[0], elementosDiv[1], elementosDiv[2], elementos div[3] … hasta el índice que sea necesario para abarcar tantos elementos div como haya en el código. Podemos obtener los nodos de cualquier tipo de etiqueta: div, span, p, label, input, h1, h2, etc.

Tener en cuenta que escribimos document.getElementsByTagName('div') porque queremos empezar la búsqueda desde el nodo raíz del DOM, es decir, el nodo document (así exploramos todos los nodos). Podríamos empezar la búsqueda por otro nodo si resultara de interés.

CAMBIAR ESTILOS CSS CON JAVASCRIPT

Una posibilidad interesante de JavaScript será poder modificar de forma dinámica el aspecto de la web modificando el CSS asociado. La sintaxis para poder ejecutar modificaciones de estilo será la siguiente:

```
nodoDelDomAlQueAccedemos.style.nombrePropiedad = 'valorPropiedad';
```

Un ejemplo sería: elementosObtenidos[1].style.backgroundColor = '#FF9933';

Donde elementosObtenidos[1] corresponde a un nodo del DOM. Y backgroundColor es la propiedad CSS que deseamos cambiar pero expresada con sintaxis JavaScript. Si te fijas, la sintaxis CSS es background-color mientras que la sintaxis JavaScript es backgroundColor. En este caso son similares, pero no exactamente iguales. En otros casos sí hay coincidencia. Más adelante veremos una tabla de equivalencia entre propiedades CSS

Para ver la aplicación del acceso con getElementsByTagName y el cambio de propiedades CSS con JavaScript escribe el siguiente código y comprueba sus efectos al visualizar la página:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
label {color: maroon; display:block; padding:5px;}
</style>

<script type="text/javascript">
function cambiarAspecto(elemento) {
var elementosObtenidos = document.getElementsByTagName(elemento);
elementosObtenidos[0].style.backgroundColor = '#FF6633';
elementosObtenidos[1].style.backgroundColor = '#FF9933';
elementosObtenidos[2].style.backgroundColor = '#FFCC33';
}
</script>

</head>

<body>
<div id="cabecera">
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>
<!-- Formulario de contacto -->
<div style="width:450px;">
<form name ="formularioContacto" class="formularioTipo1" method="get" action="accion.html"
onclick="cambiarAspecto('label')">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<label>
<input type="submit" value="Enviar">
<input type="reset" value="Cancelar">
</label>
</form>
</div>
</body>
</html>
```

El resultado esperado es que al cargar la página los elementos label del form se muestren con un color de fondo blanco. Al hacer click para escribir en el formulario, se mostrará cada uno de los elementos con un color de fondo ligeramente distinto porque así lo hemos establecido a través del código.

EJERCICIO

Modifica el código anterior para que en vez de pasarse como parámetro label pases como parámetro: div, input, h1 y h2. ¿Se usan los tres índices del array en todos los casos? ¿Qué ocurre si aparecen más índices que elementos realmente existen en el documento html?

Modifica el código anterior para introducir div que simule un botón con el texto “Cambiar a Inglés” y otro div simulando un botón “Cambiar a español”. Al pulsar sobre el botón cambiar a inglés, debes acceder a los nodos del DOM de tipo label y usando relaciones de parentesco entre nodos acceder a aquellos nodos cuyo nodeValue es Nombre:, Apellidos:, y Correo electrónico: y cambiar su nodeValue por Name:, Surname: y e-mail:.

Orientación: tendrás que conocer la estructura del DOM para poder acceder a los nodos con el contenido texto que nos interesa, ya que estos nodos son descendientes de los nodos label.

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01129E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT Y CSS: EQUIVALENCIAS ENTRE PROPIEDADES. LISTA O TABLA PARA CAMBIAR CSS POR JAVASCRIPT. CAMELCASE. (CU01129E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº29 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

SINTAXIS JAVASCRIPT PARA PROPIEDADES CSS

En la anterior entrega del curso hemos visto cómo utilizando JavaScript podemos modificar propiedades CSS asocidas al código HTML de nuestra página web. Dicha sintaxis se basa en el uso de la palabra clave style y la sintaxis JavaScript-DOM para propiedades CSS, cuyas equivalencias mostraremos.

Recordemos la sintaxis para poder ejecutar modificaciones de estilo con JavaScript:

```
nodoDelDomAlQueAccedemos.style.nombrePropiedad = 'valorPropiedad';
```

Un ejemplo sería: elementosObtenidos[1].style.backgroundColor = '#FF9933';

Donde elementosObtenidos[1] corresponde a un nodo del DOM. Y backgroundColor es la propiedad CSS que deseamos cambiar pero expresada con sintaxis JavaScript. La sintaxis CSS es background-color mientras que la sintaxis JavaScript es backgroundColor. En este caso son similares, pero no exactamente iguales. En otros casos sí hay coincidencia. A continuación mostramos una lista de equivalencias con la que nos será útil trabajar, ya que no será posible memorizarlo todo.

Propiedad CSS	Sintaxis JavaScript
background	background
background-attachment	backgroundAttachment
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-color	borderColor
border-left	borderLeft

Propiedad CSS	Sintaxis JavaScript
border-left-color	borderLeftColor
border-left-style	borderLeftStyle
border-left-width	borderLeftWidth
border-right	borderRight
border-right-color	borderRightColor
border-right-style	borderRightStyle
border-right-width	borderRightWidth
border-style	borderStyle
border-top	borderTop
border-top-color	borderTopColor
border-top-style	borderTopStyle
border-top-width	borderTopWidth
border-width	borderWidth
clear	clear
clip	clip
color	color
cursor	cursor
display	display
filter	filter
float	cssFloat
font	font
font-family	fontFamily
font-size	fontSize
font-variant	fontVariant
font-weight	fontWeight
height	height
left	left
letter-spacing	letterSpacing
line-height	lineHeight
list-style	listStyle
list-style-image	listStyleImage
list-style-position	listStylePosition
list-style-type	listStyleType
margin	margin
margin-bottom	marginBottom

Propiedad CSS	Sintaxis JavaScript
margin-left	marginLeft
margin-right	marginRight
margin-top	marginTop
overflow	overflow
padding	padding
padding-bottom	paddingBottom
padding-left	paddingLeft
padding-right	paddingRight
padding-top	paddingTop
page-break-after	pageBreakAfter
page-break-before	pageBreakBefore
position	position
text-align	textAlign
text-decoration	textDecoration
text-indent	textIndent
text-transform	textTransform
top	top
vertical-align	verticalAlign
visibility	visibility
width	width
z-index	zIndex

La regla general que se puede seguir es que para referirnos a una propiedad CSS en JavaScript debemos usar “CamelCase”. CamelCase es un estilo para escribir nombres de elementos de un lenguaje que están formados por varias palabras según el cual en este caso la primera letra la escribiremos en minúscula y luego el comienzo de cada palabra irá en mayúscula (sin separación). Así si quisiéramos poner un nombre de variable para reflejar la “distancia entre puntos” usaríamos distanciaEntrePuntos, o para reflejar el “número de personas que entran por hora al teatro” podríamos usar numeroPersonasEntranPorHoraTeatro.

La mayor parte de las propiedades CSS siguen esta regla. En algunos casos una palabra es palabra clave en JavaScript y para evitar confusiones se antecede de css, por ejemplo la propiedad CSS float tiene como equivalente cssFloat y la propiedad CSS text tiene como equivalente cssText.

Próxima entrega: CU01130E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

GETELEMENTSBYNAME JAVASCRIPT. EVENTO ONSUBMIT AL ENVIAR UN FORMULARIO FORM. CHECKED. EJEMPLO (CU01130E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº30 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

GETELEMENTSBYNAME JAVASCRIPT

El acceso a un nodo concreto del DOM usando getElementById es muy frecuente, pero hay otras maneras de acceder a los nodos del DOM. Una de ellas es usando getElementsByName('valorNameBuscado'). Esta función nos devuelve un array conteniendo todos los nodos DOM cuyo atributo name coincide con valorNameBuscado.

El orden en que aparecerán los elementos en el array (comenzando con índice cero) es el mismo en el que aparezcan en el código de la página web. Ejemplo:

```
var elementosAnimal = document.getElementsByName('animal');
```

Nos devolverá **un array con todos los nodos** de tipo element que tengan como atributo html name "animal", empezando con índice cero: elementosAnimal[0], elementosAnimal[1], elementosAnimal[2], elementosAnimal [3] … hasta el índice que sea necesario para abarcar tantos elementos como haya en el código con el atributo name="animal". Podemos obtener los nodos de cualquier tipo de etiqueta que lleve el atributo name (input, textarea, select, button, img, etc.).

No todos los elementos HTML admiten que se les incluya un atributo name. Por ejemplo … no es correcto porque el atributo name no es aplicable a los elementos div.

Los elementos HTML que admiten el atributo name son: button, form, fieldset, iframe, input, keygen, object, output, select, textarea, map, meta, param.

A diferencia del atributo id que ha de ser único para un elemento dentro de una página web, el atributo name puede aparecer repetido en diversos elementos del código HTML de la página web. Un caso típico sería disponer varios radiobuttons cuyo atributo name es el mismo.

Siempre que sea posible elegir entre usar getElementsById ó getElementsByName recomendamos usar getElementsById pues aporta mayor seguridad.

Tener en cuenta que escribimos document.getElementsByName('animal') porque queremos empezar la búsqueda desde el nodo raíz del DOM, es decir, el nodo document (así exploramos todos los nodos). Podríamos empezar la búsqueda por otro nodo si resultara de interés.

EVENTO ONSUBMMIT

Ya hemos trabajado con algunos eventos como onclick, onload… añadimos ahora a nuestro repertorio un evento denominado onsubmmit, que es el evento que se produce cuando el usuario pulsa el botón de envío de un formulario. Ejemplo:

```
<form name ="formularioContacto" method="get"
action="http://aprenderaprogramar.com" onsubmit="validar()">
```

Según este ejemplo, cuando el usuario pulse el botón enviar del formulario, se ejecutará la función JavaScript validar().

Para ver la aplicación del acceso con getElementsByName y el evento onsubmit con JavaScript escribe el siguiente código y comprueba sus efectos al visualizar la página y pulsar el botón enviar:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo JavaScript - aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
label {color: maroon; display:inline-block; padding:5px;}
</style>
<script type="text/javascript">
function informarItemsElegidos(elemento) {
var elementosObtenidos = document.getElementsByName(elemento);
var msg = 'Animales que ha elegido que le gustan incluye: ';
var elegidos = 0;
if (elementosObtenidos[0].checked == true) {msg = msg + elementosObtenidos[0].value; elegidos=elegidos+1;}
if (elementosObtenidos[1].checked == true) { if (elegidos>=1) {msg = msg + ', ';}
msg = msg + elementosObtenidos[1].value; elegidos=elegidos+1;}
if (elementosObtenidos[2].checked) { if (elegidos>=1) {msg = msg + ', ';}
msg = msg + elementosObtenidos[2].value; elegidos=elegidos+1;}
if (elementosObtenidos[3].checked) { if (elegidos>=1) {msg = msg + ', ';}
msg = msg + elementosObtenidos[3].value; elegidos=elegidos+1;}
if (elegidos == 0 ) {msg = '¡No ha elegido ningún animal!';}
alert (msg);
}
</script>
</head>
<body>
<div id="cabecera">
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>
<!-- Formulario de contacto -->
<div style="width:450px;">
<form name ="formularioContacto" class="formularioTipo1" method="get" action="http://aprenderaprogramar.com"
onsubmit="informarItemsElegidos('animal')">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<p>Elige los animales que te gusten:</p>
<input type="checkbox" name="animal" id="leon" value="leon" /> <label for="leon">León      </label>
<input type="checkbox" name="animal" id="tigre" value="tigre" /> <label for="tigre">Tigre     </label>
```

```
<input type="checkbox" name="animal" id="guepardo" value="guepardo" /> <label for="guepardo">Guepardo
&amp;nbsp;&amp;nbsp;&amp;nbsp; </label>
<input type="checkbox" name="animal" id="jaguar" value="jaguar" /> <label for="jaguar">Jaguar </label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<label>
<input type="submit" value="Enviar" />
<input type="reset" value="Cancelar" />
</label>
</form>
</div>
</body>
</html>
```

El resultado esperado es que al pulsar el botón enviar del formulario se nos muestre un mensaje en función de los animales que hayamos elegido. Si no hemos elegido ningún animal se nos mostrará: ¡No ha elegido ningún animal!. Si hemos elegido por ejemplo el tigre y el jaguar se nos mostrará: Animales que ha elegido que le gustan incluye: tigre, jaguar.

EJERCICIO

Responde a las siguientes preguntas:

a) ¿Escribir if (elementosObtenidos[1].checked == true) genera el mismo resultado que escribir if (elementosObtenidos[1].checked)? ¿Por qué?

b) Modifica el código para obtener el mismo resultado pero sin utilizar if anidados.

c) Utilizando la propiedad length aplicada a la colección de nodos obtenida mediante getElementsByName, modifica el código para que el resultado sea que al enviar el formulario el mensaje que aparezca sea "El número total de animales disponibles era … y usted ha elegido …". Por ejemplo, si tenemos 4 animales (León, Tigre, Guepardo, Jaguar) y hemos elegido Tigre nos debe aparecer el mensaje "El número total de animales disponibles era 4 y usted ha elegido 1".

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01131E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FOR JAVASCRIPT (BUCLES) CONOCER TAMAÑO DE UN ARRAY CON LENGTH. BREAK PARA DETENER EJECUCIÓN. EJEMPLOS Y EJERCICIOS (CU01131E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº31 del Tutorial básico “JavaScript desde cero”.

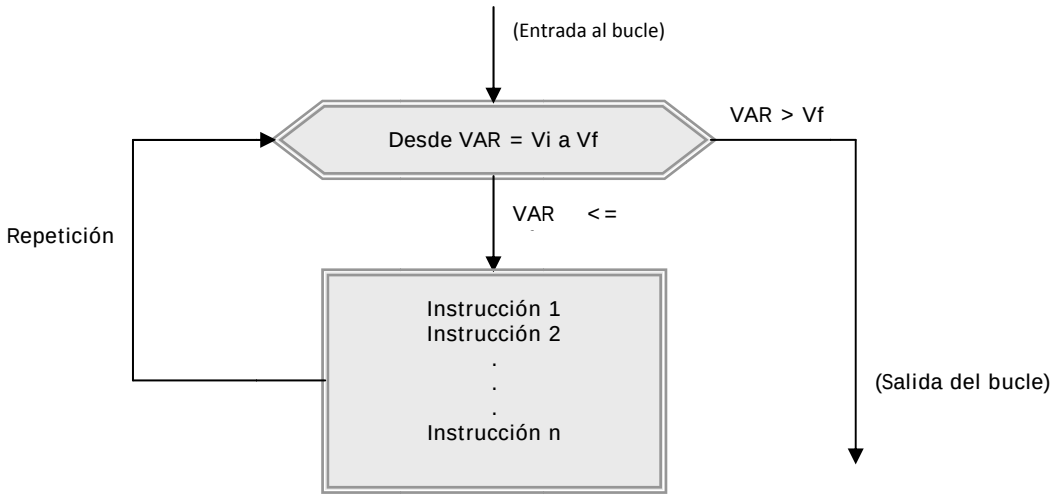
Autor: César Krall

CONCEPTO GENERAL DE BUCLE

Nos referimos a estructuras de repetición o bucles en alusión a instrucciones que permiten la repetición de procesos un número n de veces. Los bucles se pueden materializar con distintas instrucciones como for, while, etc. Un bucle se puede anidar dentro de otro dando lugar a que por cada repetición del proceso exterior se ejecute n veces el proceso interior. Lo veremos con ejemplos.

BUCLE CON INSTRUCCIÓN FOR. OPERADOR ++ Y --. SENTENCIA BREAK.

En JavaScript existen distintas modalidades de for. El caso más habitual, que es el que expondremos a continuación, lo denominaremos for normal o simplemente for. Conceptualmente el esquema más habitual es el siguiente:



La sintaxis habitual es: for (var i = unNumero; i < otroNumero; i++) { instrucciones a ejecutarse }, donde var i supone la declaración de una variable específica y temporal para el bucle. El nombre de la variable puede ser cualquiera, pero suelen usarse letras como i, j, k, etc. unNumero refleja el número en el que se empieza a contar, con bastante frecuencia es 0 ó 1. i < otroNumero ó i <= otroNumero refleja la condición que cuando se verifique supondrá la salida del bucle y el fin de las repeticiones. i++ utiliza el operador ++ cuyo significado es “incrementar la variable i en una unidad”. Este operador se puede usar en otras partes del código, no es exclusivo para los bucles for. Igualmente se dispone del operador “gemelo” – –, que realiza la operación en sentido contrario: reduce el valor de la variable en una unidad. Escribe el siguiente código y comprueba los resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
.boton{padding:15px; width: 200px;  text-align:center; clear:both;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
</style>
<script type="text/javascript">
function ejemploFor() {
var palabra = 'Esternocleidomastoideo';
var subpalabra = new Array();
subpalabra[0]='';
var msg = 'Diez primeras letras: \n\n';
for (var i=1; i<=10; i++){subpalabra[i] = subpalabra[i-1] + palabra.charAt(i-1);}
for (var i=1; i<=10; i++){msg = msg + subpalabra[i] +'\n';}
msg = msg + '\nPalabra al revés: \n\n';
for (var i=palabra.length; i>0; i--){
        msg = msg + palabra.charAt(i-1);
}
alert (msg);
}
</script>
</head>
<body>
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
<h3 class="boton" onclick="ejemploFor()">Pulsa aquí</h3>
</body>
</html>
```

El resultado esperado es que se muestre lo siguiente:

Diez primeras letras:

E
Es
Est
Este
Ester
Estern
Esterno
Esternoc
Esternocl
Esternocle

Palabra al revés:

oediotsamodielconretsE

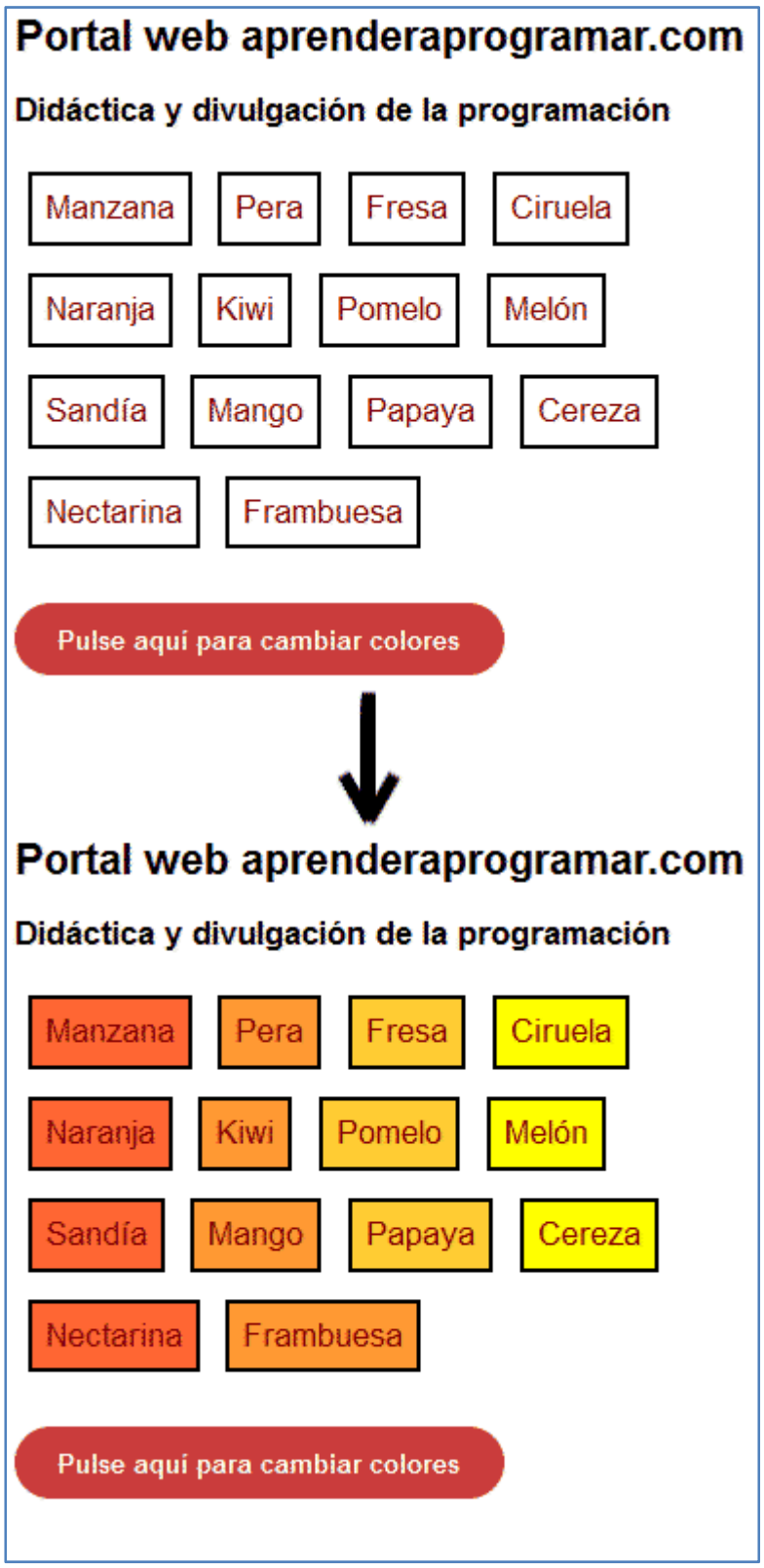
Hemos usado el método charAt(i) que aplicado a una cadena de texto nos devuelve el carácter situado en la posición i (considerando que el primer carácter está en posición 0, el segundo en 1, etc.).

Un bucle for (o de cualquier otro tipo) puede ser interrumpido y finalizado en un momento intermedio de su ejecución mediante una instrucción break;. El uso de esta instrucción dentro de bucles solo tiene sentido cuando va controlada por un condicional que determina que si se cumple una condición, se interrumpe la ejecución del bucle. Por ejemplo: if (i==5) {break;}

Una posibilidad interesante es utilizar métodos para acceder a nodos del DOM y después recorrer dichos nodos con un bucle. Aquí mostramos un ejemplo donde además usamos arrays, el método length para conocer el número de elementos que forman un array y condicionales. Escribe el código y comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo DOM - aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
p {font-size: 24px; color: maroon; float: left; margin:10px; border: solid black; padding:10px;}
.boton{padding:15px; width: 330px;  text-align:center; clear:both;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
.boton:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
var izqda_dcha = true;
function cambiarColores(elemento) {
var color = ['#FF6633', '#FF9933', '#FFCC33', 'yellow'];
var elementosObtenidos = document.getElementsByTagName(elemento);
if (izqda_dcha == true) {
            for (var i=0; i<elementosObtenidos.length; i++) {
            elementosObtenidos[i].style.backgroundColor = color[i%4];
            }
} else {
            for (var j=elementosObtenidos.length-1; j>=0; j--) {
            elementosObtenidos[j].style.backgroundColor = color[(j+3)%4];
            }
}
if (izqda_dcha == false) { izqda_dcha = true; } else { izqda_dcha = false; }
}
</script>
</head>
<body>
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
<div style="width:500px; float:left; margin-bottom:30px;">
<p>Manzana</p><p>Pera</p><p>Fresa</p><p>Ciruela</p>
<p>Naranja</p><p>Kiwi</p><p>Pomelo</p><p>Melón</p>
<p>Sandía</p><p>Mango</p><p>Papaya</p><p>Cereza</p>
<p>Nectarina</p><p>Frambuesa</p></div>
<h3 class="boton" onclick="cambiarColores('p')">Pulse aquí para cambiar colores</h3>
</body>
</html>
```

El resultado esperado es que se muestren inicialmente las cajas con el texto de color blanco y que cuando pulsemos el botón cambien de color. Pulsando nuevamente el botón, los colores volverán a cambiar.



EJERCICIO

1) Crea una función que pida una palabra al usuario y usando un bucle for y el método charAt, muestre cada una de las letras que componen la entrada. Por ejemplo si se introduce “ave” debe mostrar:

Letra 1: a
Letra 2: v
Letra 3: e

2) Crea una función denominada mostrarContParrafos, que utilizando el acceso a los nodos del DOM de tipo párrafo, muestre el texto que contienen. Por ejemplo para el código de ejemplo visto anteriormente el resultado debería ser: Párrafo 1 contiene: manzana; Párrafo 2 contiene: pera. Párrafo 3 contiene: fresa. Párrafo 4 contiene … (etc., hasta que no haya más párrafos).

Puedes comprobar si tus respuestas son correctas consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU01132E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

WHILE JAVASCRIPT. DO WHILE. CONTINUE PARA SALTAR BUCLE. BUCLES CON LABEL (ETIQUETA O NOMBRE). EJEMPLOS (CU01132E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

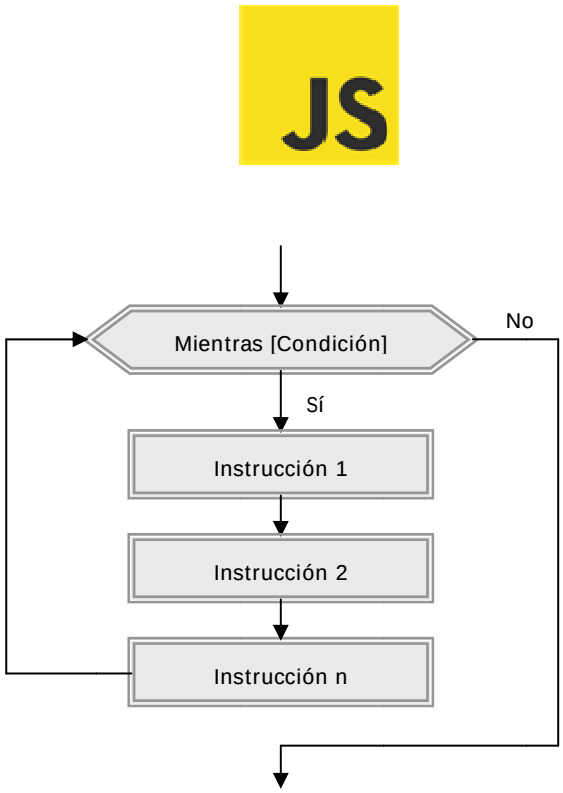
Fecha revisión: 2029

Resumen: Entrega nº32 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

BUCLE CON INSTRUCCIÓN WHILE EN JAVASCRIPT. EJEMPLO USO DE BREAK.

El bucle while presenta ciertas similitudes y ciertas diferencias con el bucle for. La repetición en este caso se produce no un número predeterminado de veces, sino mientras se cumpla una condición. Conceptualmente el esquema más habitual es el siguiente:



La sintaxis en general es: *while (condición) { instrucciones a ejecutarse }* donde *condición* es una expresión que da un resultado true o false en base al cual el bucle se ejecuta o no. Escribe y prueba el siguiente código, donde además vemos un ejemplo de uso de la instrucción *break;*.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
.boton{padding:15px; width: 200px;  text-align:center; clear:both;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
</style>
<script type="text/javascript">
function ejemploWhile() {
var i = 0; var msg = '';
while (true) {
i++;
msg = msg + '\t'+ i + '\n';
if (i==9) {break;}
}
```

```
alert ('Bucle con while: \n'+ msg);
}
</script>
</head>
<body>
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
<h3 class="boton" onclick="ejemploWhile()">Pulsa aquí</h3>
</body>
</html>
```

El resultado esperado es que se muestre lo siguiente:

Bucle con while:
1
2
3
4
5
6
7
8
9

En este código hemos hecho algo un poco extraño. Como condición a evaluar hemos puesto “true”. Esto significa que la condición es siempre verdadera, lo que en teoría daría lugar a un bucle infinito y a un bloqueo del ordenador. Sin embargo, utilizamos un contador auxiliar que inicializamos en cero y en cada repetición del bucle aumentamos en una unidad. A su vez, introducimos una condición dentro del bucle según la cual cuando el contador alcanza el valor 9 se ejecuta la instrucción break.

Este ejemplo debe valernos solo como tal: en general la condición de entrada al bucle será una expresión a evaluar como (i < 10 ó a >= 20 ó reductor < compresor) y no un valor *true*. Y en general la salida a un bucle se realizará de forma natural mediante la evaluación de la condición y no mediante una instrucción *break;*.

BUCLE CON INSTRUCCIÓN DO … WHILE. EJEMPLO DE USO.

El bucle do … while es muy similar al bucle while. La diferencia radica en cuándo se evalúa la condición de salida del ciclo. En el bucle while esta evaluación se realiza antes de entrar al ciclo, lo que significa que el bucle puede no llegar ejecutarse. En cambio, en un bucle do … while, la evaluación se hace después de la primera ejecución del ciclo, lo que significa que el bucle obligatoriamente se ejecuta al menos en una ocasión. Después de la condición del while debemos escribir punto y coma. Por ejemplo while (contador < 10);

A modo de ejercicio, escribe este código y comprueba los resultados que se obtienen con él:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">body {background-color:white; font-family: sans-serif;}
.boton{padding:15px; width: 200px;  text-align:center; clear:both;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
</style>
<script type="text/javascript">
function ejemploDoWhile() {
var contador = 0;var msg = '';
do { msg = msg + '\t Contando... '+ (contador+1) + '\n';
contador +=1;
} while (contador < 10);
alert ('Bucle con do ... while: \n'+ msg);
}
</script>
</head>
<body>
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
<h3 class="boton" onclick="ejemploDoWhile()">Pulsa aquí</h3>
</body></html>
```

El resultado esperado es que se muestre lo siguiente:

Bucle con do ... while:
Contando... 1
Contando... 2
Contando... 3
Contando... 4
Contando... 5
Contando... 6
Contando... 7
Contando... 8
Contando... 9
Contando... 10

BUCLES ETIQUETADOS O CON NOMBRE. SENTENCIA CONTINUE

En JavaScript las sentencias pueden llevar un nombre o etiqueta que los identifique y que sirvan para hacer referencia a ellos en algún momento. Por ejemplo los bucles podemos etiquetarlos como vemos en el siguiente ejemplo:

```
msg = msg + 'Bucles etiquetados: \n'
bucleExterior: for(var i=0; i<2; i++) {
        bucleInterior: for (var j=0; j<5; j++){
        msg = msg + 'i vale ' + i + ', j vale ' + j + ', i*j =  ' + (i*j) +'\n';
        }
}
```

La sentencia continue da lugar a que se salte a la siguiente iteración del bucle sin ejecutar las instrucciones existentes después de la sentencia continue. Si escribimos continue; la sentencia afecta al bucle más próximo a donde se encuentre la sentencia. Si escribimos continue nombreDeLaEtiquetaDelBucle; la sentencia afecta a aquel bucle que hayamos referenciado.

El siguiente código nos muestra ejemplos de uso de continue en bucles etiquetados y sin etiquetar:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
.boton{padding:15px; width: 200px;  text-align:center; clear:both;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
</style>
<script type="text/javascript">
function ejemploBuclesConContinue() {
var contador = 0;
var msg = 'Bucle do con ejemplo de continue: \n';
do {
if (contador>5 && contador < 12) {contador +=1; continue;}
msg = msg + '\t Contando... '+ (contador+1) + '\n';
//Otras instrucciones aquí...
contador +=1;
} while (contador < 20);
msg = msg + 'Bucles etiquetados y uso de continue referenciado: \n'
bucleExterior: for(var i=0; i<2; i++) {
            bucleInterior: for (var j=0; j<5; j++){
            comprobarSijvale2: if (j==2) {continue bucleExterior;}
            msg = msg + 'i vale ' + i + ', j vale ' + j + ', i*j =  ' + (i*j) +'\n';
            }
}
msg = msg + 'Bucles sin etiquetar y continue sin referenciar: \n'
bucleExterior: for(var i=0; i<2; i++) {
            bucleInterior: for (var j=0; j<5; j++){
            if (j==2) {continue;}
            msg = msg + 'i vale ' + i + ', j vale ' + j + ', i*j =  ' + (i*j) +'\n';
            }
}
alert (msg);
}
</script>
</head>
<body>
<h1>Portal web aprenderaprogramar.com</h1>
<h2>Didáctica y divulgación de la programación</h2>
<h3 class="boton" onclick="ejemploBuclesConContinue()">Pulsa aquí</h3>
</body>
</html>
```

El resultado esperado de ejecución es el siguiente. Compruébalo y razona el por qué de este resultado:

Bucle do con ejemplo de continue:
Contando... 1
Contando... 2
Contando... 3
Contando... 4
Contando... 5
Contando... 6
Contando... 13
Contando... 14
Contando... 15
Contando... 16
Contando... 17
Contando... 18
Contando... 19
Contando... 20
Bucles etiquetados y uso de continue referenciado:
i vale 0, j vale 0, i*j = 0
i vale 0, j vale 1, i*j = 0
i vale 1, j vale 0, i*j = 0
i vale 1, j vale 1, i*j = 1
Bucles sin etiquetar y continue sin referenciar:
i vale 0, j vale 0, i*j = 0
i vale 0, j vale 1, i*j = 0
i vale 0, j vale 3, i*j = 0
i vale 0, j vale 4, i*j = 0
i vale 1, j vale 0, i*j = 0
i vale 1, j vale 1, i*j = 1
i vale 1, j vale 3, i*j = 3
i vale 1, j vale 4, i*j = 4

EJERCICIO

1) Crea una función que pida una palabra al usuario y usando un bucle while y el método charAt, muestre cada una de las letras que componen la entrada. Por ejemplo si se introduce “ave” debe mostrar:

Letra 1: a Letra 2: v Letra 3: e

Puedes comprobar si tus respuestas son correctas consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU01133E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FUNCIÓN EVAL JAVASCRIPT. GETELEMENTSBY CLASSNAME . EJEMPLO CÓDIGO CALCULADORA SIMPLE (CU01133E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº33 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

FUNCIÓN EVAL. GETELEMENTSBYCLASSNAME.

La función eval tiene diferentes aplicaciones en JavaScript. Veremos la aplicación más básica y comentaremos algunos aspectos de esta función cuyo uso es poco recomendado. Veremos también una posibilidad adicional para seleccionar nodos según el valor de su atributo class. La sintaxis a emplear es getElementsByClassName. Se utiliza ClassName en lugar de Class porque class es una palabra reservada JavaScript.

FUNCIÓN EVAL.

La función eval tiene diferentes aplicaciones en JavaScript. Algunos expertos en programación desaconsejan directamente su uso indicando que debe evitarse su utilización (es famosa la frase de Douglas Crockford “eval is evil”, eval es el demonio). Otros expertos indican que es una herramienta que bien usada puede ser muy útil. Nosotros no vamos a decantarnos por una u otra postura. Simplemente expondremos algunas cuestiones básicas sobre esta función.

La sintaxis de la función es la siguiente:

```
eval ('cadenaDeTextoPasadaALaFuncion');
```

El valor de cadenaDeTextoPasadaALaFuncion puede ser:

a) Una expresión JavaScript. Por ejemplo '3+2'. En este caso eval('3+2') evalúa la expresión y devuelve el valor 5, es decir, el resultado de evaluar la expresión. Podríamos escribir valorOperacion = eval('3+2'); y valorOperacion tomaría el valor 5.

b) Una sentencia o fragmento de código JavaScript. Por ejemplo:

eval('moverImagen(\'adelante\')'); dará lugar a que se ejecute la llamada a la función moverImagen('adelante') y que se devuelva, si existe, un valor de retorno. Fíjate que para poder incluir la comilla simple dentro de la cadena de texto hemos usado el escape del carácter con el simbolo \. También sería válido usar esta sintaxis:

eval("moverImagen('adelante')");

Como decimos eval ('cadenaDeTextoPasadaALaFuncion'); tiene dos implicaciones: una evaluación o ejecución, y la obtención de un valor de retorno (si existe). De este modo la función eval puede usarse como argumento para otras funciones, por ejemplo:

alert(Hemos obtenido un resultado: ' + eval("moverImagen('adelante')"));

El resultado será el contenido de la sentencia return de la función si existía. En caso de no existir un valor de retorno este será un valor no válido como undefined.

Veamos un ejemplo un poco más elaborado donde se utiliza eval para hacer una llamada a una función:

```
function componer(sentido) {
var accion = 'mover';
var sobreElemento = 'Imagen';
var haciaDonde = sentido;
var llamada = accion + sobreElemento+'(\''+haciaDonde+'\')';
alert ('La unión del texto genera ' + llamada);
eval(llamada);
}
```

En este ejemplo hemos compuesto un texto a partir de las partes “mover”, “Imagen”, y un parámetro “sentido”. Si sentido es igual a 'adelante' por ejemplo, el resultado es que se ejecutará eval(moverImagen('adelante')); y por tanto se ejecuta la función moverImagen('adelante')

Esto nos permite invocar funciones compuestas a partir de fragmentos, donde cada fragmento puede estar determinado por condicionales o por otras funciones.

Se señalan distintos inconvenientes para la función eval:

a) Genera código difícil de leer, donde se mezcla texto con variables.

b) Resulta de ejecución pesada y lenta para el intérprete, pudiendo dar lugar a que nuestra aplicación web responda mal o lentamente.

c) Puede poner en riesgo la seguridad de las aplicaciones, ya que si por ejemplo introducimos en eval una cadena proporcionada por un usuario malicioso, éste puede intentar que se ejecute código que comprometa la seguridad.

Iremos conociendo alternativas al uso de eval, pero de momento señalaremos que el siguiente código genera el mismo resultado que el anterior evitando el uso de eval:

```
function componer(sentido) {
var accion = 'mover';
var sobreElemento = 'Imagen';
var haciaDonde = sentido;
var llamada = accion + sobreElemento+'(\''+haciaDonde+'\')';
alert ('La unión del texto genera ' + llamada);
var tmpFunc = new Function(llamada);
tmpFunc();
}
```

GETELEMENTSBYCLASSNAME

El acceso a un nodo concreto del DOM usando getElementById es muy frecuente, pero hay otras maneras de acceder a los nodos del DOM. Ya conocemos algunas alternativas. Otra de ellas es usar getElementsByClassName('valorClassBuscado'). Esta función nos devuelve un array conteniendo todos los nodos DOM cuyo atributo class coincide con valorClassBuscado.

El orden en que aparecerán los elementos en el array (comenzando con índice cero) es el mismo en el que aparezcan en el código de la página web. Ejemplo:

```
var elementosDestacado = document.getElementsByClassName('destacado');
```

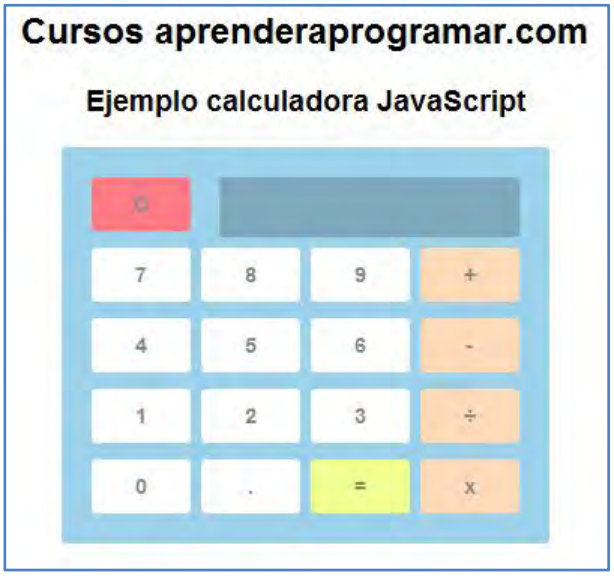
Nos devolverá **un array con todos los nodos** de tipo element que tengan como atributo html class "destacado", empezando con índice cero: elementosDestacado[0], elementosDestacado[1], elementosDestacado[2], elementosDestacado[3] … hasta el índice que sea necesario para abarcar tantos elementos como haya en el código con el atributo class="destacado". Podemos obtener los nodos de cualquier tipo de etiqueta que lleve el atributo class (div, span, input, etc.).

A diferencia del atributo id que ha de ser único para un elemento dentro de una página web, el atributo class puede aparecer repetido en diversos elementos del código HTML de la página web. Normalmente se usa para identificar partes de la página web a la que deben aplicársele las mismas reglas CSS.

Tener en cuenta que escribimos document.getElementsByClassName('destacado') porque queremos empezar la búsqueda desde el nodo raíz del DOM, es decir, el nodo document (así exploramos todos los nodos). Podríamos empezar la búsqueda por otro nodo si resultara de interés.

CREAR UNA CALCULADORA SIMPLE CON JAVASCRIPT

A continuación vamos a estudiar un ejemplo de código donde creamos una calculadora simple usando JavaScript con la sintaxis getElementsByClassName y la función eval para realizar los cálculos.



Hemos incluido código CSS para crear una visualización agradable. Así, CSS se encarga principalmente del aspecto y JavaScript se encarga principalmente de la parte dinámica (respuesta ante las acciones del usuario). Escribe este código y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center;}
#calculadora { font: bold 14px Arial,sans-serif; background-color: #9DD2EA;  border-radius: 3px; height: auto; margin: 0 auto;
   padding: 20px 20px 9px; width: 285px;}
.parteSuperior .pantalla { background-color: rgba(0, 0, 0, 0.2); border-radius: 3px; color: #FFFFFF; float: right; font-size: 17px;
height: 40px; letter-spacing: 1px; line-height: 40px; padding: 0 10px; text-align: right; width: 180px;}
.teclas, .parteSuperior {overflow: hidden; }
.teclas span, .parteSuperior span.limpiar {     background-color: #FFFFFF; border-radius: 3px; color: #888888; cursor: pointer;
   float: left; height: 36px; line-height: 36px; margin: 0 7px 11px 0; text-align: center; transition: all 0.4s ease 0s; width: 66px;}
.parteSuperior span.limpiar { background-color:#FF7C87;}
.teclas span.operador { background-color: #FFDAB9; margin-right: 0; }
.teclas span.igual { background-color: #F1FF92; color: #888E5F;}
.parteSuperior span.limpiar, { background-color: #FF9FA8; color: #FFFFFF; }
.teclas span:hover, .teclas span.igual:hover, .parteSuperior span.limpiar:hover {
   background-color: #9C89F6; color: #FFFFFF;
}
</style>
<script type="text/javascript">
function pulsada (tecla) {
var listaNodosPantalla = document.getElementsByClassName('pantalla');
var nodoTextoPantalla = listaNodosPantalla[0].firstChild;
switch(tecla) {
            case 'C':
            nodoTextoPantalla.nodeValue = ' ';
            break;
            case '=':
            var resultado = eval(nodoTextoPantalla.nodeValue);
            nodoTextoPantalla.nodeValue = resultado;
            break;
            default:
            nodoTextoPantalla.nodeValue = nodoTextoPantalla.nodeValue + tecla;
            break;
}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo calculadora JavaScript</h3></div>
<!-- Calculadora -->
<div id="calculadora">
            <!-- pantalla y tecla limpiar -->
            <div class="parteSuperior">
                        <span class="limpiar" onclick="pulsada('C')">C</span>
                        <div class="pantalla"> </div> <!--Ojo tenemos un espacio para que exista nodeValue-->
            </div>
            <div class="teclas">
                        <!-- operadores y otras teclas -->
                        <span onclick="pulsada('7')">7</span>
                        <span onclick="pulsada('8')">8</span>
```

```
                    <span onclick="pulsada('9')">9</span>
                    <span class="operador" onclick="pulsada('+')">+</span>
                    <span onclick="pulsada('4')">4</span>
                    <span onclick="pulsada('5')">5</span>
                    <span onclick="pulsada('6')">6</span>
                    <span class="operador" onclick="pulsada('-')">-</span>
                    <span onclick="pulsada('1')">1</span>
                    <span onclick="pulsada('2')">2</span>
                    <span onclick="pulsada('3')">3</span>
                    <span class="operador" onclick="pulsada('/')">÷</span>
                    <span onclick="pulsada('0')">0</span>
                    <span onclick="pulsada('.')">.</span>
                    <span class="igual" onclick="pulsada('=')">=</span>
                    <span class="operador" onclick="pulsada('*')">x</span>
            </div>
</div>
</body>
</html>
```

De este código podemos comentar lo siguiente. Debes ser capaz de comprender cómo el código CSS da lugar a la visualización de la calculadora. Debes ser capaz de comprender por qué la invocación listaNodosPantalla[0].firstChild nos permite acceder al nodo de texto con el contenido de la pantalla de la calculadora. Debes ser capaz de comprender cómo manipulamos el contenido de la pantalla accediendo al nodo texto y cómo modificamos su contenido y mostramos resultados usando la función eval. Si tienes dudas consulta en los foros aprenderaprogramar.com

EJERCICIOS

1) Modifica el código del ejemplo de la calculadora javascript para que en lugar del condicional switch uses el condicional if (con if else ó if else if cuando sea necesario).

2) Utilizando el ejemplo de código que usa la función eval en la función <<function componer(sentido)>> que hemos visto dentro del código de los ejemplos, crea una página web donde existan dos botones que pongan “Adelante” y “Atrás”, de modo que cuando se pulsen den lugar a que se llame la función componer(sentido). En esta función, a través de la función eval se ejecutará moverImagen('adelante') ó moverImagen('atras') según el parámetro recibido. La función moverImagen(sentido) puede contener un simple mensaje indicando el valor del parámetro <<sentido>> recibido.

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01134E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

QUERYSELECTORALL JAVASCRIPT Y QUERYSELECTOR. ACCEDER A ELEMENTOS POR SELECTORES CSS. EJEMPLOS (CU01134E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº34 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

QUERYSELECTORALL Y QUERYSELECTOR JAVASCRIPT

Hemos visto que JavaScript provee métodos para acceder y manipular nodos del DOM, en concreto getElementById, getElementsByTagName y getElementsByName, que nos permiten acceder a un elemento por su valor de atributo id, o a la colección de elementos cuya etiqueta es de un determinado tipo o tiene como atributo name un valor concreto. Veremos ahora métodos que nos permiten acceder a cualquier nodo identificado por un selector CSS específico.

QUERYSELECTORALL Y QUERYSELECTOR

Si recordamos la sintaxis CSS, a través de selectores podíamos definir de forma muy específica cómo se debían a aplicar estilos dentro de nuestra página web. Por ejemplo a través de la regla:

#menu1 div.destacado a {background-color: yellow;}

Se consigue que los elementos a (links) dentro de elementos div cuyo atributo class es “destacado” y están dentro de un elemento con id menu1 se muestren con color de fondo amarillo.

El selector #menu1 div.destacado a es un selector complejo que no resultaría alcanzable con las técnicas de acceso a nodos que hemos visto hasta el momento. Por ello surgen los métodos querySelector y querySelectorAll, que permiten acceder a nodos a través de un selector CSS.

La sintaxis a emplear es la siguiente:

```
document.querySelectorAll("aquí el selector CSS");
```

Por ejemplo document.querySelectorAll("#calculadora .teclas span.operador");

La sintaxis a incluir dentro de las comillas para especificar el selector es exactamente la misma sintaxis que empleamos con CSS. Dentro de la especificación se pueden indicar varios selectores separados por comas, al igual que hacemos con CSS para aplicar una misma regla a distintos elementos.

Esta instrucción nos devuelve un array de nodos del DOM que cumplen con la especificación del selector CSS. Los índices del array serán 0, 1, 2, … hasta un número indeterminado (que dependerá del número de nodos que cumplan con el selector). Los nodos estarán ordenados según su orden de aparición en el árbol del DOM que define el documento HTML.

Frente al método general querySelectorAll existe otro método específico, querySelector, que devuelve un único nodo que cumple la especificación del selector. Este único nodo devuelto será el primero encontrado que cumpla con la especificación dentro del árbol del DOM.

La sintaxis es:

```
document.querySelector ("aquí el selector CSS ");
```

Típicamente el array de nodos (o el nodo individual) obtenido con estos métodos los almacenaremos dentro de una variable JavaScript para después realizar algún tipo de tratamiento. Por ejemplo:

```
var listaNodosOperadores = document.querySelectorAll("#calculadora .teclas span.operador");

var nodoIgual = document.querySelector (".igual");
```

Una vez tenemos una colección de nodos en una variable, podemos recorrerla usando un bucle for. Por ejemplo:

```
for (var i=0; i<listaNodosOperadores.length; i++) {

    // Ejecutar acciones sobre el nodo listaNodosOperadores[i].

 }
```

Los métodos querySelector se pueden aplicar sobre document como hemos visto (para seleccionar todos los nodos hijos del nodo raíz que cumplen un criterio), pero también sobre un nodo especificado obtenido de otra manera. Por ejemplo:

```
var listaNodos = document.getElementById("calculadora").querySelectorAll(".teclas span.numero");
```

Aquí primero se obtiene el nodo cuyo id es “calculadora” y después se obtienen todos los nodos hijos de calculadora cuyo que son elementos span con class=”numero” que se encuentran dentro de un elemento con un valor class=”teclas”

En este caso lo indicado anteriormente se puede escribir también como:

```
var listaNodosOperadores = document.querySelectorAll("#calculadora .teclas span.operador");
```

Esta última forma de escritura resulta quizás más clara, y por eso la consideraremos preferible cuando podamos elegir entre usar una forma u otra.

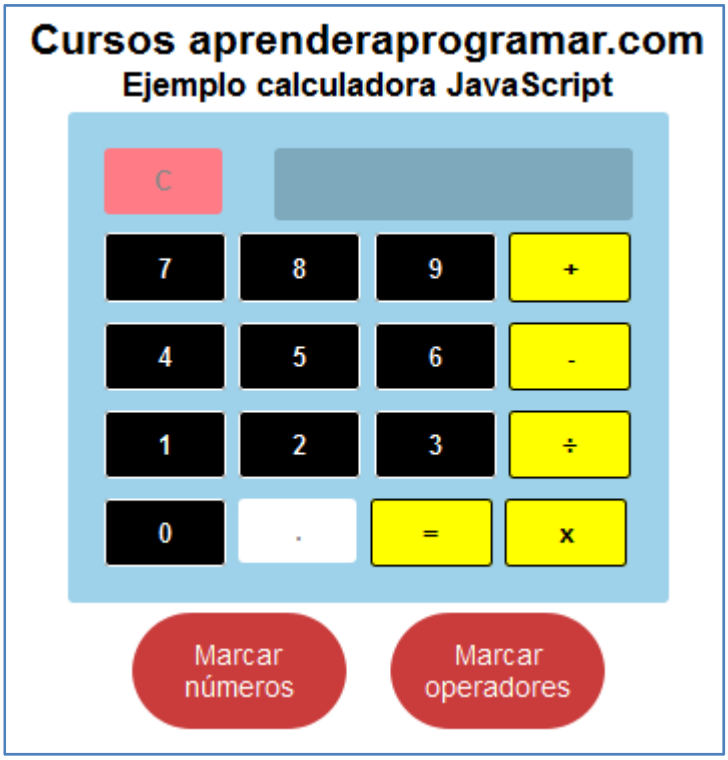
Escribe y prueba el siguiente código, donde vemos un ejemplo de uso de querySelector y de querySelectorAll. Ten en cuenta que en algunos navegadores, en especial los más antiguos, quizás no se obtengan los resultados deseados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center;}
#calculadora {    font: bold 14px Arial,sans-serif; background-color: #9DD2EA;
   border-radius: 3px; height: auto; margin: 0 auto;  padding: 20px 20px 9px; width: 295px;
}
.parteSuperior .pantalla {
   background-color: rgba(0, 0, 0, 0.2); border-radius: 3px; color: #FFFFFF; float: right; font-size: 17px;
   height: 40px; letter-spacing: 1px; line-height: 40px; padding: 0 10px; text-align: right; width: 180px;
}
.teclas, .parteSuperior {overflow: hidden; }
.teclas span, .parteSuperior span.limpiar {     background-color: #FFFFFF; border-radius: 3px; color: #888888;
 cursor: pointer; float: left; height: 36px; line-height: 36px; margin: 0 7px 11px 0;
   text-align: center; transition: all 0.4s ease 0s; width: 66px;
}
.parteSuperior span.limpiar { background-color:#FF7C87;}
.teclas span.operador { background-color: #FFDAB9; margin-right: 0; }
.teclas span.igual { background-color: #F1FF92; color: #888E5F;}
.parteSuperior span.limpiar, { background-color: #FF9FA8; color: #FFFFFF; }
.teclas span:hover, .teclas span.igual:hover, .parteSuperior span.limpiar:hover { background-color: #9C89F6;
color: #FFFFFF; }

#contenedor {width:285px;margin: 40px auto; overflow:hidden;}
#marcaNumeros, #marcaOperadores {padding:15px; width: 90px; display: inline-block;
margin: 10px; cursor: pointer; color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#marcaNumeros:hover, #marcaOperadores:hover {background: rgb(66, 184, 221);}
</style>

<script type="text/javascript">

function marcarOperadores() {
var listaNodosOperadores = document.querySelectorAll("#calculadora .teclas span.operador");
var nodoIgual = document.querySelector (".igual");
     for (var i=0; i<listaNodosOperadores.length; i++) {
          listaNodosOperadores[i].style.backgroundColor = "yellow";
                                        listaNodosOperadores[i].style.color = "black";
                                        listaNodosOperadores[i].style.border = "solid 1px";
          }
           nodoIgual.style.backgroundColor = "yellow";
           nodoIgual.style.color = "black";
           nodoIgual.style.border = "solid 1px";
}

function marcarNumeros() {
var listaNodosNumeros = document.querySelectorAll("#calculadora .teclas span.numero");
     for (var i=0; i<listaNodosNumeros.length; i++) {
          listaNodosNumeros[i].style.backgroundColor = "black";
                                        listaNodosNumeros[i].style.color = "white";
                                        listaNodosNumeros[i].style.border = "solid 1px";
          }
}
</script>
</head>
```

```
<body>
<div id="cabecera">
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplo calculadora JavaScript</h3>
</div>
<!-- Calculadora -->
<div id="calculadora">
	<!-- pantalla y tecla limpiar -->
	<div class="parteSuperior">
		<span class="limpiar" onclick="pulsada('C')">C</span>
		<div class="pantalla"> </div> <!--Ojo tenemos un espacio para que exista nodeValue-->
	</div>

	<div class="teclas">
		<!-- operadores y otras teclas -->
		<span class = "numero" onclick="pulsada('7')">7</span>
		<span class = "numero" onclick="pulsada('8')">8</span>
		<span class = "numero" onclick="pulsada('9')">9</span>
		<span class="operador" onclick="pulsada('+')">+</span>
		<span class = "numero" onclick="pulsada('4')">4</span>
		<span class = "numero" onclick="pulsada('5')">5</span>
		<span class = "numero" onclick="pulsada('6')">6</span>
		<span class="operador" onclick="pulsada('-')">-</span>
		<span class = "numero" onclick="pulsada('1')">1</span>
		<span class = "numero" onclick="pulsada('2')">2</span>
		<span class = "numero" onclick="pulsada('3')">3</span>
		<span class="operador" onclick="pulsada('/')">÷</span>
		<span class = "numero" onclick="pulsada('0')">0</span>
		<span class ="punto" onclick="pulsada('.')">.</span>
		<span class="igual" onclick="pulsada('=')">=</span>
		<span class="operador" onclick="pulsada('*')">x</span>
	</div>
</div>
<div id="contenedor">
<div id ="marcaNumeros" onclick="marcarNumeros()"> Marcar números </div>
<div id="marcaOperadores" onclick="marcarOperadores()" >Marcar operadores</div>
</div>
</body>
</html>
```

El resultado esperado es que se muestre una calculadora y en su parte inferior dos botones: “Marcar números” y “Marcar operadores”. Al pulsar sobre “Marcar números” las teclas correspondientes a los números de la calculadora deberán verse de color negro con los números de color blanco. Al pulsar sobre “Marcar operadores” las teclas correspondientes a los operadores (+, -, =, etc.) deberán verse de color amarillo con los operadores de color negro.

Fíjate en el código: la selección de las teclas afectadas y el cambio de estilos se hace gracias al uso de los operadores querySelectorAll y querySelector de JavaScript.



EJERCICIO

1) Modifica el código del ejemplo de la calculadora JavaScript para que además de tener dos botones que permitan marcar números y operadores con distintos colores tenga:

a) Un botón que permita marcar la tecla C con color rojo fuerte y símbolo C en blanco.

b) Un botón que permita marcar la tecla con el punto con color violeta fuerte y símbolo . en blanco.

c) Un botón que permita marcar la pantalla visor de la calculadora en azul oscuro.

d) Un botón que permita limpiar todo lo que se haya marcado y volver a la situación original (es decir, que la calculadora se vea tal y como se veía inicialmente).

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01135E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FOR IN JAVASCRIPT: RECORRER ARRAYS O PROPIEDADES DE OBJETOS. DIFERENCIA ENTRE FOR NORMAL Y FOR IN. EJEMPLOS (CU01135E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº35 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

RECORRER COLECCIONES CON FOR IN

El recorrido de arrays es posible usando un bucle for normal o un bucle while, donde existe un control explícito de los índices que se van recorriendo (0, 1, 2, 3…). JavaScript permite recorrer arrays y colecciones de objetos usando un tipo especial de bucle: el bucle for – in (que a su vez tiene ciertas similitudes con el for each).

La sintaxis a emplear para recorrer las propiedades de un objeto es la siguiente:

```
    for (nombreIndice in nombreObjeto) {
    ... ejecución de sentencias …
}
```

En el caso concreto de los arrays podemos escribir típicamente lo siguiente:

```
    for (indice in nombreDelArray) {
    ...
    realizar operaciones sobre nombreDelArray[indice]...
    ...
    }
```

En este código vemos cómo podemos usar recorrer un array con un for normal o, de forma equivalente, con un for in:

```
function ejemploForIn() {
var dato = [2, 6, 5, 1, 18, 44];
var msgForNormal = '';
var msgForIn = '';
//For normal
for (var i=0; i<dato.length; i++) { msgForNormal = msgForNormal + dato[i] + ' - '; }
//For in
for (i in dato) { msgForIn = msgForIn + dato[i] + ' - '; }
alert ('msgForNormal contiene ' + msgForNormal + ' y msgForIn contiene '+ msgForIn);
}
```

Incorpora el código dentro de una página web y comprueba cómo el resultado que obtienes es el siguiente: msgForNormal contiene 2 - 6 - 5 - 1 - 18 - 44 - y msgForIn contiene 2 - 6 - 5 - 1 - 18 - 44 -

Es decir, hemos recorrido el array con un for normal y con un for in. ¿Es mejor usar un for tradicional o un for in? Ambas formas resultan relativamente equivalentes y útiles, aunque en algunos casos específicos puede interesar más usar una forma u otra, y en algunos casos específicos no se obtienen los mismos resultados usando ambas formas. Nosotros nos inclinamos preferentemente por el uso del for tradicional, pero en determinadas circunstancias y para algunos usos especiales más avanzados (de los que hablaremos cuando estudiemos los objetos en JavaScript y el recorrido de las propiedades de un objeto) puede ser interesante usar for - in.

Una cuestión a tener en cuenta cuando se hacen usos más avanzados de for – in es que el orden en que se obtienen los elementos de la colección no tiene por qué coincidir con el orden de los índices. En el ejemplo anterior teníamos un array var dato = [2, 6, 5, 1, 18, 44]; y el orden esperado de recorrido es 2 - 6 - 5 - 1 - 18 – 44. Pero el for in no necesariamente respeta este orden, podría hacer un recorrido en un orden aparentemente aleatorio como 18 - 6 - 44 - 1 – 5. Con los arrays probablemente no ocurra esto, pero con otro tipo de colecciones sí puede ocurrir. Por tanto al usar un for – in podemos estar seguros de que se van a recorrer todos los elementos en la colección, pero no podemos asegurar que ese recorrido vaya a seguir un orden determinado. Si el orden es importante, será mejor usar un for tradicional u otro tipo de control.

Otra consideración a tener en cuenta es el rendimiento o velocidad de ejecución esperable cuando utilizamos un for – in respecto al que obtenemos cuando usamos un for tradicional. En general, si se trata de un array pequeño las diferencias pueden resultar inapreciables. Sin embargo, con arrays de muchos datos la ejecución es mucho más rápida utilizando un for tradicional.

EJERCICIO

1) Genera un script que pida cinco números al usuario usando un bucle for normal (usa prompt para pedir los datos y conviértelos a valor numérico posteriormente). Almacena los números introducidos por el usuario en un array. A continuación usando un for in que recorra el array, muestra un mensaje informando del resultado de multiplicar cada uno de los números por 3. Ejemplo:

Se pedirán al usuario cinco números, supongamos que introduce 1, 3, 9, 10 y 7

A continuación se mostrará el mensaje:

Multiplicamos por 5 los números introducidos: 1*5 = 5, 3*5 = 15, 9*5 = 45, 10*5 = 50 y 7*5 =35.

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01136E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

NODELIST JAVASCRIPT. DIFERENCIAR NODELIST Y ARRAY. ACCEDER AL TEXTO DE NODOS CON TEXTCONTENT, INNERTEXT (CU01136E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº36 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DIFERENCIA ENTRE NODELIST Y ARRAY

Sabemos que instrucciones como getElementsByTagName nos devuelven una colección de nodos que en algunos momentos se denomina "array" de nodos. Sin embargo, lo que devuelven este tipo de métodos no es exactamente un array, sino una estructura de datos denominada NodeList.

El hecho de que getElementsByTagName, getElementsByName, querySelectorAll no devuelvan un array puede llevarnos a confusión en algunos momentos. Por ejemplo, podemos pensar en recorrer un NodeList usando un bucle for – in, y no obtener los resultados esperados porque una colección NodeList no admite el uso de for – in (sí admite sin embargo el uso de un for tradicional). Pero además del for in, hay otras posibilidades propias de los arrays que se pierden cuando se usan NodeLists. Por ello en algunos casos puede resultar de interés la conversión de un NodeList en un array.

Una peculiaridad de los NodeList es que son colecciones dinámicas cuyo contenido se actualiza automáticamente cuando la página web cambia dinámicamente. Supongamos una página web donde tenemos cinco elementos div y usamos var nodosDiv = getElementsByTagName('div'); para rescatarlos. En ese momento la colección nodosDiv tiene 5 elementos. Supongamos ahora (no vamos a indicar cómo hacerlo, con la idea nos basta), que usando JavaScript añadimos un nuevo elemento div de modo que ahora en la página web tenemos 6 elementos div. Debido al carácter dinámico de los nodeLists, nodosDiv pasa automáticamente a tener 6 elementos. Sin embargo, si trabajáramos con arrays, debido a su carácter estático, el array seguiría teniendo 5 elementos (a no ser que a través de código añadiéramos instrucciones expresas para incorporar el nuevo elemento).

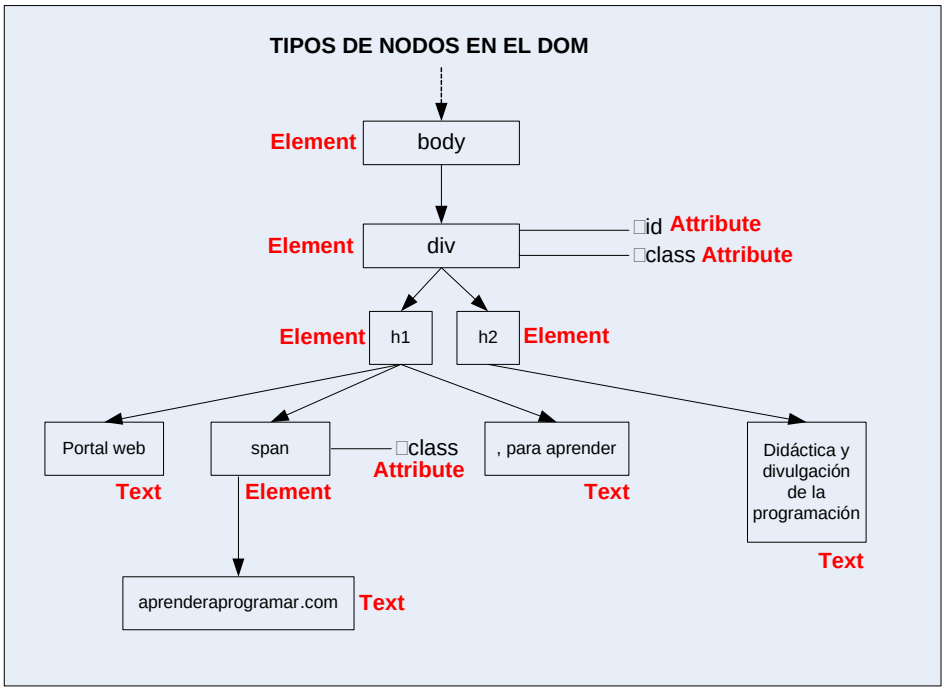
Por tanto los NodeLists podemos decir que tienen ciertas similitudes con los arrays y ciertas diferencias con estos. Los NodeList no tienen algunas posibilidades que tienen los arrays, pero a cambio son dinámicos, lo que puede resultar de gran interés en determinadas circunstancias.

ACCEDER AL TEXTO DENTRO DE NODOS: TEXTCONTENT (INNERTEXT)

Hemos visto que es posible acceder al contenido de un nodo del DOM tipo texto usando nodeValue. Por ejemplo una expresión como document.childNodes[1].childNodes[2].nodeValue nos puede permitir extraer el texto de un nodo de tipo texto.

También podemos acceder a nodos del DOM y extraer texto usando expresiones como document.firstChild.firstChild.nextSibling.nodeValue, o combinando expresiones de este tipo con el uso de childNodes.

Pero dado cómo se genera la estructura de nodos del DOM, muchas veces el acceso a nodos de tipo texto resulta complicado.



El esquema anterior representa que dentro del elemento div se encuentra el texto "Portal web aprenderaprogramar.com, para aprender. Didáctica y divulgación de la programación". Sin embargo ese texto se encuentra repartido por diferentes nodos debido a la presencia de subdivisiones dentro del elemento div creadas por h1, h2, span, etc.

Si queremos extraer (o definir) el texto contenido en todos los nodos derivados de un nodo padre podemos usar la propiedad textContent de los nodos.

La sintaxis a emplear normalmente será del tipo:

```
var textoEnElNodoYSusHijos = nombreDelNodo.textContent;
```

También podemos escribir nombreDelNodo.textContent = "El ganador es Barack Obama";

Es importante resaltar que con textContent se extrae no sólo el texto directamente asociado al nodo, sino también el texto contenido en otros nodos hijos.

En el ejemplo gráfico anterior, si extraemos el textContent del nodo div obtendremos "Portal web aprenderaprogramar.com, para aprender. Didáctica y divulgación de la programación", que incluye el texto encontrado en todos sus nodos hijos.

Hay una alternativa de funcionamiento muy similar a textContent que ha sido utilizada por algunos navegadores: innerText. Su comportamiento es muy similar al de textContent, aunque no exactamente igual. innerText no es reconocido por todos los navegadores y no se considera un estándar válido, por ello no le prestaremos más atención y no lo usaremos. No obstante, hemos considerado conveniente citarlo por si lo encuentras mientras revisas el código en alguna página web.

A modo de resumen: usaremos textContent para extraer el contenido de texto dentro de un nodo y sus nodos hijos.

RECORRER NODOS DEL DOM

Es frecuente que se escriba que var elementosDiv = document.getElementsByTagName('div') nos devolverá un array con todos los nodos de tipo element cuya etiqueta sea div, empezando con índice cero: elementosDiv[0], elementosDiv[1], elementosDiv[2], elementosDiv[3] …

Si esto es así, debería ser posible recorrer la colección de nodos obtenidos usando un for in. Sin embargo, es posible que nos encontremos que el for in no sirva para recorrer los nodos obtenidos por un método de selección de nodos del DOM. ¿Por qué?

Vamos a ver primero un ejemplo de código donde podemos comprobarlo y después lo comentaremos. Escribe este código y guárdalo con un nombre como ejemplo.html. Aquí usamos textContent, innerText, bucles for normales y bucles for in. Comprueba los resultados en tu navegador:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; margin:25px;}
#pulsador {padding:15px; width: auto; display: inline-block;
margin: 25px; cursor: pointer;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#pulsador:hover, #marcaOperadores:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
function ejemploForIn() {
var nodoSpan = document.getElementsByTagName('span');
var msg = 'Primer for \n';
alert ('items en nodoSpan son ' + nodoSpan.length);
//Primer for
for (var i=0; i< nodoSpan.length; i++) { msg = msg + i + ': contiene ' +nodoSpan[i].textContent + ' , y nodeValue
'+nodoSpan[i].firstChild.nodeValue +'  ** \n'; }
alert ('Contenido de los nodos span \n ' + msg);
msg = 'Segundo for \n';
//Segundo for
for (var i=0; i< nodoSpan.length; i++) { msg = msg + i + ': contiene ' +nodoSpan[i].innerText + '** \n'; }
alert ('Contenido de los nodos span \n ' + msg);
msg = 'Tercer for \n';
//Tercer for
for (i in nodoSpan) { msg = msg + i + ': contiene ' + nodoSpan[i].textContent + ' -- \n'; }
alert ('Contenido de los nodos span \n ' + msg);
}
</script>
</head>
<body>
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo for in JavaScript</h3>
</div>
<div id="calculadora"><span>7</span><span>+</span><span>4</span><span>-</span>
                        <span>1</span><span>÷</span><span>0</span><span>.</span>
                        <span>=</span><span>x <p>Genera un <b>nuevo</b> nodo</p></span>
</div>
<div id ="pulsador" onclick="ejemploForIn()"> Probar </div>
</body></html>
```

El resultado esperado depende del navegador que utilicemos. Aquí vamos a mostrar los resultados obtenidos en dos navegadores diferentes:

Resultados navegador 1	Resultados navegador 2
items en nodoSpan son 10 Contenido de los nodos span **Primer for** 0: contiene 7 , y nodeValue 7 ** 1: contiene + , y nodeValue + ** 2: contiene 4 , y nodeValue 4 ** 3: contiene - , y nodeValue - ** 4: contiene 1 , y nodeValue 1 ** 5: contiene ÷ , y nodeValue ÷ ** 6: contiene 0 , y nodeValue 0 ** 7: contiene . , y nodeValue . ** 8: contiene = , y nodeValue = ** 9: contiene x Genera un nuevo nodo , y nodeValue x ** Contenido de los nodos span **Segundo for** 0: contiene undefined // 1: contiene undefined // 2: contiene undefined // 3: contiene undefined // 4: contiene undefined // 5: contiene undefined // 6: contiene undefined // 7: contiene undefined // 8: contiene undefined // 9: contiene undefined // Contenido de los nodos span **Tercer for** 0: contiene 7 -- 1: contiene + -- 2: contiene 4 -- 3: contiene - -- 4: contiene 1 -- 5: contiene ÷ -- 6: contiene 0 -- 7: contiene . -- 8: contiene = -- 9: contiene x Genera un nuevo nodo -- item: contiene undefined -- namedItem: contiene undefined -- @@iterator: contiene undefined -- length: contiene undefined --	items en nodoSpan son 10 Contenido de los nodos span **Primer for** 0: contiene 7 , y nodeValue 7 ** 1: contiene + , y nodeValue + ** 2: contiene 4 , y nodeValue 4 ** 3: contiene - , y nodeValue - ** 4: contiene 1 , y nodeValue 1 ** 5: contiene ÷ , y nodeValue ÷ ** 6: contiene 0 , y nodeValue 0 ** 7: contiene . , y nodeValue . ** 8: contiene = , y nodeValue = ** 9: contiene x Genera un nuevo nodo , y nodeValue x ** Contenido de los nodos span **Segundo for** 0: contiene 7 // 1: contiene + // 2: contiene 4 // 3: contiene - // 4: contiene 1 // 5: contiene ÷ // 6: contiene 0 // 7: contiene . // 8: contiene = // 9: contiene x Genera un nuevo nodo // **Tercer for** 0: contiene 7 -- 1: contiene + -- 2: contiene 4 -- 3: contiene - -- 4: contiene 1 -- 5: contiene ÷ -- 6: contiene 0 -- 7: contiene . -- 8: contiene = -- 9: contiene x Genera un nuevo nodo – length: contiene undefined -- item: contiene undefined -- namedItem: contiene undefined --

Del resultado obtenido podemos señalar lo siguiente:

1) Se pueden obtener distintos resultados ejecutando un mismo código en diferentes navegadores, lo cual es un problema a la hora de crear desarrollos web. Debemos asumirlo y tenerlo en cuenta en la medida de lo posible, ya que no hay forma de esquivarlo.

2) El resultado de recorrer una colección de nodos con document.getElementsByTagName('span'); no es el mismo si usamos un for tradicional que si usamos un for in. Esto nos indica que las colecciones de nodos que devuelven estos métodos no son arrays propiamente dichos. Pero si no son arrays, ¿entonces qué son? Son objetos que tienen similitudes con los arrays, denominados NodeList o listas de nodos. Los NodeLists son objetos a veces se dice que son objetos de tipo array (array-like objects). Sin embargo, no disponen de todas las posibilidades de que dispone un array. Una de las posibilidades de que no disponen, es el recorrido mediante bucles for in. Por eso cuando intentamos recorrer una colección de nodos con un for in obtenemos resultados extraños, que incluso pueden ser diferentes de un navegador a otro.

3) Un navegador reconoce innerText pero el otro no. Como hemos dicho, preferiremos usar textContent como método para extraer texto, pero nos podremos encontrar código donde aparezca innerText. Algunos navegadores tratan innerText como si fuera textContent, otros de forma ligeramente diferente, y otros directamente no reconocen innerText.

EJERCICIO

Crea un documento html con un texto en una etiqueta h1 como "Ejercicio curso aprenderaprogramar.com" y un div a continuación. Genera un script que pida cinco números al usuario usando un bucle for normal (usa prompt para pedir los datos y conviértelos a valor numérico posteriormente). Almacena los números introducidos por el usuario en un array. A continuación, accede al nodo del div y establece que muestre un texto informando del resultado de multiplicar cada uno de los números por 3 (para ello usa textContent). Ejemplo:

Al cargar la página aparecerá: Ejercicio curso aprenderaprogramar.com
… (div vacío)

Se pedirán al usuario cinco números, supongamos que introduce 1, 3, 9, 10 y 7

A continuación en la página se visualizará: Ejercicio curso aprenderaprogramar.com
Multiplicamos por 5 los números introducidos: 1*5 = 5, 3*5 = 15, 9*5 = 45, 10*5 = 50 y 7*5 =35.

Para comprobar si tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01137E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

VERSIONES DE JAVASCRIPT. DIFERENCIA CON ECMASCRIPT-262 O ISO/IEC. ESPECIFICACIÓN OFICIAL. ALGO DE HISTORIA. (CU01137E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº37 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

VERSIONES DE JAVASCRIPT

Cuando comenzamos el curso indicamos que íbamos a centrarnos en aprender las bases fundamentales para usar JavaScript del lado del cliente, sin entrar a detallar todas las instrucciones y detalles del lenguaje, ni las diferencias entre versiones. No obstante, en este momento vamos a hacer un break para conocer algunos aspectos de interés.

ALGO DE HISTORIA. DE NETSCAPE A ECMA INTERNATIONAL.

En los años 90 comenzó a usarse a gran escala internet por parte de una gran masa de usuarios, después de una etapa introductoria donde se produjo el desarrollo y nacimiento de ese gran fenómeno cultural. En las etapas iniciales de internet no existían wifi, ADSL, fibra óptica o cualquiera de los adelantos que hoy día nos parecen comunes. Lo más común era la conexión a través de la línea telefónica fija de los hogares (que quedaba "interrumpida" mientras se hacía uso de internet) a través de un módem, y las velocidades de transmisión de datos eran muy lentas. La navegación se hacía con frecuencia tediosa debida a la lentitud de carga de las páginas, y cómo mejorar la experiencia de navegación, comodidad y velocidad eran un asunto candente.

Este asunto ha tenido respuesta en múltiples vertientes: tendido de nuevas redes de cableado, aparición de las redes inalámbricas, mejora de los servidores, mejora de los computadores personales y mejora de los navegadores y lenguajes de programación son algunos de los que podemos citar.

La creación de JavaScript se le atribuye a Brendan Eich, matemático y tecnólogo estadounidense. Eich fue contratado por Netscape Communications Corporation, en aquel momento la empresa que desarrollaba el navegador más difundido de la época: Netscape. Dentro de Netscape y como forma de dar respuesta a la mejora de la experiencia de navegación y velocidad de operación de las páginas web fue creado JavaScript en 1995.

La adopción del nombre "JavaScript" se atribuye a motivos de marketing, tratando de aprovechar que Java era un lenguaje de programación que estaba adquiriendo gran popularidad y que un nombre similar podía hacer que el nuevo lenguaje fuera atractivo. Pero salvando algunas similitudes, ambos lenguajes son bien distintos. Su principal parecido podemos decir que es el nombre y algunos aspectos de sintaxis, ya que su finalidad y filosofía son muy distintos.

Entre las diferentes empresas y lenguajes que pugnaban por convertirse en el lenguaje del lado del servidor más usado fue emergiendo JavaScript como ganador. A finales de 1996 Netscape informó que traspasaba la responsabilidad de definir y estandarizar JavaScript a una organización internacional para el desarrollo de estándares en el sector de tecnologías de la información y comunicación denominada Ecma International. Esta organización tuvo su origen en la European Computer Manufacturers Association (ECMA, o Asociación Europea de fabricantes de computadores). En realidad ya no se trata

de una organización europea, sino plenamente internacional, que se define como un organismo que asocia a productores de tecnologías de información y comunicación.

Ecma International es el organismo actualmente responsable de la definición de la especificación oficial JavaScript, a la que se denomina ECMAScript. En Ecma International participan grandes empresas relacionadas con internet, computadores y telecomunicaciones, entre las que podemos citar Microsoft, Apple, Google, Yahoo, Toshiba, IBM, Hitachi, Fujitsu, Intel, AMD, Adobe Systems, eBay, Hewlett Packard, Konica Minolta, Sony, etc. y también universidades e instituciones sin ánimo de lucro como Stanford University, UEC Tokyo University, Mozilla Foundation y otras.

Ecma International trabaja en la estandarización en diversos frentes: lenguajes de programación (C#, Eiffel, C++, etc.), redes de telecomunicaciones, redes inalámbricas, seguridad de productos electrónicos, aspectos ambientales relacionados con productos electrónicos y las telecomunicaciones, contaminación acústica, campos electromagnéticos, etc. Cada estándar producido por Ecma International se plasma en un documento y se le da una numeración. Por ejemplo al estándar para JavaScript se le ha denominado Standard ECMA-262.

Uno de los campos de trabajo de Ecma International es generar la especificación oficial de JavaScript, a la que se denomina ECMAScript ó Standard ECMA-262. Dicha especificación es adoptada por las diferentes empresas o entidades que desarrollan los navegadores web para que los navegadores reconozcan JavaScript y éste pueda ser usado por todos los programadores web esperando (al menos teóricamente) poder usar todos el mismo lenguaje y obtener los mismos resultados. La realidad no alcanza esta suposición ideal, y lo cierto es que aunque todos los navegadores usan JavaScript a partir de la especificación ECMAScript, nos podemos encontrar con matices, diferentes especificidades y diferentes resultados entre navegadores. Hay que reconocer sin embargo, que ha habido grandes avances en lo que a estandarización se refiere.

VERSIONES DE JAVASCRIPT

JavaScript tiene hoy día como especificación oficial la especificación que desarrolla Ecma International, denominada ECMASCript. Este estándar es desarrollado por un grupo de trabajo que define como objetivos la estandarización del lenguaje de programación ECMAScript, lenguaje de propósito general, multiplataforma y neutral respecto de las empresas. La tarea de Ecma International comprende la definición de la sintaxis del lenguaje, su semántica, así como de las librerías y tecnologías suplementarias que facilitan el desarrollo con este lenguaje (entre ellas API soporte para JavaScript).

Las evolución de las versiones de JavaScript podemos resumirla así:

- 1995. Primeras versiones de JavaScript, todavía con nombres provisionales como Mocha, LiveScript.
- 1997. Definición del primer estándar JavaScript a cargo de Ecma International que fue denominado ECMA-262 first edition también denominado JavaScript 1.2.
- 1998. Aparición del segundo estándar JavaScript denominado ECMA-262 second edition también denominado JavaScript 1.3.
- 2000. Aparición de la especificación del estándar JavaScript denominado ECMA-262 third edition también denominado JavaScript 1.5.
- 2010. Aparición de la especificación del estándar JavaScript denominado ECMA-262 fifth edition también denominado JavaScript 1.8.5.
- 2019. Fecha prevista para ECMA-262 sixth edition.

Hay algunas dudas que nos pueden surgir.

¿Por qué no existe ECMA-262 fourth edition? Esta especificación no llegó nunca a publicarse, por lo que se produjo directamente el salto desde ECMA-262 3rd edition hasta ECMA-262 5th edition.

¿Qué diferencia hay entre ECMAScript y JavaScript? Podemos decir que ambas cosas son la misma, si bien podríamos distinguir matices. ECMAScript es un estándar definido por ECMA International que es luego implementado por distintas instituciones o empresas en sus navegadores. Para aludir al lenguaje o implementación que exactamente reconoce un navegador suele usarse el término JavaScript en lugar de ECMAScript. La Mozilla Foundation, creadora del navegador Mozilla Firefox, mantiene una numeración de versiones (1.7, 1.8, 1.8.1, …1.8.5, 1.9, 1.15, 1.20 …) que indica la evolución del intérprete JavaScript dentro del navegador.

¿Por qué este lío de nombres en lugar de usar un único nombre? Además los citados también podríamos nombra a JavaScript de otras maneras: JScript, ISO/IEC 16262:2011, etc. Todo tiene su razón de ser en motivos históricos, el hecho de que algunos nombres son marcas registradas y no pueden ser usados sin el permiso del propietario de la marca, luchas entre organismos, empresas, la terminología propia de la estandarización, etc.

ESPECIFICACIÓN OFICIAL DE JAVASCRIPT

La especificación oficial de JavaScript (ECMA-262) puede leerse o descargarse en formato pdf accediendo a la dirección web http://www.ecma-international.org/

Ahí también se pueden encontrar especificaciones oficiales de estándares relacionados con JavaScript como el estándar ECMA-357 que define “ECMAScript para XML”, la integración de JavaScript con XML.

¿Qué interés tiene para nosotros la especificación oficial? Tiene un interés relativo, ya que se trata de un documento denso, extenso y que aborda todos los detalles de una especificación oficial que puede presentar algunas divergencias con lo que nos encontremos en la realidad trabajando con los navegadores.

Nosotros lo consideraremos un documento de referencia donde podremos hacer consultas puntuales (sobre todo a medida que usemos más en profundidad el lenguaje).

En este curso nos centramos en aspectos básicos del lenguaje y en comprender su lógica, las buenas prácticas y fundamentos. No nos resulta de excesivo interés remitirnos a la especificación oficial, ni tener en cuenta las versiones de la especificación oficial ni de las implementaciones que use cada navegador.

Este pequeño recorrido por la historia y versiones de JavaScript lo hemos considerado adecuado para poder entender mejor la terminología que nos podemos encontrar cuando leamos información sobre JavaScript en libros, revistas o páginas web. Las versiones de JavaScript suelen ser incrementales: es decir, mantienen los fundamentos y amplían las posibilidades del lenguaje. Nosotros con este curso tenemos el objetivo de aprender los fundamentos que nos permitan desarrollar JavaScript sin importarnos más que lo justo el navegador y las versiones. Así que continuamos.

Próxima entrega: CU01138E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

INNERHTML JAVASCRIPT. MODIFICAR TEXTO, HTML O ELEMENTOS COMO UN DIV EN TIEMPO REAL. THIS: ACCEDER AL NODO ACTUAL. (CU01138E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº38 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

INNERHTML JAVASCRIPT

La propiedad innerHTML de los nodos de tipo elemento nos permite acceder al código HTML comprendido entre las etiquetas de apertura y cierre que definen el nodo y modificarlo si lo deseamos. La aplicación más sencilla puede ser modificar un texto, pero el verdadero potencial de innerHTML es el de modificar el código HTML en sí, no solo el texto.

Mucha gente usa innerHTML para modificar texto, pero realmente tenemos otras opciones para hacer modificaciones en texto: textContent o nodeValue nos pueden servir para ello.

La sintaxis a emplear normalmente será del tipo:

```
nodoAlQueAccedemos.innerHTML = 'Nuevo código HTML en el nodo';
```

Los nodos que admiten el uso de innerHTML son los tipo Element con capacidad para contener código HTML. Por tanto, no obtendremos resultado si tratamos de aplicarlo a otro tipo de nodos como los de tipo texto. Escribe este código en tu navegador, guárdalo con extensión html y comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head> <title>Curso JavaScript aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> body {font-family: sans-serif; text-align:center; }
div {width: 600px; background-color:yellow; overflow:hidden;margin: 5px auto;}
h2 {width: 80%; background-color:#DEB887; cursor:pointer; margin: 10px auto;  }</style>
</head>
<body><div><h2 onclick="alert(this.nodeName)">Haz click aquí para ver el nodeName de this</h2>
<h2 onclick="alert(this.nodeName)">Haz click <span style="color:blue;">aquí para ver el nodeName de this</span></h2>
<h2>Haz click <span onclick="alert(this.nodeName)" style="color:red;">aquí para ver el nodeName de this</span></h2>
<h2 onclick="alert(this.firstChild.nodeName)">Haz click aquí para ver el nodeName del hijo de h2</h2>
<h2 onclick="this.firstChild.nodeValue='Aaaaaaah!'">Haz click aquí para modificar el texto en el hijo del nodo h2 usando
NodeValue</h2>
<h2 onclick="this.textContent='Eeeeeeeh!'">Haz click aquí para modificar el texto en hijo derivado del nodo h2 usando
textContent</h2>
<h2 onclick="this.innerHTML='Siiiiiih!'">Haz click aquí para modificar el texto con innerHTML del nodo h2</h2>
<h2 onclick="alert('Contenido innerHTML actual es: '+this.innerHTML);
 this.innerHTML='<div style=\' width:50%; border-style:solid; background-color: #FF7F50;  \'>Ahamham!</div>'">
Haz click aquí para modificar el código con innerHTML del nodo h2</h2>
<h2 onclick="this.textContent='Uuuuuuh!'">Haz click <span style="color:blue;">pero no existe un textContent
modificable</span></h2>
<h2 onclick="alert('Obtenemos: '+this.textContent)">Haz click <span style="color:blue;">para ver el textContent</span> y
verás</h2>
<h2 onclick="this.firstChild.innerHTML='No podemos'">Si intentamos modificar un nodo texto con innerHTML...</h2>
</div></body></html>
```

REFERENCIAR UN NODO USANDO LA PALABRA CLAVE THIS

Para interpretar este código hemos de tener en cuenta que la palabra clave this aplicada dentro de un manejador de evento onclick nos devuelve el nodo de tipo Element definido por las etiquetas HTML donde se define el onclick. Por ejemplo: <h2 onclick="alert(this.nodeName)">Hola</h2> nos devuelve el nodeName del nodo definido por las etiquetas <h2> … </h2> y este nodeName es H2. Recordar que de acuerdo con la estructura de construcción del DOM, el nodo H2 tendrá como hijo un nodo de tipo texto cuyo nodeValue será Hola.

Los resultados esperados cuando visualizamos y comprobamos el código HTML que hemos puesto como ejemplo anteriormente son (comprueba tú si obtienes estos mismos resultados o no):

Cuando hacemos click sobre	Se muestra	Comentarios
Haz click aquí para ver el nodeName de this	H2	Es el nodo para el cual está definido el evento onclick
Haz click aquí para ver el nodeName de this	H2	Aunque el nodo H2 tenga un nodo hijo span, el nodo para el que está definido el evento onclick es H2 y por tanto this hace referencia a este nodo.
Haz click aquí para ver el nodeName de this	SPAN	Al haber modificado el onclick y asociarlo al span, el nodo referenciado por this pasa a ser el SPAN
Haz click aquí para ver el nodeName del hijo de h2	#text	Estamos aludiendo a firstChild de H2, que es el nodo de tipo texto que contiene el texto que se muestra en pantalla
Haz click aquí para modificar el texto en el hijo del nodo h2 usando NodeValue	Aaaaaaah!	Modificamos el texto accediendo al nodo de tipo texto y modificando su valor nodeValue
Haz click aquí para modificar el texto en hijo derivado del nodo h2 usando textContent	Eeeeeeeh!	Modificamos el texto usando textContent (texto contenido en los nodos derivados del referenciado)
Haz click aquí para modificar el texto con innerHTML del nodo h2	Siiiiiih!	Modificamos el texto modificando el código HTML interno al nodo
Haz click aquí para modificar el código con innerHTML del nodo h2	Ahamham!*	Se muestra el texto Ahamham! dentro de un rectángulo de color anaranjado
Haz click pero no existe un textContent modificable	--	No hay resultado. Se puede modificar el textContent si sólo existe texto asociado a un nodo hijo, pero en este caso el texto está repartido entre varios nodos hijos.
Haz click para ver el textContent y verás	Muestra un mensaje	Se muestra el mensaje: Obtenemos: Haz click para ver el textContent y verás. Esto demuestra que el textContent concatena los textos obtenidos del recorrido de los diferentes nodos sucesores donde aparezca texto.
Si intentamos modificar un nodo texto con innerHTML...	--	No hay resultado. No se puede modificar un nodo tipo texto usando innerHTML porque innerHTML aplica solo a los nodos de tipo Element.

* En este caso tenemos el siguiente comportamiento:

Al hacer un primer click

Se nos muestra un mensaje: “Contenido innerHTML actual es: Haz click aquí para modificar el código con innerHTML del nodo h2”. Esto refleja que dentro de la etiqueta h2 únicamente tenemos un texto.

Al hacer un segundo click

Se nos muestra un mensaje: “Contenido innerHTML actual es: Ahamham! ”

Esto refleja que el anterior contenido dentro de h2 ha desaparecido. Ahora dentro de la etiqueta h2 tenemos un div al que hemos aplicado estilos haciendo que muestre un color de fondo naranja, un borde, y que contiene el texto Ahamham!

Sin embargo, el código fuente o código HTML asociado a la página que nos facilita el navegador (que podemos visualizar accediendo a él a través de opciones de menú) no ha cambiado. Esto significa que se trata de cambios dinámicos que son controlados por el navegador y no se reflejan en que el código fuente “accesible” sea distinto en cada momento. O dicho de otra manera: la visualización de una página web puede no coincider con lo que veamos que indica el código fuente si dicho código fuente ha sido manipulado dinámicamente con JavaScript.

LA POTENCIALIDAD DE INNERHTML

innerHTML tiene una gran potencialidad debido a que nos permite generar código HTML dinámicamente y esto facilita la manipulación de páginas web y la creación de efectos hasta los límites de nuestra imaginación. Todo ello a pesar de que históricamente no formaba parte del lenguaje JavaScript. Se convirtió es un estándar de facto, es decir, no formando parte de la especificación oficial JavaScript sin embargo fue introducido y aceptado por todos los navegadores.

En la próxima entrega del curso veremos un ejemplo más avanzado de utilización de innerHTML para el control dinámico de procesos en el frontEnd. Si el término frontEnd te resulta extraño, ten en cuenta que suele usarse en alusión a código que se ejecuta del lado del cliente, mientras que backEnd suele usarse en alusión a procesos del lado del servidor. El trabajo con JavaScript se califica normalmente de trabajo en el frontEnd de las aplicaciones web.

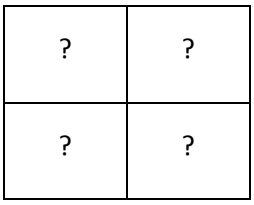
EJERCICIO

Crea un documento html con div central (“el cuadrado principal”) que contenga a su vez cuatro divs (los cuadrados secundarios). Debajo del cuadrado principal debe aparecer un botón con el texto “Avanzar”. Inicialmente los cuatro cuadrados tendrán fondo blanco y un signo ? de gran tamaño. Al pulsar en el botón por primera vez, el cuadrado superior izquierdo debe aparecer con fondo negro, texto blanco, y tener como texto “El”. Al pulsar de nuevo el botón, el cuadrado superior derecho debe aparecer con fondo blanco y texto negro y tener como texto “poder”. Al pulsar de nuevo el botón el cuadrado

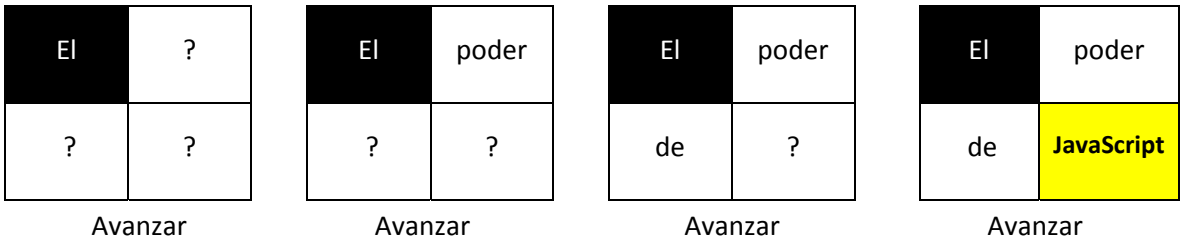
inferior izquierdo debe aparecer con fondo blanco y texto negro y tener como texto “de”. Al pulsar de nuevo el botón el cuadrado inferior derecho debe aparecer con fondo amarillo y texto negro y tener como texto “JavaScript”. Si se pulsa nuevamente el botón avanzar debe mostrarse un mensaje que indique “No es posible avanzar más”.

Aquí resumimos cómo tiene que ser el cambio:

Inicialmente:



A medida que se pulse el botón irá apareciendo:



Para comprobar si tu código y tus respuestas son correctas puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01139E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EJEMPLO INNERHTML JAVASCRIPT. EJERCICIO RESUELTO. MODIFICAR CAMPOS DE UNA TABLA DINÁMICA: EDITAR CAMPOS. (CU01139E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº39 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

EJERCICIO RESUELTO CON INNERHTML JAVASCRIPT

En la anterior entrega del curso hemos visto la sintaxis y forma de uso básica de innerHTML en JavaScript. Ahora vamos a plantear y dar la solución a un ejercicio en el que tratamos de hacer un uso más avanzado de innerHTML. En concreto, dada una tabla con datos, veremos cómo podemos convertir los datos en editables.

ENUNCIADO DEL EJERCICIO

Con este ejercicio queremos por un lado utilizar los conocimientos adquiridos a lo largo del curso, y por otra parte hacernos a la idea de que siempre necesitaremos buscar información adicional para resolver algunas cuestiones que no conocemos a priori. Cuando se trabaja en desarrollos web es imposible conocer toda la sintaxis y las posibles instrucciones, propiedades, etc. y por tanto es necesario recurrir a la consulta de libros, revistas o páginas web para obtener información adicional a nuestro conocimiento.

El enunciado del ejercicio es el siguiente:

Una página web carga una tabla con datos relativos al contenido nutricional de los alimentos como se muestra a continuación:

Contenido nutricional por cada 100 g de alimento.

Alimento	Calorías (kCal)	Grasas (g)	Proteína (g)	Carbohidratos (g)	Acciones
Arándano	49	0.2	0.4	12.7	Editar
Plátano	90	0.3	1.0	23.5	Editar
Cereza	46	0.4	0.9	10.9	Editar
Fresa	37	0.5	0.8	8.3	Editar

Se desea que al pulsar en el texto Editar de la columna Acciones, ocurra lo siguiente:

a) El texto de esa columna que ponía <<Editar>> en color azul, será reemplazado por el texto <<En edición>> en color gris o negro.

b) Los datos en la fila correspondiente se convertirán en casillas de texto editables de modo que el usuario pueda modificar los datos de esa fila.

c) Debe aparecer en la parte inferior de la tabla el texto: <<Pulse Aceptar para guardar los cambios o cancelar para anularlos>> y dos botones: Aceptar y Cancelar, que podrán ser pulsados por el usuario para Aceptar los cambios o para cancelar.

Si el usuario pulsa en el botón Aceptar los datos de la fila en edición deben ser enviados a una url de destino por el método get. Por ejemplo si la url de destino es aprenderaprogramar.com y editáramos la fila correspondiente a Fresa y pulsamos aceptar, el navegador debe enviarnos a una url como esta:

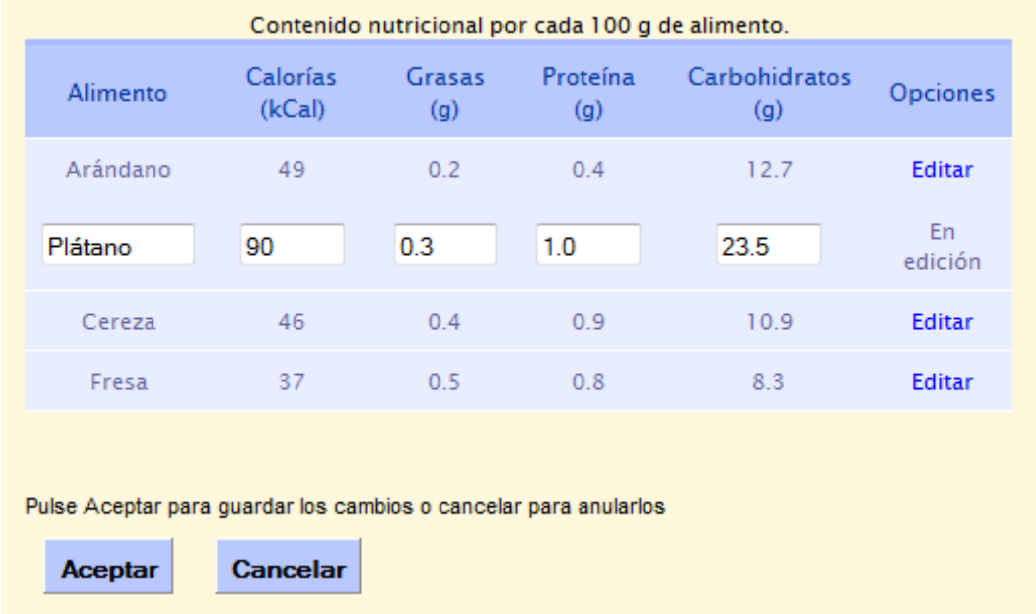
http://aprenderaprogramar.com/?alimento=Fresa&calorias=57&grasas=1.5&proteina=0.4&carbohidratos=12.8

Atención: los datos que se deben enviar a la url de destino son los datos editados por el usuario, no los datos originales de la tabla.

Si el usuario pulsa en Cancelar se recargará la tabla original (de forma que no habrá ninguna fila en edición).

Si estando seleccionada una fila en edición el usuario pulsa sobre Editar en otra fila, se mostrará un mensaje indicando lo siguiente: “Solo se puede editar una línea. Recargue la página para poder editar otra”.

La siguiente imagen refleja la idea de lo que se quiere conseguir:



En este ejercicio suponemos que la tabla va a cargar datos desde una base de datos, pero como nuestro objetivo no es el manejo de bases de datos ni de lenguajes del lado del servidor sino ver aplicaciones de JavaScript, nos limitaremos a crear la tabla manualmente con el código HTML necesario.

ORIENTACIONES PARA LA SOLUCIÓN: PASO 1

Intenta resolver el ejercicio con los conocimientos adquiridos a lo largo del curso y realizando alguna consulta adicional en internet. Es bueno intentar enfrentarse a los problemas por uno mismo y tratar de resolverlos por uno mismo, ya que favorece el aprendizaje como programadores. Ten en cuenta que este ejercicio posiblemente no tenga una solución única, sino que hay distintas formas de plantearlo y distintas formas de resolverlo. Intenta crear tu propia solución y después compárala con la que indicaremos nosotros tratando de ver las ventajas e inconvenientes de una y otra forma de resolver el problema.

ORIENTACIONES PARA LA SOLUCIÓN: PASO 2

Si necesitas una orientación aquí te indicamos a grandes rasgos la solución (complementa estas indicaciones buscando en internet aquello que puedas necesitar) que hemos creado nosotros para que trates de implementarla por ti mismo. Los pasos a seguir serían los siguientes:

1) Crea un archivo html de nombre testEdicionTablas.html donde tengas el código HTML de la tabla, de modo que el texto editar esté dentro de etiquetas span que respondan al evento onclick="transformarEnEditable(this)". Debajo de la tabla añade un div vacío. El div vacío servirá para añadirle lo que se debe mostrar cuando el usuario pulsa sobre un botón editar.

2) Crea un archivo css de nombre estilos.css donde tengas los estilos para la tabla y para las etiquetas span con el texto Editar, botones, etc.

3) Crea un archivo js de nombre functions.js donde tengas una variable global de nombre <<editando>> y tipo booleano que te permita saber si se encuentra en edición alguna fila o no.

4) Crea una función <<function transformarEnEditable(nodo)>> que debe encargarse de transformar en editables los datos de una fila definiéndolos en etiquetas <input> … </input> usando innerHTML. Además debe añadir el texto << Pulse Aceptar para guardar los cambios o cancelar para anularlos >> y los dos botones: Aceptar y Cancelar. Si ya había una fila en edición, en lugar de estos se deberá mostrar el mensaje <<Solo se puede editar una línea. Recargue la página para poder editar otra>>

5) Crea una función <<function capturarEnvio()>> que sirva para que cuando el usuario pulse el botón Aceptar, se cree un formulario con datos ocultos (hidden) capturados de las casillas de la fila en edición y se envíen a la url de destino por el método get. El formulario lo añadirás con innerHTML y el envío lo puedes realizar con la sentencia javascript document.nombreDelformulario.submit();

6) Crea la función anular que recargue la página original cuando el usuario pulse el botón cancelar. Esto puedes hacerlo con la sentencia JavaScript window.location.reload();

SOLUCIÓN PROPUESTA

A continuación indicamos el código de la solución propuesta. Ten en cuenta que esta no tiene por qué considerarse ni la única solución ni la mejor solución. Es posible que tú hayas definido una solución mejor que esta. Compara tu solución con la propuesta y trata de ver las ventajas e inconvenientes de cada una de ellas.

Archivo testEdicionTablas.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html> <head> <title>Portal web - aprenderaprogramar.com</title> <meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="estilos.css">
<script type="text/javascript" src="functions.js"></script>
</head>
<body>
<table>
<caption>Contenido nutricional por cada 100 g de alimento.</caption>
<tr> <th>Alimento</th> <th>Calorías (kCal)</th> <th>Grasas (g)</th>
<th>Proteína (g)</th> <th>Carbohidratos (g)</th> <th>Opciones</th>
</tr>
<tr> <td>Arándano</td> <td>49</td> <td>0.2</td>
<td>0.4</td> <td>12.7</td> <td><span class="editar" onclick="transformarEnEditable(this)">Editar</span></td>
</tr>
<tr> <td>Plátano</td> <td>90</td> <td>0.3</td>
<td>1.0</td> <td>23.5</td> <td><span class="editar" onclick="transformarEnEditable(this)">Editar</span></td>
</tr>
<tr> <td>Cereza</td> <td>46</td> <td>0.4</td>
<td>0.9</td> <td>10.9</td> <td><span class="editar" onclick="transformarEnEditable(this)">Editar</span></td>
</tr>
<tr> <td>Fresa</td> <td>37</td> <td>0.5</td>
<td>0.8</td> <td>8.3</td> <td><span class="editar" onclick="transformarEnEditable(this)">Editar</span></td>
</tr>
</table>
<div id="contenedorForm">
</div></body></html>
```

Archivo estilos.css:

```
/* Curso JavaScript estilos aprenderaprogramar.com*/
body {font-family: Arial, Helvetica, sans-serif; background-color: #FFF8DC;}

table { font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-Serif; font-size: 12px; margin: 45px; width: 550px;
   text-align: center; border-collapse: collapse; }

th { font-size: 13px; font-weight: normal; padding: 8px; background: #b9c9fe; border-top: 4px solid #aabcfe;
   border-bottom: 1px solid #fff; color: #039; }

td { padding: 8px; background: #e8edff; border-bottom: 1px solid #fff; color: #669; border-top: 1px solid transparent; }

tr:hover td { background: #d0dafd; color: #339; }

.editar {color: blue; cursor:pointer;}

#contenedorForm {margin-left: 45px; font-size:12px;}

.boton {   color: black; padding: 5px; margin: 10px;
  background-color: #b9c9fe;
  font-weight: bold; }
```

Archivo functions.js:

```
/*Curso JavaScript aprenderaprogramar.com */
var editando=false;

function transformarEnEditable(nodo) {
//El nodo recibido es SPAN
if (editando == false) {
var nodoTd = nodo.parentNode; //Nodo TD
var nodoTr = nodoTd.parentNode; //Nodo TR
var nodoContenedorForm = document.getElementById('contenedorForm'); //Nodo DIV
var nodosEnTr = nodoTr.getElementsByTagName('td');
var alimento = nodosEnTr[0].textContent; var calorias = nodosEnTr[1].textContent;
var grasas = nodosEnTr[2].textContent; var proteina = nodosEnTr[3].textContent;
var carbohidratos = nodosEnTr[4].textContent; var opciones = nodosEnTr[5].textContent;
var nuevoCodigoHtml = '<td><input type="text" name="alimento" id="alimento" value="'+alimento+'" size="10"></td>'+
'<td><input type="text" name="calorias" id="calorias" value="'+calorias+'" size="5"</td>'+
'<td><input type="text" name="grasas" id="grasas" value="'+grasas+'" size="5"</td>'+
'<td><input type="text" name="proteina" id="proteina" value="'+proteina+'" size="5"</td>'+
'<td><input type="text" name="carbohidratos" id="carbohidratos" value="'+carbohidratos+'" size="5"</td> <td>En
edición</td>';

nodoTr.innerHTML = nuevoCodigoHtml;

nodoContenedorForm.innerHTML = 'Pulse Aceptar para guardar los cambios o cancelar para anularlos'+
'<form name = "formulario" action="http://aprenderaprogramar.com" method="get" onsubmit="capturarEnvio()"
onreset="anular()">'+
'<input class="boton" type = "submit" value="Aceptar"> <input class="boton" type="reset" value="Cancelar">';
editando = "true";}
else {alert ('Solo se puede editar una línea. Recargue la página para poder editar otra');
}
}

function capturarEnvio() {
var nodoContenedorForm = document.getElementById('contenedorForm'); //Nodo DIV
nodoContenedorForm.innerHTML = 'Pulse Aceptar para guardar los cambios o cancelar para anularlos'+
'<form name = "formulario" action="http://aprenderaprogramar.com" method="get" onsubmit="capturarEnvio()"
onreset="anular()">'+
'<input type="hidden" name="alimento" value="'+document.querySelector('#alimento').value+'">'+
'<input type="hidden" name="calorias" value="'+document.querySelector('#calorias').value+'">'+
'<input type="hidden" name="grasas" value="'+document.querySelector('#grasas').value+'">'+
'<input type="hidden" name="proteina" value="'+document.querySelector('#proteina').value+'">'+
'<input type="hidden" name="carbohidratos" value="'+document.querySelector('#carbohidratos').value+'">'+
'<input class="boton" type = "submit" value="Aceptar"> <input class="boton" type="reset" value="Cancelar">';
document.formulario.submit();
}

function anular() {
window.location.reload();
}
```

Revisa el código propuesto, ejecútalo en tu navegador y comprueba su funcionamiento. Aunque usamos algunos elementos novedosos como el evento onreset o la invocación window.location.reload(), debes ser capaz de comprender el código HTML, CSS y JavaScript, así como todos los procesos que tienen lugar y cómo se controla el flujo de la aplicación y los cambios que tienen lugar dinámicamente. Si tienes dudas, consulta en los foros aprenderaprogramar.com.

Próxima entrega: CU01140E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT CREATEELEMENT, CREATETEXTNODE, APPENDCHILD. MODIFICAR PROPIEDAD ONCLICK CON JAVASCRIPT (CU01140E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº40 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

CREAR NODOS DEL DOM

Hasta ahora hemos visto cómo acceder a nodos del DOM y modificar sus propiedades. ¿Es posible crear nodos y añadirlos en puntos específicos del árbol del DOM? Sí, esto es posible gracias a métodos disponibles para el objeto document de JavaScript.

CREAR UN NODO

Un nodo de tipo Element se crea invocando el método createElement del objeto document de JavaScript. La sintaxis a emplear normalmente es de este tipo:

```
var nuevoNodo = document.createElement('tagElegido');
```

Donde tagElegido representa el tipo de nodo que vamos a crear indicando su correspondiente etiqueta HTML. Por ejemplo var nuevoNodo = document.createElement('div'); crea un nodo … .

Un nodo de tipo Text se crea invocando el método createTextNode del objeto document de JavaScript. La sintaxis a emplear normalmente es de este tipo:

```
var nodoTexto = document.createTextNode('texto contenido en el nodo');
```

Donde 'texto contenido en el nodo' indica el texto que queremos insertar.

Normalmente tendremos que crear la dependencia en el árbol del DOM por el cual un nodo creado deberá definirse como hijo de un nodo ya existente. Es la forma de que un nodo creado se integre dentro de nuestra página web. La sintaxis a emplear normalmente es de este tipo:

```
nombreNodoPadre.appendChild(nombreNodoHijo);
```

Escribe este código, guárdalo como archivo html y comprueba sus resultados en tu navegador:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head> <title>Curso JavaScript aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center; }
div {border-style:solid;margin: 30px; padding:25px; display:inline-block;}
div div {background-color: yellow;}
</style>
<script type="text/javascript">
var contador = 1
function crearNodoHijo(nodoPadre) {
var nodoHijo = document.createElement("div");
var nodoTexto = document.createTextNode("Soy el nodo hijo "+contador);
nodoHijo.appendChild(nodoTexto);
nodoPadre.appendChild(nodoHijo);
contador = contador +1;
}
</script>
</head>
<body>
<div id="nodoRaiz" onclick="crearNodoHijo(this)">
Contenido inicial es este texto
</div>
</body>
</html>
```

El resultado esperado es que cada vez que se hace click dentro del div principal, se añade un nuevo div de forma que por cada pulsación van apareciendo nuevos recuadros donde se indica “Soy el nodo hijo 1”, “Soy el nodo hijo 2”, “Soy el nodo hijo 3”, etc. En este ejemplo, la pulsación en cualquier parte interior al div inicial (estemos o no dentro de un div hijo) genera el evento por el cual se crea un hijo.

MODIFICAR LA PROPIEDAD ONCLICK CON JAVASCRIPT

En este otro ejemplo vemos cómo podemos crear un div interno a aquel div en que se hace click. Para ello, hemos de establecer la propiedad onclick del nodo creado. La sintaxis a emplear para ello es esta:

```
nombreDeNodo.onclick = function() {nombreDeFuncion(param1, param2, …, paramN)};
```

nombreDeFuncion indica la función que debe ejecutarse como respuesta al evento onclick. Param1, param2, …, paramN indica los parámetros que se le pasan a la función (en caso de no existir parámetros simplemente se abrirán y cerrarán paréntesis).

Escribe este código, guárdalo como archivo html y comprueba sus resultados en tu navegador:

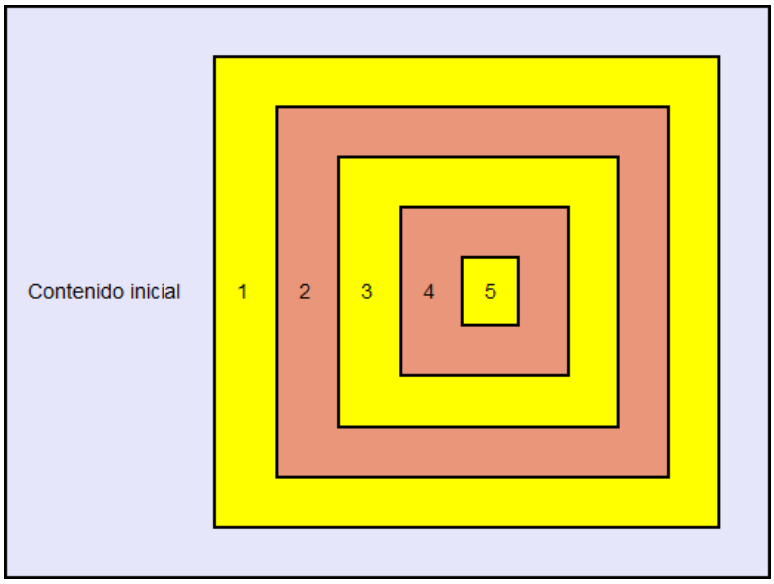
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head> <title>Curso JavaScript aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center; }
div {border-style:solid;margin: 20px; padding:15px; display:inline-block;}
</style>
<script type="text/javascript">
var contador = 1;
function crearNodoHijo(nodoPadre) {
if (nodoPadre.children.length == 0) {
            var nodoHijo = document.createElement("div");
            nodoHijo.id ='divElementID'+contador;
            nodoHijo.onclick = function() {crearNodoHijo(this)};
            nodoHijo.innerHTML = ''+contador;
            if (contador%2 == 0) {nodoHijo.style.backgroundColor = '#E9967A';}
            else {nodoHijo.style.backgroundColor = 'yellow'; }
            nodoPadre.appendChild(nodoHijo);
            contador = contador +1;
}
}
</script>
</head>
<body>
<div id="nodoRaiz" onclick="crearNodoHijo(this)" style="background-color:#E6E6FA">
Contenido inicial
</div>
</body>
</html>
```

Merece la pena detenernos en la línea nodoHijo.onclick = function() {crearNodoHijo(this)};

Para establecer el valor del atributo onclick de un nodo, hemos de definir una función para ser ejecutada. Esto se hace con la sintaxis que hemos indicado. No es válido nodoHijo.onclick = 'crearNodoHijo(this) '; porque equivale a asignar un texto, ni nodoHijo.onclick = crearNodoHijo(this); porque aquí estaríamos asignando a la propiedad el hipotético resultado (obtenido con return) de ejecutar la función crearNodoHijo.

El resultado esperado es que cada vez que hacemos click en el div más interno entro todos los existentes, se genere un nuevo div, y que los colores se vayan alternando entre color amarillo y un color salmón. El evento onclick se genera para todos los div existentes (ya que el más interno está contenido dentro de todos ellos). Es decir, suponiendo que tengamos por ejemplo tres div uno externo, otro intermedio y otro interno, al pulsar encima del div interno se generan tres eventos onclick, uno para cada div. No obstante, en nuestro ejemplo sólo generamos un nodo hijo si el nodo que recibe el click no tiene hijos. De este modo, no se generan nuevos hijos excepto para el div más interno, que es el único que no tiene hijos, mientras que los div más externos no sufren cambios.

Gráficamente, el resultado esperado es que se vayan generando sucesivos recuadros internos a medida que se pulsa en el recuadro central.



¿Podríamos determinar usando JavaScript en qué div exactamente se está haciendo click sin tener que basarnos en el número de hijos que tenga? Sí, esto está relacionado con la captura de eventos y lo estudiaremos más adelante.

EJERCICIO

Crea una página web donde inicialmente exista un div con borde solid y ancho delimitado con el texto "Curso JavaScript aprenderaprogramar.com". Debajo del div deben mostrarse dos botones o pulsadores. Un botón debe indicar "Añadir al final" y otro "Anidar". Cuando se pulse el botón "Añadir al final", debe añadirse un div al final de la página con el texto "Nodo creado 1" (o Nodo creado 2, 3, 4… según corresponda). Cuando se pulse el botón "Anidar" debe añadirse un div dentro del div inicial con el texto "Nodo creado 5" (o Nodo creado 6, 7, 8… según corresponda). La numeración será única, es decir, podremos saber en qué orden han sido creados los div estén dentro o fuera del div inicial.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01141E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT INSERTBEFORE, CLONENODE (COPIAR O DUPLICAR NODOS), REMOVECHILD, REPLACECHILD. EJEMPLOS (CU01141E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº41 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

MÁS ACCIONES SOBRE NODOS DEL DOM

JavaScript ofrece algunas posibilidades adicionales de actuar sobre nodos del DOM. En este caso vamos a ver una variante de appendChild denominada insertBefore, un método para clonar nodos denominado cloneNode, y los métodos para eliminar y reemplazar nodos removeChild y replaceChild.

INSERTBEFORE

Además de usar appendChild, que nos permite añadir un nodo como nodo hijo del nodo especificado con la sintaxis nombreNodoPadre.appendChild(nombreNodoHijo); tenemos otra opción para insertar un nodo en un documento HTML (página web): el método insertBefore. La sintaxis a emplear normalmente es de este tipo:

```
nodoQueSeráPadreDelNuevoNodo.insertBefore(nuevoNodo,
nodoAntesDelQueHaremosLaInserción);
```

Escribe el siguiente código, guárdalo con extensión html y visualiza los resultados en tu navegador. El resultado esperado es que inicialmente aparece un formulario donde se pide nombre, apellidos y correo electrónico. Se nos da opción a pulsar en un enlace para poder añadir otra persona al formulario y cuando lo hacemos nos aparecen las casillas para la persona adicional (tantas personas como queramos) debido a que estamos insertando un nuevo nodo con las nuevas casillas.

Analiza el código y trata de razonar sobre algunos detalles, por ejemplo cuál es el papel de . Si tienes dudas consulta en los foros aprenderaprogramar.com.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo curso aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {background-color:yellow; font-family: sans-serif;}
label {color: maroon; display:block; padding:5px;}
.simulaLink {color: blue; cursor:pointer;}
</style>
<script type="text/javascript">
var contador = 1;
function addPersona () {
var nodoForm = document.querySelector('form');
var nuevoNodo = document.createElement('div');
contador = contador+1;
```

```
nuevoNodo.innerHTML = '<label for="nombre'+contador+'"><span>Nombre persona '+contador+':</span> <input
id="nombre'+contador+
'" type="text" name="nombre'+contador+'" /></label>'+
'<label for="apellidos'+contador+'"><span>Apellidos persona '+contador+':</span> <input id="apellidos'+contador+
'" type="text" name="apellidos'+contador+'" /></label>'+
'<label for="email'+contador+'"><span>Correo electrónico persona '+contador+':</span> <input id="email"'+contador+
' type="text" name="email'+contador+'" /></label>'+
'<p>Si quieres apuntar simultáneamente otra persona <span class="simulaLink" onclick="addPersona()">pulsa
aquí</span></p>';
nodoForm.insertBefore(nuevoNodo, nodoForm.querySelector('#referenciaVacia1'));
}
</script>
</head>
<body><div id="cabecera"><h1>Portal web aprenderaprogramar.com</h1><h2>Didáctica y divulgación de la
programación</h2></div>
<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="http://aprenderaprogramar.com">
<p>Indica los datos de la persona que se apunta al curso:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<p>Si quieres apuntar simultáneamente otra persona <span class="simulaLink" onclick="addPersona()">pulsa
aquí</span></p>
<div id="referenciaVacia1"></div>
<label><input type="submit" value="Enviar"><input type="reset" value="Cancelar"></label>
</form></body></html>
```

DUPLICAR NODOS EN EL DOCUMENTO HTML. CLONENODE

Si estamos trabajando con un documento HTML (página web) y tratamos de reinsertar un nodo ya existente, el resultado no será que se genere una copia del nodo a insertar en la nueva posición, sino que en primer lugar se eliminará el nodo automáticamente de su posición inicial y se insertará en la nueva posición designada.

¿Cómo podemos entonces añadir exactamente el mismo nodo (una copia) en otra posición del documento HTML?

Para ello disponemos del método cloneNode. La sintaxis a emplear normalmente es de este tipo:

```
var nodoNuevoCopia = nodoACopiar.cloneNode(valorBooleano);
```

Donde valorBooleano es un parámetro que se le pasa al método que en caso de valer true supone que se clonará el nodo a copiar con todos sus descendientes (copia completa) y en caso de valer false supone que se clonará el nodo sin sus descendientes (copia simple).

El ejercicio propuesto más adelante incluye un ejemplo de uso de cloneNode.

ELIMINAR NODOS EN EL DOCUMENTO HTML CON REMOVECHILD

El método removeChild elimina un nodo y todos sus descendientes. La sintaxis para emplearlo es:

```
nodoPadre.removeChild(nodoHijo);
```

Donde nodoPadre es el nodo padre del nodo que queremos eliminar, y nodoHijo el nodo a eliminar (junto a todos sus descendientes).

Una forma habitual de acceder al nodo padre es usar la invocación nodoHijo.parentNode. Por ejemplo:

nodoAEliminar.parentNode.removeChild(nodoAEliminar);

Nos permite acceder al nodo padre del que queremos eliminar e invocar sobre él removeChild.

Crea un documento html con el siguiente código y comprueba cómo se pueden añadir nodos con appendChild y eliminar nodos con removeChild pulsando un botón “añadir” o “eliminar”.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head> <title>Curso JavaScript aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center; }
#nodoRaiz {border-style:solid;margin: 30px; padding:25px; display:inline-block;}
.nodoHijo {background-color: yellow; border-style:solid;margin: 20px; padding:15px; display:inline-block;}
#contenedor {width:405px;margin: 0 auto;}
#adelante, #atras {padding:15px; width: 130px; float: left; margin-left:20px;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#adelante:hover, #atras:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
var contador = 1
function crearNodoHijo(nodoPadre) {
        var nodoHijo = document.createElement("div");
        var nodoTexto = document.createTextNode("Soy el nodo hijo "+contador);
        nodoHijo.className = 'nodoHijo'; nodoHijo.appendChild(nodoTexto); nodoPadre.appendChild(nodoHijo);
        contador = contador +1;
}
function removeNodo() {
        var nodosEliminar = document.querySelectorAll('.nodoHijo');
        if (nodosEliminar.length>0) { //Si hay nodos que se puedan eliminar
        var nodoAEliminar = nodosEliminar[nodosEliminar.length-1]; //Acceso al último nodo hijo
        nodoAEliminar.parentNode.removeChild(nodoAEliminar);
        contador = contador -1;  }
}
</script>
</head>
<body>
<div id="nodoRaiz" onclick="crearNodoHijo(this)">Contenido inicial es este texto</div>
<div id="contenedor">
<div id ="atras" onclick="crearNodoHijo(document.querySelector('#nodoRaiz'))"> Añadir </div>
<div id="adelante" onclick="removeNodo()" > Eliminar </div>
</div></body></html>
```

REEMPLAZAR UN NODO CON REPLACECHILD

El método replaceChild permite eliminar un nodo y reemplazarlo por otro. La sintaxis para emplearlo es:

```
nodoPadre.replaceChild(nuevoNodo, nodoQueSeráReemplazado);
```

Donde nuevoNodo es el nodo que vamos a insertar para reemplazar al nodo que hemos denominado nodoQueSeráReemplazado.

Si nuevoNodo ya existe en el DOM, primero será eliminado del lugar en que ya existía y luego será insertado en el lugar indicado. El nodo reemplazado es eliminado en todo caso.

Crea un documento html con el siguiente código y comprueba cómo se puede reemplazar el primer nodo de una serie por el último usando replaceChild.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head> <title>Curso JavaScript aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center; }
#nodoRaiz {border-style:solid;margin: 30px; padding:25px; display:inline-block;}
.nodoHijo {background-color: yellow; border-style:solid;margin: 20px; padding:15px; display:inline-block;}
#contenedor {width:405px;margin: 0 auto;}
#adelante, #atras {padding:15px; width: 130px; float: left; margin-left:20px;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#adelante:hover, #atras:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
var contador = 1
function crearNodoHijo(nodoPadre) {
        var nodoHijo = document.createElement("div");
        var nodoTexto = document.createTextNode("Soy el nodo hijo "+contador);
        nodoHijo.className = 'nodoHijo';
        nodoHijo.appendChild(nodoTexto);
        nodoPadre.appendChild(nodoHijo);
        contador = contador +1;
}
function cambiarNodo() {
        var nodosHijos = document.querySelectorAll('.nodoHijo');
        var nodoPadre = document.querySelector('#nodoRaiz');
        if (nodosHijos.length>1) { //Si hay nodos que se puedan cambiar
        var tmpNodo = nodosHijos[0].cloneNode(true);
        nodoPadre.replaceChild(nodosHijos[nodosHijos.length-1], nodosHijos[0]);
        nodoPadre.appendChild(tmpNodo);
        }
}
</script>
</head>
```

```
<body>
<div id="nodoRaiz">
Contenido inicial es este texto
</div>
<div id="contenedor">
<div id ="atras" onclick="crearNodoHijo(document.querySelector('#nodoRaiz'))"> Añadir </div>
<div id="adelante" onclick="cambiarNodo()" > Cambiar </div>
</div>
</body>
</html>
```

Fíjate cómo el primer nodo de la serie de hijos lo clonamos para después de ser reemplazado por el último, poder insertar el clon en la posición final.

EJERCICIO

El siguiente código pretendía duplicar un nodo cada vez que se pulsara sobre él, de modo que inicialmente apareciera en pantalla el mensaje “Pulsa aquí para duplicar este nodo” y que cada vez que se pulsara sobre él, se duplicara (tantas veces como veces se pulsara). El problema es que no funciona, es decir, no obtenemos ningún resultado. Analiza el código y responde estas cuestiones:

1) ¿Por qué no funciona?

2) Realiza los cambios en el código que nos permitan realizar lo que teníamos previsto creando un nuevo nodo del mismo tipo y con el mismo texto que el deseado y añadiéndolo como último hijo del nodo body.

3) Realiza los cambios en el código que nos permitan realizar lo que teníamos previsto clonando el nodo con el método cloneNode y añadiendo el nodo clonado como último hijo del nodo body.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head> <title>Curso JavaScript aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
body {font-family: sans-serif; text-align:center; }
div {border-style:solid;margin: 30px; padding:25px; display:inline-block;}
div div {background-color: yellow;}
</style>
<script type="text/javascript">
function crearNodoHijo(nodoPadre) {
var nodoHijo = nodoPadre;
document.body.appendChild(nodoHijo);
}
</script>
</head>
<body>
<div id="nodoRaiz" onclick="crearNodoHijo(this)">Pulsa aquí para duplicar este nodo</div>
</body></html>
```

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01142E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FUNCIONES JAVASCRIPT PARA CADENAS DE TEXTO: TOUPPERCASE, TOLOWERCASE, CHARAT, SUBSTRING, SLICE, INDEXOF, SPLIT, REPLACE, REPLACEALL. (CU01142E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº42 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

FUNCIONES JAVASCRIPT PARA CADENAS DE TEXTO

JavaScript ofrece numerosas funciones predefinidas que facilitan el trabajo con cadenas de texto. Entre las posibilidades que ofrecen estas funciones tenemos el extraer un carácter, extraer un fragmento de cadena, separar una cadena en múltiples cadenas indicando un separador, etc.

En las siguientes tablas resumimos las principales funciones disponibles. Las hemos agrupado según el uso más habitual que se hace de ellas en funciones que se usan habitualmente para modificar las cadenas, funciones que se usan habitualmente para extraer subcadenas o caracteres y funciones que se usan para determinar el índice de posición de un carácter bajo ciertas condiciones.

En algunos casos pueden presentarse diferencias entre navegadores cuando se hacen invocaciones anormales o cuando se usan algunas funciones como slice con números negativos.

FUNCIONES MODIFICADORAS

Función	Tarea y comentarios	Ejemplo
toUpperCase()	Transforma la cadena a mayúsculas	textoUsuario.toUpperCase()
toLowerCase()	Transforma la cadena a minúsculas	textoUsuario.toLowerCase()
replace('carácterA', 'caracterB')	Reemplaza la primera aparición de carácterA por carácterB en la cadena.	textoUsuario.replace('e', 'E');
replace (/carácterA/g, 'carácterB')	Reemplaza todas las apariciones de carácterA por carácterB en la cadena. Tener en cuenta que el primer parámetro no va entrecomillado.	textoUsuario.replace (/e/g, 'E') Reemplaza todas las e minúsculas por E mayúsculas. Si queremos reemplazar los espacios escribiremos: textoUsuario.replace (/ /g, 'E') dejando un espacio entre las barras. Es la forma de expresar el replaceAll de otros lenguajes.
replace (/cadenaA/g, 'cadenaB')	Reemplaza todas las apariciones de la subcadena cadenaA por cadenaB. Tener en cuenta que el primer parámetro no va entrecomillado.	textoUsuario.replace (/plo/g, 'XX') Reemplaza todas las apariciones de plo sustituyéndolas por XX

LA AUSENCIA DE REPLACEALL

Hemos indicado una sintaxis un tanto extraña para reemplazar todas las apariciones de un carácter en una cadena: replace (/carácterA/g, 'carácterB') o de una subcadena por otra, con la sintaxis de tipo replace (/cadenaA/g, 'cadenaB').

Lo que podemos intuir como "extraño" es que hemos de incluir el primer parámetro entre las barras y que no aparecen comillas en este primer parámetro. El motivo para ello es la ausencia de una instrucción replaceAll en JavaScript que nos obliga a buscar alguna alternativa para conseguir el objetivo de reemplazar todas las apariciones de un carácter o subcadena en una cadena. La alternativa que hemos señalado aquí se basa en expresiones regulares: una forma de definir "patrones" de cadenas a ser reconocidas por JavaScript. Hablaremos de expresiones regulares más adelante, por lo que no vamos a entrar ahora en más detalles.

FUNCIONES PARA EXTRAER SUBCADENAS O CARACTERES

Función	Tarea y comentarios	Ejemplo
substring(firstIn, lastOut)	Devuelve la subcadena extraída entre los índices firstIn y lastOut-1. Es decir, el carácter en la posición firstIn se incluye y el carácter en la posición lastOut se excluye. Recordar que el primer índice es cero y el último la longitud menos uno.	textoUsuario.substring(2, 6) Para la cadena **desarrolloWeb** nos devolvería "sarr", donde d es el carácter en posición cero, s el que está en posición 2 y o el que está en posición 6 (que queda excluido).
substring (firstIn)	Misma función omitiendo el segundo parámetro. Devuelve la subcadena desde el índice firstIn hasta el final de la cadena	textoUsuario.substring(5)
slice (firstIn, lastOut)	Análogo a substring pero permite que los índices sean negativos. En este caso, se toman desde el final de la cadena hacia el principio, excluyendo el último carácter.	textoUsuario.slice(-5, -1) Si la cadena es "aprenderaprogramar" nos devuelve rama, siendo la r la quinta letra empezando por el final. Para extraer hasta el final usar (-5, 0) ó (-5).
slice (firstIn)	Análogo a substring pero permite definir que se tome una porción final de cadena, desde el índice señalado hasta el final.	textoUsuario.slice(-5); Devuelve las cinco últimas letras.
split('caracterSepara')	Devuelve un array con las subcadenas resultantes de dividir la cadena original en subcadenas delimitadas por el carácter separador especificado (que queda excluido). Si se indican unas comillas vacías se extraen todos los caracteres a un array.	textoUsuario.split(' ') División por espacios en blanco textoUsuario.split(''); Extracción de todos los caracteres
nombreCadena[indice]	Devuelve el carácter en la posición indicada por índice.	textoUsuario[3] Devuelve el carácter en índice 3 textoUsuario[textoUsuario.length-1] devuelve el último carácter

FUNCIONES PARA RECUPERAR ÍNDICES DE POSICIONES

Función	Tarea y comentarios	Ejemplo
charAt(indicePosicion)	Devuelve la letra situada en la posición indicePosicion. Tener en cuenta que el primer índice es cero y el último la longitud menos uno.	var primeraLetra = textoUsuario.charAt(0)
indexOf ('cadena')	Devuelve el índice de la primera aparición de la cadena especificada empezando a buscar desde la izquierda. Si no existe el carácter se devuelve -1. Recordar que la primera letra lleva índice cero.	textoUsuario.indexOf('p')
indexOf ('cadena', indiceInicial)	Devuelve el índice de la primera aparición de la cadena especificada empezando la búsqueda desde el índice inicial especificado (incluido).	textoUsuario.indexOf('p', 5)
lastIndexOf ('cadena')	Devuelve el índice de la última aparición de la cadena especificada en un string. Si no existe la cadena se devuelve -1. Recordar que la primera letra lleva índice cero.	textoUsuario.lastIndexOf('p')

EJEMPLO

Escribe este código y guárdalo con extensión HTML. Comprueba los resultados de la ejecución en tu navegador.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> body {font-family: sans-serif; margin:25px; text-align:center;}
#pulsador {padding:15px; width: auto; display: inline-block; margin: 25px; cursor: pointer;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#pulsador:hover, #marcaOperadores:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
function ejemploCadenasDeTexto() {
var textoUsuario = prompt ('Introduzca una frase');
var mayusculas = textoUsuario.toUpperCase(); var minusculas = textoUsuario.toLowerCase();
var primeraLetra = textoUsuario.charAt(0); var ultimaLetra = textoUsuario.charAt(textoUsuario.length-1);
var subcadena26 = textoUsuario.substring(2, 6); var subcadena26sli = textoUsuario.slice(2,6);
var subcadena5aFinal = textoUsuario.substring(5); var subcadena5aFinalsli = textoUsuario.slice(5);
var tresCharDesdeFinal = textoUsuario.slice(-5, -2); var cincoUltimasLetras = textoUsuario.slice(-5);
var indicePrimeraP = textoUsuario.indexOf('p'); var indicePrimeraPDesdeIndice5 = textoUsuario.indexOf('p', 5);
var ultimaP = textoUsuario.lastIndexOf('p'); var subcadenasPorEspacios = textoUsuario.split(' ');
var cadenaCambiaPrimeraePorE = textoUsuario.replace('e', 'E');
var cadenaCambiaTodasePorE = textoUsuario.replace (/e/g, 'E');
var reemplazoSubcadena = textoUsuario.replace (/plo/g, 'XX');
var caracterIndice3 = textoUsuario[3];
var msg = 'Entrada: '+textoUsuario + '\n';
msg = msg + 'La entrada en mayúsculas es: ' + mayusculas +'\nLa entrada en minúsculas es: ' + minusculas + '\n';
msg = msg + 'Primera letra es: ' + primeraLetra + ' y última letra es: '+ultimaLetra + '\n';
msg = msg + 'Subcadena entre tercera letra y sexta letra es: ' + subcadena26 + '. Si se extrae con slice: '+subcadena26sli
+'\n';
msg = msg + 'Subcadena desde el caracter indice 5 hasta el final es: ' + subcadena5aFinal+ '. Si se extrae con slice:
'+subcadena5aFinalsli+'\n';
msg = msg + 'Uso de slice negativo para extraer del quinto por atrás al antepenúltimo caracter: ' + tresCharDesdeFinal+ '\n';
msg = msg + 'Cinco últimos caracteres: '+cincoUltimasLetras + '\n';
msg = msg + 'Indice posición de la primera p en la cadena: ' + indicePrimeraP + ' y de la última p es: ' +ultimaP +'\n';
msg = msg + 'Indice primera p desde la 5a posición es: ' + indicePrimeraPDesdeIndice5 +'\n';
msg = msg + 'División de subcadenas por espacios: ';
for (var i=0; i<subcadenasPorEspacios.length; i++) { msg = msg + 'Subcadena ' + (i+1) + ':\<\<'+subcadenasPorEspacios[i] +
'\>\> '; }
msg = msg + '\n';
msg = msg + 'Cadena con primera e reemplazada por E: ' + cadenaCambiaPrimeraePorE + '\n';
msg = msg + 'Cadena con e reemplazada por E todas: ' + cadenaCambiaTodasePorE + '\n';
msg = msg + 'Cadena reemplazando plo por XX: ' + reemplazoSubcadena + '\n';
msg = msg + 'El carácter con índice 3 es: ' + caracterIndice3;
alert (msg);
}
</script>
</head>
<body>
<div id="cabecera">
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplo funciones para texto JavaScript</h3>
</div>
<div id ="pulsador" onclick="ejemploCadenasDeTexto()"> Probar </div>
</body>
</html>
```

El resultado esperado si introducimos la cadena "Ejemplo aplicación JavaScript aprenderaprogramar.com diploma" es el siguiente:

Entrada: Ejemplo aplicación JavaScript aprenderaprogramar.com diploma
La entrada en mayúsculas es: EJEMPLO APLICACIÓN JAVASCRIPT APRENDERAPROGRAMAR.COM DIPLOMA
La entrada en minúsculas es: ejemplo aplicación javascript aprenderaprogramar.com diploma
Primera letra es: E y última letra es: a
Subcadena entre tercera letra y sexta letra es: empl. Si se extrae con slice: empl
Subcadena desde el caracter indice 5 hasta el final es: lo aplicación JavaScript aprenderaprogramar.com diploma. Si se extrae con slice: lo aplicación JavaScript aprenderaprogramar.com diploma
Uso de slice negativo para extraer del quinto por atrás al antepenúltimo caracter: plo
Cinco últimos caracteres: ploma
Indice posición de la primera p en la cadena: 4 y de la última p es: 55
Indice primera p desde la 5a posición es: 9
División de subcadenas por espacios: Subcadena 1:<<Ejemplo>> Subcadena 2:<<aplicación>> Subcadena 3:<<JavaScript>> Subcadena 4:<<aprenderaprogramar.com>> Subcadena 5:<<diploma>>
Cadena con primera e reemplazada por E: EjEmplo aplicación JavaScript aprenderaprogramar.com diploma
Cadena con e reemplazada por E todas: EjEmplo aplicación JavaScript aprEndEraprogramar.com diploma
Cadena reemplazando plo por XX: EjemXX aplicación JavaScript aprenderaprogramar.com diXXma
El carácter con índice 3 es: m

EJERCICIO

Crea un documento HTML (página web) donde exista un formulario que se envíe por el método GET. Se pedirá al usuario que introduzca nombre, apellidos y correo electrónico. Define dentro de la etiqueta form que cuando se produzca el evento onsubmit (pulsación del botón de envío del formulario) se ejecute una función a la que denominaremos validar que no recibe parámetros.

La función validar debe realizar estas tareas y comprobaciones:

a) Comprobar que el nombre contiene al menos tres letras. Si no es así, debe aparecer un mensaje por pantalla indicando que el texto no cumple tener al menos tres letras. Por ejemplo si se trata de enviar Ka como nombre debe aparecer el mensaje: “El nombre no cumple tener al menos tres letras”.

b) Comprobar que el correo electrónico contiene el carácter @ (arroba) y el carácter . (punto). De no ser así, deberá aparecer un mensaje indicando que al correo electrónico le falta uno o ambos caracteres. Por ejemplo si se trata de enviar pacogmail.com deberá aparecer el mensaje: “Falta el símbolo @ en el correo electrónico”.

c) Antes de enviarse los datos del formulario a la página de destino, todas las letras del correo electrónico deben transformarse a minúsculas. Por ejemplo si se ha escrito PACO@GMAIL.COM debe enviarse paco@gmail.com

d) Antes de enviarse los datos del formulario a la página de destino, si el correo electrónico contiene la subcadena " at " debe reemplazarse por el símbolo @. Por ejemplo si se ha escrito paco at gmail.com debe enviarse paco@gmail.com

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01143E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CREAR OBJETOS JAVASCRIPT CON THIS Y NEW. EJEMPLOS PARA ENTENDER QUÉ SON LOS OBJETOS Y PARA QUÉ SIRVEN. (CU01143E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº43 del Tutorial básico "JavaScript desde cero".

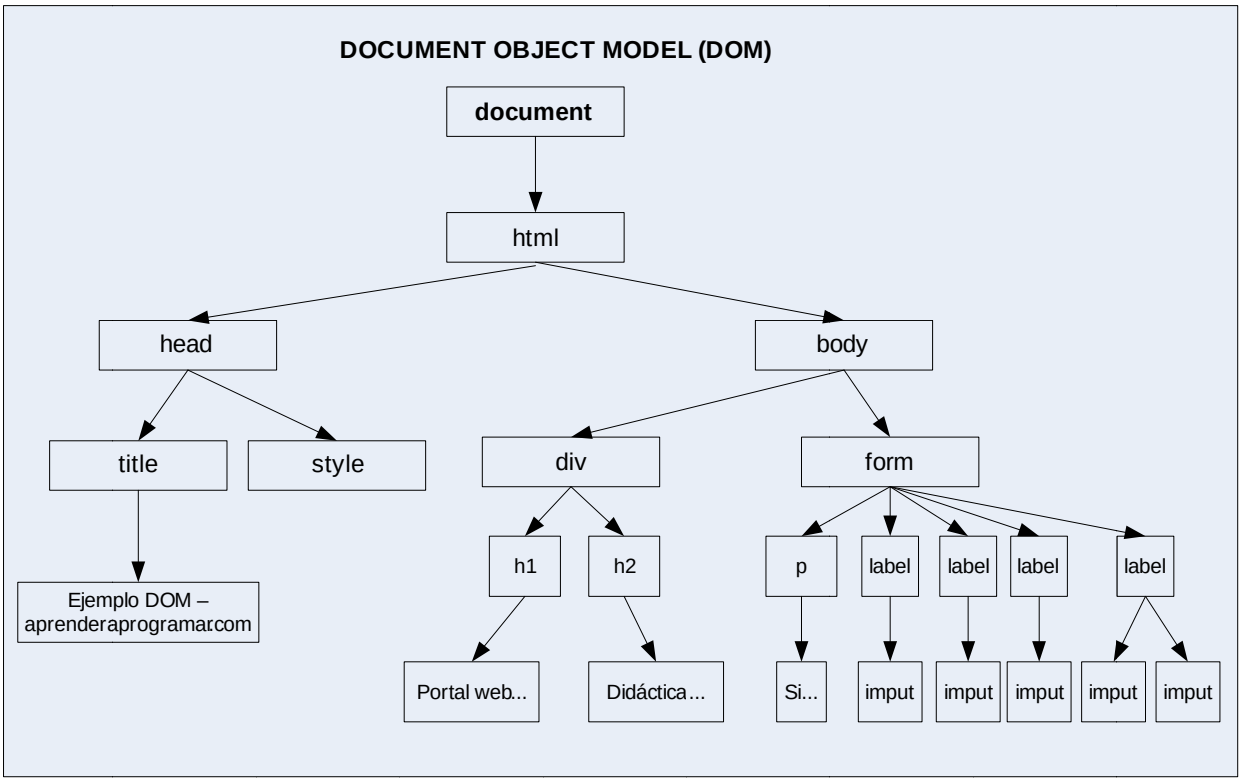
Autor: César Krall

OBJETOS JAVASCRIPT

En muchas ocasiones nos hemos referido a "el objeto document de JavaScript" o hemos dicho que un NodeList es un tipo de objeto JavaScript. Nos interesa ahora centrarnos en estudiar con mayor detenimiento el concepto de objeto JavaScript.

Un objeto es un tipo avanzado de datos que admite atributos (propiedades) y funciones (métodos). Tendremos que ir paso a paso para comprender esta definición.

Si recordamos cómo considera JavaScript que se estructura una página web a través del DOM, la representación que hacíamos era de este tipo:



Podemos señalar que el propio DOM lleva la palabra objeto contenida en sí mismo: Document **Object** Model. Esto ya nos indica que para JavaScript todos los elementos presentes en el DOM son objetos. Por ejemplo el elemento document es un objeto que representa la página web en su conjunto (el documento HTML) y a su vez consta de numerosos nodos que también son objetos. JavaScript utiliza los objetos como estructura de datos fundamental: prácticamente todo (casi todo) en JavaScript son objetos.

Un objeto puede tener valores asociados a los que denominamos propiedades o atributos. Por ejemplo en un nodo al que hayamos denominado nodoDiv el atributo id nos indica el valor de id en el documento html para la etiqueta asociada.

En el código HTML tendremos por ejemplo:

Y rescatamos el nodo así: var nodo = document.getElementById('cabecera');

Y luego nos referimos al atributo id del objeto nodo como nodo.id.

Un atributo podría ser una entidad simple, por ejemplo un atributo id con valor 'cabecera', pero también podría ser una entidad compleja, es decir, otro objeto. Por ejemplo la propiedad doctype del objeto document (que invocamos como document.doctype) nos devuelve otro objeto.

Un objeto puede tener métodos (funciones) asociados, que permiten realizar operaciones invocando dicho objeto. Por ejemplo getElementById es un método disponible para el objeto document.

Escribe este código, guárdalo como documento HTML y trata de razonar sobre lo que se visualiza.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() {
var elDocType = document.doctype;
var resultadoEjecutarMetodo = document.getElementById('cabecera');
var msg = 'document es un objeto. Una de sus propiedades es doctype, puede valer por ejemplo: ' + elDocType + ' que es
otro objeto, ';
msg = msg +'es decir, una propiedad puede ser un objeto. La propiedad name del objeto doctype devuelve: ' +
elDocType.name;
msg = msg + ' y la propiedad publicId devuelve ' + elDocType.publicId + ' \n\n';
msg = msg + 'document como objeto tiene métodos, por ejemplo getElementById nos devuelve ';
msg = msg + 'el elemento (objeto) con id indicado, que a su vez tiene atributos y métodos. Por ';
msg = msg + 'ejemplo el div con id cabecera objeto ' + resultadoEjecutarMetodo + ' tiene atributo nodeName:
'+resultadoEjecutarMetodo.nodeName + '\n\n';
alert (msg);
}
</script>
</head>
<body>
<div id="cabecera">
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplo funciones JavaScript</h3>
</div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Fijate cómo los objetos cuando se tratan de mostrar por pantalla (en modo texto) devuelven un corchete seguido de la palabra object y el tipo de objeto. Por ejemplo [object DocumentType] ó [object HTMLDivElement]. En cambio las propiedades o atributos únicamente devuelven un texto, sin los corchetes ni la palabra object.

Nosotros podremos definir "moldes" de objetos con sus propiedades y con sus métodos, crear uno o varios objetos del tipo definido y hacerlos trabajar dentro de nuestro código para facilitar la tarea de la programación.

Pensemos en la ventaja que esto nos supone: consideremos que tenemos un array como una serie de números y queremos conocer su suma. Podríamos escribir algo como:

```
var suma = 0;
var numero = [3, 2, 9];
for (var i=0; i<numero.length; i++){
suma = suma + numero[i];
}
alert ('La suma de los números es: ' + suma);
```

Pero esto nos obliga a usar un bucle cada vez que queremos conocer la suma de los números contenidos en el array.

¿Y si definiéramos el array como un objeto al que le añadimos un método por el cual sea capaz de sumar automáticamente el valor de sus números y devolvérnoslo?

Esto sería bastante útil. El interés entonces de usar objetos está en que nos permiten combinar datos y funciones en una misma entidad. El objeto almacenará unos datos (por ejemplo los valores numéricos contenidos en un array) y tendrá definidos unos métodos o funciones internas que permiten realizar operaciones o procesos (por ejemplo obtener la suma de los valores numéricos).

DEFINIR TIPOS DE OBJETOS SIMPLES CON JAVASCRIPT

Empecemos por definir un objeto muy simple, algo que representará una cuenta bancaria. Nuestro molde, patrón o definición de tipo se llamará cuentaBancaria y cada vez que queramos crear una cuenta bancaria escribiremos algo así como *new cuentaBancaria*.

Hay varias formas de definir tipos de objetos y de crear objetos. Empezaremos usando esta sintaxis para definir un tipo de objeto:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = function () { ... código .... }
this.método2 = function (param1, param2, ..., paramN) { ... código ...}
}
```

Donde par1, par2, ... , parN son los posibles parámetros necesarios para crear un objeto.

nombrePropiedad1, nombrePropiedad2, etc. son los atributos o propiedades (información) que porta el objeto.

método1, método2, etc. son los métodos o funciones que pueden ser invocadas a través del objeto. Estos métodos pueden no requerir parámetros o sí requerir parámetros. El método1 sería un ejemplo de método que no requiere parámetros y el método2 un ejemplo de método que sí requiere parámetros.

Para crear un objeto del tipo definido escribiremos:

```
var nombreInstanciaObjetoCreada = new nombreTipoObjeto (param1, param2, ..., paramN);
```

Para invocar una propiedad escribiremos:

```
nombreDelObjeto.nombreDeLaPropiedad
```

Para invocar un método escribiremos:

```
nombreDelObjeto.nombreDelMétodo (param1, param2, ..., paramN)
```

Donde param1, param2, paramN son los parámetros necesarios para que el método (función) se pueda ejecutar, si es que hay parámetros. Si no los hubiera, simplemente escribiremos los paréntesis de apertura y cierre.

Dado que para invocar propiedades y métodos se usa el nombre del objeto seguido de un punto y el nombre de la propiedad o método, se dice que las propiedades y métodos se invocan usando el operador punto.

Para comprender todo lo anterior escribe este código y guárdalo con extensión html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function CuentaBancaria () {
this.nombreTitular = "Barack";
this.apellidosTitular = "Obama";
this.saldo = 600000;
this.mostrarDatos = function () {
        var msg = 'Los datos de la cuenta son Nombre: ' + this.nombreTitular;
        msg = msg + '; Apellidos: ' + this.apellidosTitular + '; Saldo: ' + this.saldo;
        alert(msg);
        }
}
```

```
function ejemploCreaObjetos() {
var cuenta1 = new CuentaBancaria();
msg = 'El saldo en la cuenta bancaria de Obama es: ' + cuenta1.saldo + ' dólares';
alert (msg);
cuenta1.mostrarDatos();
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo funciones JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploCreaObjetos()"> Probar </div>
</body>
</html>
```

Cuando pulsamos sobre el texto “Probar” se crea el objeto cuenta1 mediante la invocación de new cuentaBancaria(). Al crearse el objeto, se establecen sus propiedades, en este ejemplo nombreTitular es una propiedad y su contenido es “Barack”. También se establecen métodos, en este ejemplo el método mostrarDatos que permite visualizar la información que contiene el objeto.

Una vez creado el objeto ya se pueden invocar sus propiedades y sus métodos con la sintaxis de punto.

Pero este ejemplo es poco útil, porque cada vez que creáramos un objeto cuentaBancaria, éste objeto siempre tendría el mismo titular y el mismo saldo.

Normalmente nos interesará crear objetos que respeten el patrón o molde (el tipo definido) pero donde cada objeto porte unos datos diferentes. Para ello lo más habitual será pasar esos datos como parámetros cuando se crea el objeto. Escribe el código de este ejemplo y pruébalo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function CuentaBancaria (datoTitular, datoApellidos, datoSaldo) {
this.nombreTitular = datoTitular;
this.apellidosTitular = datoApellidos;
this.saldo = datoSaldo;
this.mostrarDatos = function () {
        var msg = 'Los datos de la cuenta son Nombre: ' + this.nombreTitular;
        msg = msg + '; Apellidos: ' + this.apellidosTitular + '; Saldo: ' + this.saldo;
        alert(msg);
        }
}
function ejemploCreaObjetos() {
var cuenta1 = new CuentaBancaria('Barack', 'Obama', 600000);
cuenta1.mostrarDatos();
var cuenta2 = new CuentaBancaria('Vladimir', 'Putin', 900000);
cuenta2.mostrarDatos();
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo funciones JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploCreaObjetos()"> Probar </div>
</body></html>
```

Resulta un tanto paradójico que para crear un objeto se use function. Y además para crear los métodos dentro del objeto, también se usa function. Hablaremos de ello más adelante.

EJERCICIO

Crea un documento HTML (página web) donde exista un botón "Crear cuenta bancaria". Cuando el usuario pulse sobre el botón debe:

a) Pedirse al usuario un nombre de titular, apellidos de titular y saldo de la cuenta.

b) Crear un nuevo objeto cuentaBancaria (similar al que hemos creado en los ejemplos) que se inicializará con los datos facilitados por el usuario.

c) Mostrar un mensaje informando de que se ha creado la nueva cuenta bancaria y de los datos asociados a la cuenta bancaria creada.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01144E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CLASS JAVASCRIPT ¿CLASES? OBJETOS PREDEFINIDOS. WINDOW, OBJETO GLOBAL. NUMBER, MATH, DATE, REGEXP, ERROR. (CU01144E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº44 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CLASS JAVASCRIPT

Las personas que están aprendiendo JavaScript y han programado previamente en otros lenguajes "tradicionales" como Java, C++, Visual Basic, etc. usando programación orientada a objetos, buscan con frecuencia las clases (class) en JavaScript. Pero tal y como indica la especificación oficial, JavaScript no usa clases como lo hacen C++, SmallItalk o Java, aunque sí permita la creación de objetos.

Al igual que muchas personas buscan class en JavaScript, también buscan que el comportamiento de JavaScript se asemeje al de otros lenguajes, y la verdad es que JavaScript tiene en gran medida un comportamiento distinto al de lenguajes tradicionales populares como Java. El uso de términos comunes entre JavaScript y otros lenguajes, y el intento por programar en JavaScript como se haría en otros lenguajes, ha provocado que exista cierta confusión terminológica y con frecuencia errores de diseño o de concepto a la hora de concebir la programación con JavaScript. Por tanto si has programado previamente con lenguajes como C++ ó Java, te daremos la recomendación de que no supongas que JavaScript se va a comportar igual que estos lenguajes.

JavaScript se dice que es un lenguaje orientado a objetos que sigue un paradigma diferente de otros lenguajes como Java, al paradigma de JavaScript se le llama programación basada en prototipos o programación basada en instancias. En la programación basada en prototipos no existen las clases tal y como existen en otros lenguajes, ni existe la herencia tal y como existe en otros lenguajes, aunque sí podemos encontrar ciertas similitudes.

Algunos expertos consideran que JavaScript es el lenguaje más incomprendido entre todos los existentes, porque pocos programadores conocen bien su filosofía y potencialidad. Nosotros a lo largo del curso estamos tratando de comprender JavaScript, y seguiremos en ello a lo largo de los próximos apartados.

DEFINICIÓN DE OBJETO

Ya hemos hablado sobre objetos y tenemos una idea bastante clara de lo que son. Vamos ahora a dar una definición un poco más académica:

<<Un objeto es una estructura de datos que posee un nombre y que está formado por contenedores de información que pueden albergar otros objetos, valores primitivos o funciones. Un objeto pertenece al tipo de dato Object de JavaScript.>>

A los objetos y valores primitivos que alberga un objeto los solemos llamar propiedades, mientras que a las funciones que alberga un objeto las llamamos métodos, de ahí que muchas veces se diga que un objeto es una colección de propiedades y métodos.

Casi todo en JavaScript son objetos. Pero no todo. No son objetos los datos que son tipos primitivos: String, Number, Boolean, Null, Undefined. No obstante, existen tipos objeto equivalentes a los tipos primitivos. Por ejemplo, podemos crear un String como tipo primitivo o como objeto usando:

var cadena = 'aprenderaprogramar.com'; // Tipo primitivo

var cadenaOb = new String ('aprenderaprogramar.com'); // Objeto

Un tipo primitivo se considera un tipo de dato simple: contiene una información simple. En cambio un objeto es un tipo de datos complejo, que puede albergar gran cantidad de información de distinta naturaleza.

OBJETOS EN EL ENTORNO DE JAVASCRIPT Y OBJETOS DE USUARIO

Hemos visto cómo crear un objeto donde nosotros definimos sus propiedades y métodos. Pero existen objetos que son predefinidos por JavaScript y por tanto el navegador los reconoce sin necesidad de que nosotros los creemos.

Objeto predefinido	Utilidad
Objeto global	Objeto único y global, existente antes de que comience la ejecución de código. window suele considerarse que es el objeto global.
Object	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con objetos
Function	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con funciones
Array	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con arrays
String	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con strings
Boolean	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con booleanos
Number	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con números
Math	Facilita el uso de funciones y posibilidades matemáticas
Date	Facilita el trabajo con fechas
RegExp	Permite trabajar con expresiones regulares para reconocer fragmentos de cadenas o patrones presentes en cadenas
JSON	Permite trabajar con el formato de datos JSON
Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError	Objetos que permiten controlar y obtener información sobre errores

Escribe por ejemplo alert ('El valor de number max value es ' + Number.MAX_VALUE);

Comprobarás que te devuelve un resultado del tipo "El valor de number max value es 1.7976931348623157e+308". ¿Por qué podemos invocar Number.MAX_VALUE si no hemos definido el objeto Number ni sus propiedades y métodos? Porque este objeto, al igual que otros, son objetos predefinidos de JavaScript.

Los objetos predefinidos de JavaScript nos resultarán muy útiles para distintos cometidos. Por ejemplo el objeto Math nos permite realizar cálculos matemáticos u obtener números pseudoaleatorios. Prueba por ejemplo a hacer que se muestren varios números aleatorios con alert(Math.random());

Todo objeto JavaScript (incluido las funciones) se considera que son instancias del objeto Object, y debido a ello heredan propiedades y métodos de Object. Si no tienes claro qué significa que exista herencia no te preocupes pues lo iremos viendo a lo largo del curso. Te recomendamos que leas esto: herencia en POO, sólo para tener una idea de lo que significa herencia.

window, el objeto global, tiene sus propiedades y métodos. La función que venimos usando para mostrar mensajes por pantalla, alert, es un método de window. Cuando escribimos alert('Hola'); en realidad estamos invocando window.alert('Hola');

Es decir, alert y window.alert son lo mismo para el navegador. El hecho de que no sea necesario escribir window cuando escribimos alert obedece a que si se invoca una función que no ha sido definida de otra manera, se considera que es un método del objeto global.

Prueba este código y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() {
var texto = 'Aprende a programar';
window.alert('alert es un método del objeto global window');
window.alert(texto);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

JavaScript permite definir tipos de objetos y crear instancias de objetos de diferentes maneras y esto ha creado algo de confusión entre los programadores. Otros lenguajes tienen una sola manera de hacer estas cosas, en cambio JavaScript tiene varias. Esto resulta molesto para muchos programadores, sobre todo cuando tienen que revisar código creado por otras personas. La ventaja de esta existencia de múltiples formas sintácticas para hacer lo mismo es que cada programador puede elegir la que le resulte más cómoda o conveniente, y el inconveniente es la falta de uniformidad en el código desarrollado por distintos programadores y la necesidad de conocer esas múltiples formas para poder revisar código no creado por uno mismo.

Ya conocemos una de las formas de definir un tipo de objeto:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = function () { ... código .... }
this.método2 = function (param1, param2, ..., paramN) { ... código ...}
}
```

Como alternativa tenemos esta otra forma:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = nombreFuncion1;
this.método2 = nombreFuncion2;
}
function nombreFuncion1 (par1, par2, ..., parN) { ... código ... }
function nombreFuncion2 (par1, par2, ..., parN) { ... código ... }
```

En esta forma de definición lo que antes eran funciones anónimas internas, ahora se han convertido en funciones con nombre externas.

Fíjate que this.método1 = nombreFuncion1; únicamente define el nombre de la función que se ejecutará cuando se llame al método. En el nombre de la función no se incluyen paréntesis, únicamente el nombre.

La función externa reconoce a que se invoca una propiedad del objeto si se incluye this.nombrePropiedad dentro de su código. Igualmente para los métodos.

Prueba este código y comprueba sus resultados (que deben ser 14, 14 y 10).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function objetoSerie(unArray) {
        this.nombre = 'Nombre del objeto';
        this.contenidoArray = unArray;
        this.suma = obtenerSuma;
}
```

```
function obtenerSuma() {
                        var suma=0;
                        for (var i=0; i<this.contenidoArray.length; i++){ suma = suma + this.contenidoArray[i];   }
                        return suma;
}

function ejemploObjetos() {
var suma = 0; var serie = [3, 2, 9];
for (var i=0; i<serie.length; i++){ suma = suma + serie[i]; }
alert ('La suma de los números visto como array simple es: ' + suma);
serieOb = new objetoSerie(serie);
alert ('La suma de los números visto como objeto es: ' + serieOb.suma());
otroObjeto = new objetoSerie([2,1,3,4]); alert ('La suma para otro objeto es: ' + otroObjeto.suma());
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El problema de usar este método basado en funciones auxiliares definidas externamente es que dichas funciones son accesibles no sólo desde objetos del tipo definido, sino también como funciones "normales". Además, al crearse muchos métodos basados en funciones externas normalmente se termina creando un conflicto de nombres: más de una función auxiliar externa con el mismo nombre, de modo que el navegador no sabe qué función es la que debe tomar entre las dos (o más) con el mismo nombre. Por ello recomendamos seguir la definición de tipos de objeto basada en funciones internas antes que este otro método (que también es muy popular).

EJERCICIO

Define un tipo de objeto Medico en JavaScript que tenga como propiedades: nombre (String), personasCuradas (número entero), especialidad (String) y como métodos un método denominado curarPersona y otro método denominado mostrarDatos. El método curarPersona deberá añadir una unidad al valor de la propiedad personasCuradas y el método mostrarDatos deberá mostrar los datos el médico. Por ejemplo, "El médico se llama Juan Eslava, su especialidad es traumatología y lleva curadas 8 personas". Crea dos objetos del tipo definido, e invoca sus métodos para comprobar que funcionan correctamente.

Crea las dos alternativas de código: métodos con funciones internas anónimas o métodos con referencia a funciones externas.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01145E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CONSTRUCTORES JAVASCRIPT. CREAR OBJETOS VACÍOS. AÑADIR PROPIEDADES Y MÉTODOS. OBJETOS ÚNICOS O SINGLETON. EJEMPLOS (CU01145E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº45 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CONSTRUCTORES Y MÁS FORMAS DE CREAR OBJETOS

JavaScript tiene ciertas diferencias respecto a otros lenguajes de programación orientados a objetos. En JavaScript no existe el concepto de clase ni el concepto de constructor propiamente dicho, aunque por analogía con otros lenguajes muchas veces usamos estos términos.

CONSTRUCTORES EN "CLASES" JAVASCRIPT

En programación orientada a objetos se denomina constructor al código que se ejecuta cuando se crea un objeto de un tipo determinado (a cada objeto creado se le denomina instancia). En otros lenguajes el constructor se delimita dentro de etiquetas separándolo del resto del código de la clase, pero en JavaScript esto no es así. Cuando se instancia un objeto, funciona como constructor todo el código declarado dentro de la función, es decir, todo código dentro de la función que pueda ser ejecutado será ejecutado (no se ejecutarán los métodos de la función, ya que será necesario invocarlos para que se ejecuten).

Escribe este código, guárdalo con extensión html y comprueba lo que ocurre.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function Taxi (tipoMotor) {
this.tipoMotor = tipoMotor;
alert('Se ha creado un objeto taxi con tipo motor: ' + this.tipoMotor);
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
}

function ejemploObjetos() {
var coche1 = new Taxi('Diesel');
var coche2 = new Taxi('Gasolina');
alert ('El coche 1 tiene capacidad ' + coche1.getCapacidad() + ' litros\n');
alert ('El coche 2 tiene capacidad ' + coche2.getCapacidad() + ' litros\n');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Al pulsar sobre el texto "Probar" comprobamos que se muestran los mensajes:

Se ha creado un objeto taxi con tipo motor: Diesel. Se ha creado un objeto taxi con tipo motor: Gasolina El coche 1 tiene capacidad 40 litros. El coche 2 tiene capacidad 35 litros

Como obervamos, cada vez que se instancia (crea un objeto) de la clase, se ejecuta el código asociado.

En este ejemplo tenemos el código this.tipoMotor = tipoMotor; donde debemos recordar que this.tipoMotor alude a la propiedad de la clase, mientras que tipoMotor es el parámetro recibido. Dado que ambos tienen el mismo nombre, la forma de diferenciar cuándo nos referimos a la propiedad y cuándo al parámetro es usar this cuando queramos referirnos a la clase. En general, toda propiedad y todo método dentro de la clase irán antecedidos de la palabra clave this para indicar que nos referimos a propiedades y métodos y no a variables cualquiera.

MÁS FORMAS DE CREAR OBJETOS

Hemos visto que podemos declarar un tipo y luego crear objetos usando la invocación basada en new nombreDelObjeto. Existen más alternativas para crear objetos:

Creación de un objeto vacío

Para crear un objeto vacío podemos usar esta sintaxis.

```
var objetoCreado = { };
```

Esta sintaxis equivale a escribir var objetoCreado = new Object ();

El objeto vacío creado con esta sintaxis pertenece al tipo de objeto predefinido de JavaScript al cual pertenecen todos los demás objetos existentes: Object.

Además existe un objeto Object, objeto predefinido de JavaScript, que nos facilita métodos para manipular objetos. No confundir el tipo de dato Object con el objeto predefinido Object. Prueba este ejemplo donde se usa el objeto Object para definir una propiedad de un objeto creado inicialmente vacío:

```
var pintura1 = {};
Object.defineProperty(pintura1, 'autor', {value: 'Vincent Van Gogh', writable:true, enumerable:true, configurable:true});
alert ('La propiedad autor del objeto pintura1 es: ' + pintura1.autor);
pintura1.autor = 'Michelangelo';
alert ('La propiedad autor del objeto pintura1 es: ' + pintura1.autor);
pintura1.deletrear = function () {
var letras = []; var msg = '';
for(var i=0; i<pintura1.autor.length;i++){msg = msg+pintura1.autor[i]+'-'; }
alert (msg);
}
pintura1.deletrear();
```

Con este código en primer lugar creamos un objeto vacío pintura1. Luego le añadimos una propiedad valiéndonos del método defineProperty del objeto predefinido Object. En este método pasamos como parámetros el objeto del que definimos la propiedad (pintura1), el nombre de la propiedad (autor), el valor de la propiedad (Vincent Van Gogh) y sus características writable, enumerable y configurable (no vamos ahora a hablar sobre estas características). Posteriormente, accedemos a la propiedad y la modificamos para que pase a valer 'Michelangelo'. Finalmente definimos una función para el objeto creado. A continuación resumimos la sintaxis utilizada. Adición de una propiedad a un objeto:

```
Object.defineProperty(nombreObjeto, 'nombrePropiedad', {value: 'valorAsignado', writable:true, enumerable:true, configurable:true});
```

Adición de una función a un objeto:

```
nombreObjeto.nombreFuncionDefinimos = function () { ... código ... }
```

Definición y creación simultánea de un objeto

Para definir y crear un objeto en un solo paso usamos esta sintaxis:

```
var nombreObjetoCreado = {
    propiedad1: valorPropiedad1,
    propiedad2: valorPropiedad2,
    propiedadN: valorPropiedadN,
    ...
    método1: function () { ... código ... }
    método2: function (par1, par2, ..., parN) { ... código ... }
    métodoN: function () { ... código ... }
}
```

Comprueba los resultados de ejecutar este código:

```
var pintura2 = {
   autor: "Vincent Van Gogh",
   añoCreacion: 1871,
   titulo: 'Los rosales corintios',
   getInfo: function () {    return this.titulo + ':' + this.autor + ', ' + this.añoCreacion +'.';    }
}
alert(pintura2.getInfo());
```

Con esta sintaxis no se pueden crear objetos adicionales del tipo definido. El único objeto existente es el que hemos definido y creado simultáneamente. Algunos autores denominan a esta construcción un Singleton (objeto único) por analogía con el patrón Singleton usado en otros lenguajes orientados a objetos para crear objetos únicos.

EJERCICIO

Una de las utilidades de crear objetos vacíos es evitar conflictos de nombres. Supón que creas funciones como:

```
function crearEntrada() {
   // hacer algo
}

function crearSalida() {
   // hacer algo
}
```

El problema que se presenta es que en otro momento se pueda definir otra función con el mismo nombre que alguna de las ya definidas, creando un conflicto de nombres.

Crea un objeto vacío denominado GestionDeUsuarios y añádele dos métodos: un método preguntarNombre y un método despedir. Al invocar GestionDeUsuarios.preguntarNombre(user) se debe crear un objeto de tipo usuario con id de usuario user y almacenar su nombre e id de usuario. Al invocar el método GetionDeUsuarios.despedir(user) se debe mostrar un mensaje de despedida "Hasta luego nombreDeUsuario" donde nombreDeUsuario será el nombre correspondiente.

Responde la siguiente pregunta: ¿si se crea una función despedir entrará en conflicto con el método definido?

Nota: una de las librerías más populares construidas para JavaScript es jQuery. jQuery dispone de un objeto global al que se denomina $, de modo que se accede a las funciones de jQuery escribiendo algo como $.nombreFuncion(...). Esto es, con la diferencia de escala, algo similar a la creación del objeto GestionDeUsuarios visto en el ejemplo. $ evita que jQuery entre en conflictos de nombres.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01146E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

¿ARRAYS ASOCIATIVOS JAVASCRIPT? ¿MAPS? RECORRER PROPIEDADES DE OBJETOS CON FOR IN. EJEMPLOS EJERCICIOS RESUELTOS. (CU01146E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº46 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

¿ARRAYS ASOCIATIVOS JAVASCRIPT?

Con frecuencia se oye que los objetos JavaScript definen arrays asociativos. ¿Existen los arrays asociativos en JavaScript de la misma forma en que se conocen en otros lenguajes? También se alude muchas veces a los objetos JavaScript como maps. Vamos a tratar de aclarar esta terminología y su significado.

PROPIEDADES ¿DEFINEN ARRAYS ASOCIATIVOS?

Hemos visto que generalmente definimos las propiedades y luego accedemos a ellas con la notación de punto como nombreObjeto.nombrePropiedad. Pero hay otra forma alternativa de acceso: la basada en usar una notación similar a la de acceso a los elementos de un array.

La sintaxis de notación tipo array para acceder a las propiedades de un objeto es la siguiente:

nombreObjeto['nombrePropiedad']

Esta sintaxis equivale a escribir nombreObjeto.nombrePropiedad

Fijarse que en el primer caso nombrePropiedad está entre comillas y en el segundo caso no, aunque también sería válido nombreObjeto[nombreVariableTipoString].

Escribe el siguiente código en tu editor y comprueba cómo el acceso a propiedades con ambas sintaxis genera los mismos resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function TaxiRenault (tipoMotor) { this.tipoMotor = tipoMotor;
this.marca = 'Renault';
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
}
function ejemploObjetos() {
var taxi1 = new TaxiRenault();
alert ('La marca de taxi 1 como taxi1.marca es ' + taxi1.marca);
alert ('La marca de taxi 1 como taxi1.[\'marca\'] es ' + taxi1['marca']);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Si lo deseamos podemos crear objetos carentes de métodos con el número de propiedades que deseemos y después rescatar esas propiedades invocando el elemento con la sintaxis tipo array. La sintaxis es:

```
var nombreDelObjeto = { propiedad1: valorPropiedad1, propiedad2: valorPropiedad2,
propiedadN: valorPropiedadN };
```

Con esta sintaxis algunos caracteres o palabras reservadas no pueden usarse como nombre de propiedad sin encerrarlos entre comillas. Por ejemplo no debemos definir como propiedad +: obtenerSuma (donde obtenerSuma es el nombre de una función asociada). Si queremos que + sea una propiedad escribiremos '+': obtenerSuma. En realidad si usamos esta sintaxis todos los nombres de propiedades pueden encerrarse entre comillas porque JavaScript los considera de tipo String.

Resulta válido tanto var pintor = {'edad':32, 'nombre':'jose'} como var pintor = {edad:32, nombre:'jose'}

Con frecuencia se dice que los objetos JavaScript funcionan como Maps o que hacen un mapeo entre Strings y valores. El concepto de Map es similar al de diccionario: en un Map tenemos muchos pares (key, value) o (llave, valor). La llave es un identificador breve al que está asociado algo más complejo (el valor). Por ejemplo podríamos crear un map donde la llave es el número de pasaporte y el valor un objeto que porta todos los datos de una persona (nombre, domicilio, lugar de nacimiento, etc.). En un diccionario las claves o llaves son los términos y los valores las definiciones del término. Para encontrar una definición, buscamos la llave y a continuación leemos la definición.

En otras ocasiones se dice que los objetos JavaScript definen arrays asociativos porque permiten el acceso a las propiedades a través de la **sintaxis propia de arrays** usando como índice el nombre de la propiedad. ¿Pero realmente se genera un array al crear un objeto?

Vamos a comprobarlo. Escribe el siguiente código y ejecútalo. A continuación comentaremos las conclusiones que podemos sacar de él.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() {
var ejemploArray = new Array();
ejemploArray[0] = 'verde'; ejemploArray[1] = 'Argentina'; ejemploArray[2] = 'mate';
alert ('La longitud de ejemploArray es ' + ejemploArray.length);
var ejemploPropiedades = new Array();
ejemploPropiedades['color'] = 'verde'; ejemploPropiedades['pais'] = 'Argentina'; ejemploPropiedades['bebida'] = 'mate';
alert ('La longitud de ejemploPropiedades es ' + ejemploPropiedades.length);
alert ('La propiedad color vale ' + ejemploPropiedades['color']);
alert ('El índice cero del array vale ' + ejemploPropiedades[0]);
for (var i=0; i<ejemploArray.length; i++) { alert ('Valor en ejemploArray: ' + ejemploArray[i]); }
for (var i=0; i<ejemploPropiedades.length; i++) { alert ('Valor en ejemploPropiedades ' + ejemploPropiedades[i]); }
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

Al ejecutar este código el resultado esperado es el siguiente:

La longitud de ejemploArray es 3 -- > Para el array creado con índices numéricos
La longitud de ejemploPropiedades es 0 -- > Para el array creado añadiendo propiedades nombreArray['x'] = 'valor';
La propiedad color vale verde -- > La propiedad se puede rescatar
El índice cero del array vale undefined -- > No existe elemento índice cero ¿entonces es esto un array?
Valor en ejemploArray: verde, Valor en ejemploArray: Argentina, Valor en ejemploArray: mate

Comentarios: este código no hace lo que podría suponerse que debería hacer. Vamos a plantear y responder preguntas relativa a lo que ocurre cuando se ejecuta este código.

¿Es ejemploPropiedades un array? Sí, puesto que lo hemos creado con new Array().

¿Qué contiene el elemento con índice cero del array? No contiene nada (contiene undefined), ya que no ha sido definido, ni este ni ningún otro elementos. El array tiene cero elementos (está vacío).

¿Entonces al definir ejemploPropiedades['color'] = 'verde'; no estamos definiendo un elemento del array? No, con esa sintaxis lo que estamos haciendo es añadir una propiedad al objeto. Una propiedad no es un elemento en la colección de elementos que constituye un array, sino algo propio del objeto (y por tanto inherente al objeto dentro del cual estaría la colección de elementos). La confusión puede venir porque en otros lenguajes no existe esta sintaxis para acceder a las propiedades de un objeto y porque en otros lenguajes con arrays asociativos se permite el uso indistinto de cadenas o números para acceder a los elementos del array. Tenemos que pensar que JavaScript es JavaScript y no sigue necesariamente las convenciones o pautas habituales en otros lenguajes.

¿Por qué ejemploPropiedades.length devuelve cero? Porque no existen (al menos por el momento) elementos en el array. Por tanto no podemos recorrer este array con un bucle for porque no existen elementos en el array.

¿Cómo podría conocer o recorrer las propiedades existentes en un objeto?

Con un for in. La sintaxis a emplear para recorrer las propiedades de un objeto es la siguiente:

```
    for (nombrePropiedad in nombreObjeto) {
    ... ejecución de sentencias ...
}
```

Escribe este código y comprueba el resultado:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() { var ejemploPropiedades = new Array();
ejemploPropiedades['color'] = 'verde'; ejemploPropiedades['pais'] = 'Argentina'; ejemploPropiedades['bebida'] = 'mate';
msg = '';
```

```
for(nombrePropiedad in ejemploPropiedades) {
msg = msg + 'Propiedad: ' + nombrePropiedad + '. Valor propiedad es: '+ejemploPropiedades[nombrePropiedad]+ ' \n';
}
alert (msg);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado esperado es que se muestre por pantalla:

Propiedad: color. Valor propiedad es: verde
Propiedad: pais. Valor propiedad es: Argentina
Propiedad: bebida. Valor propiedad es: mate

¿Las propiedades y forma de acceso "asociativo" existen para cualquier objeto aunque no sea de tipo Array?

Sí, el acceso basado en la sintaxis nombreObjeto.['nombrePropiedad'] existen para cualquier objeto.

¿Es correcto o no hablar de arrays asociativos en JavaScript?

La especificación oficial JavaScript no hace uso del término array asociativo y nosotros preferimos no hacer uso del término array asociativo sino referirnos a sintaxis de acceso de tipo array. No obstante, encontrarás que mucha gente hace uso de este término (algunos por desconocimiento, pero otros porque lo consideran adecuado). Grandes expertos en JavaScript hacen uso del término y otros grandes expertos indican que no se debe hacer uso del término.

Si te fijas usando el for in puedes conseguir que un objeto se comporte de forma muy parecida a como lo haría un array. Ten clara la sintaxis y las posibilidades que existen, y respecto a si usar o no el término, haz lo que consideres más oportuno.

¿Para qué es útil el acceso a propiedades con la sintaxis tipo array?

Algunos programadores la usan porque les gusta, pero hay casos en que es la única forma de acceder a una propiedad. Supón que el nombre de una propiedad lo establece el usuario (o se crea dinámicamente de alguna manera) introduciendo un dato que almacenas en una variable denominada nombrePropiedad. En este caso no es posible acceder a la propiedad con la sintaxis nombreObjeto.nombrePropiedad, pero sí será posible acceder con la sintaxis nombreObjeto[nombrePropiedad].

EJERCICIO

El siguiente código hace uso de la notación tipo array para invocar propiedades. También crea objetos únicos (los objetos plus, minus, operaciones y calcular). Analiza el código y trata de comprenderlo.

Se pide realizar los siguientes cambios:

a) Reemplaza toda la notación basada en sintaxis tipo array para el acceso a propiedades por sintaxis basada en notación de punto. Ejecuta el código y comprueba su funcionamiento.

b) Sobre el código de la opción a), cambia la definición de objetos para que no sean objetos únicos, sino que plus, minus y calcular sean funciones simples, y operaciones un objeto instanciable (que tendrás que instanciar si es necesario). Ejecuta el código y comprueba su funcionamiento.

c) Sobre el código de la opción c), añade la posibilidad de hacer cálculos de multiplicación y división de la misma forma que se hacen cálculos de suma y resta. Muestra un mensaje por cada tipo de operación. Ejecuta el código y comprueba su funcionamiento.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var plus = function(x,y){ return x + y };
var minus = function(x,y){ return x - y };
var operaciones = {
  '+': plus,
  '-': minus
};
var calcular = function(x, y, operacion){ return operaciones[operacion](x, y); }
function ejemploObjetos() {
alert ('Resultado de calcular(3, 15, \'+\') es '+ calcular(3,15, '+'));
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Este código nos recuerda, entre otras cosas, que: un objeto se puede crear usando la palabra clave var, una función se puede crear usando la palabra clave var, y una propiedad de un objeto puede ser una función. Esta nomenclatura es un poco “complicada”, pero hay que irse acostumbrando poco a poco a ella.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01147E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

PROTOTYPE JAVASCRIPT. EJEMPLOS DE PROTOTIPOS Y HERENCIA. CÓMO USARLOS. SINTAXIS. EFICIENCIA. (CU01147E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº47 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

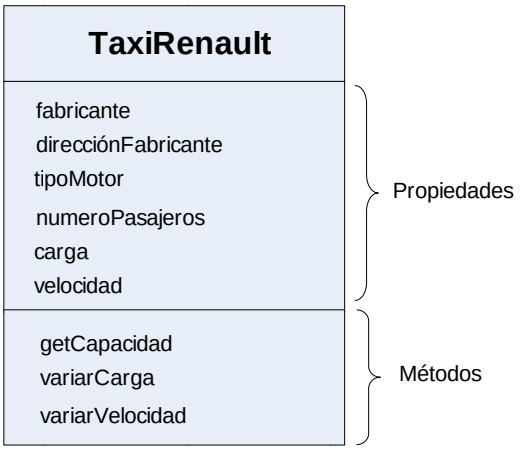
HERENCIA Y PROTOTYPE

Hasta ahora hemos visto como definir objetos con propiedades y métodos y cómo instanciarlos. Trataremos ahora de ver cómo podemos hacer que muchos objetos de un mismo tipo compartan (hereden) propiedades y métodos de modo que se optimice el código.

Cada vez que instanciamos un objeto con new se crea una instancia que "porta" espacio de memoria donde están las propiedades y los métodos.

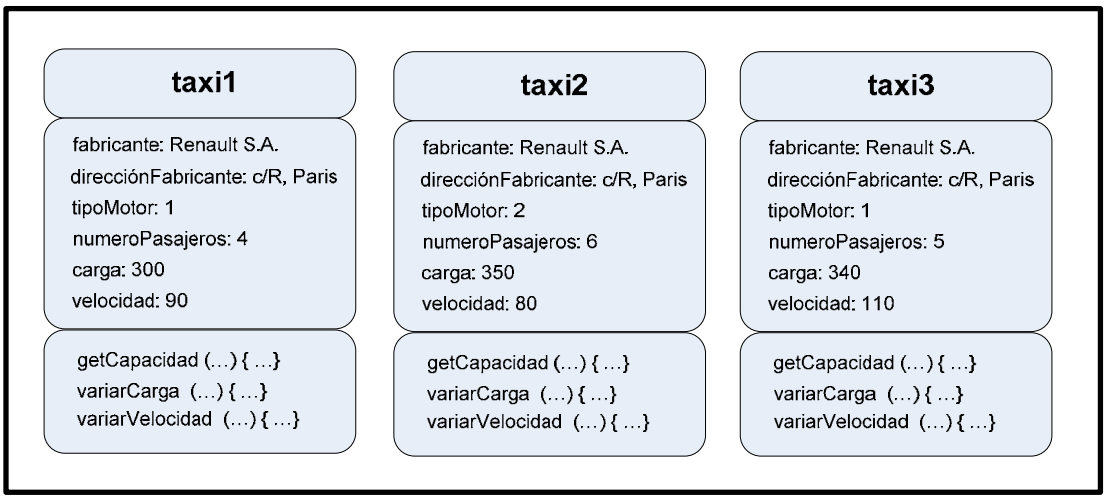
Supongamos que hemos definido un objeto TaxiRenault cuyas propiedades son fabricante, direccionFabricante, tipoMotor, numeroPasajeros, carga, y velocidad, y cuyos métodos son getCapacidad (devuelve la capacidad del depósito de combustible), variarCarga y variarVelocidad (suponemos que la carga y velocidad aumentan si se pasa un valor positivo a los métodos responsables, o que disminuyen si se pasa un valor negativo). Suponemos que propiedades y métodos se han definido usando la palabra clave this, lo cual significa que pertenecen al objeto.

Este esquema resume cómo se ha concebido el patrón o clase para los objetos de tipo TaxiRenault:



Renault podría haber sido el valor de la propiedad marca de un objeto Taxi (en vez de usar como objeto TaxiRenault), pero ten en cuenta que esto es un ejemplo para estudiar la herencia, no pretendemos que uses esto como código real.

Supongamos ahora que creamos tres objetos de tipo TaxiRenault mediante la sentencia new, a los que denominamos taxi1, taxi2 y taxi3. Un esquema para representarlos podría ser el siguiente:



Aquí vemos que cada objeto cuenta con sus propiedades y con sus métodos. Cada propiedad y cada método ocupa espacio de memoria y genera "una carga" a la hora de trabajar con ellos. También observamos que algunas propiedades siempre tienen el mismo valor en todos los objetos. En este ejemplo la propiedad fabricante siempres es Renault S.A., y la dirección del fabricante siempre es c/R, París. Otras propiedades (aunque casualmente puedan tener el mismo valor en varios objetos) son las que realmente caracterizan al objeto (tipoMotor, numeroPasajeros, carga, velocidad), es decir, son propiedades inherentes a cada objeto individual y específico, no compartidas por todos los objetos.

En cuanto a los métodos, todos los métodos hacen lo mismo: reciben (o no) unos datos, y realizan un proceso. En todos los objetos los métodos son iguales pero están repetidos.

Escribe y prueba el siguiente código, que refleja al esquema anterior.

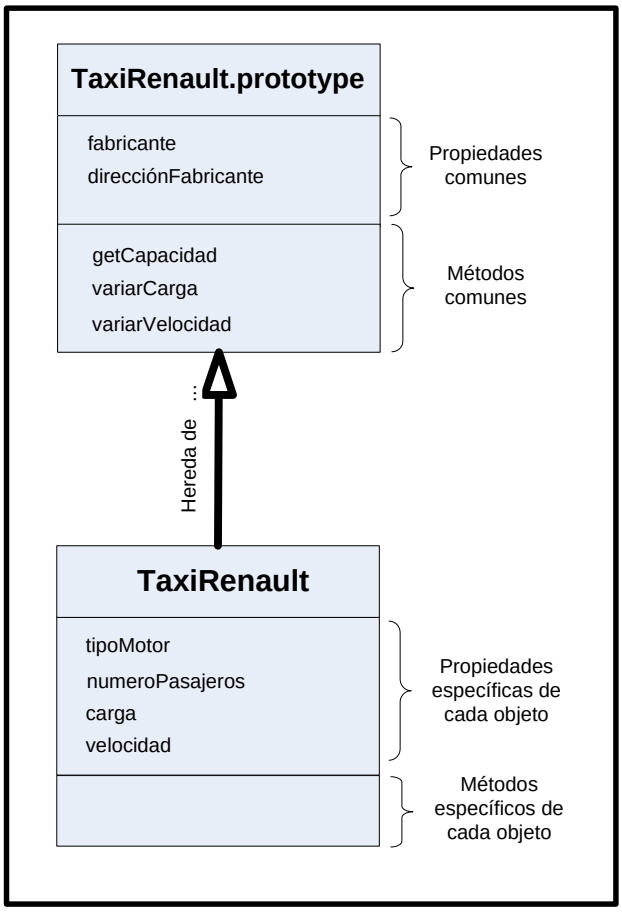
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.fabricante = 'Renault, S.A.'; this.direccionFabricante = 'c/R, Paris';
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga;      this.velocidad = velocidad;
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
this.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
this.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
}
function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('La velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10);
alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

Si los objetos TaxiRenault fueran objetos muy complejos y generáramos cientos o miles de objetos de esta manera nos encontraríamos con que podemos saturar la memoria disponible y ralentizar la ejecución de la página web, o incluso generar un bloqueo.

HERENCIA BASADA EN PROTOTIPOS

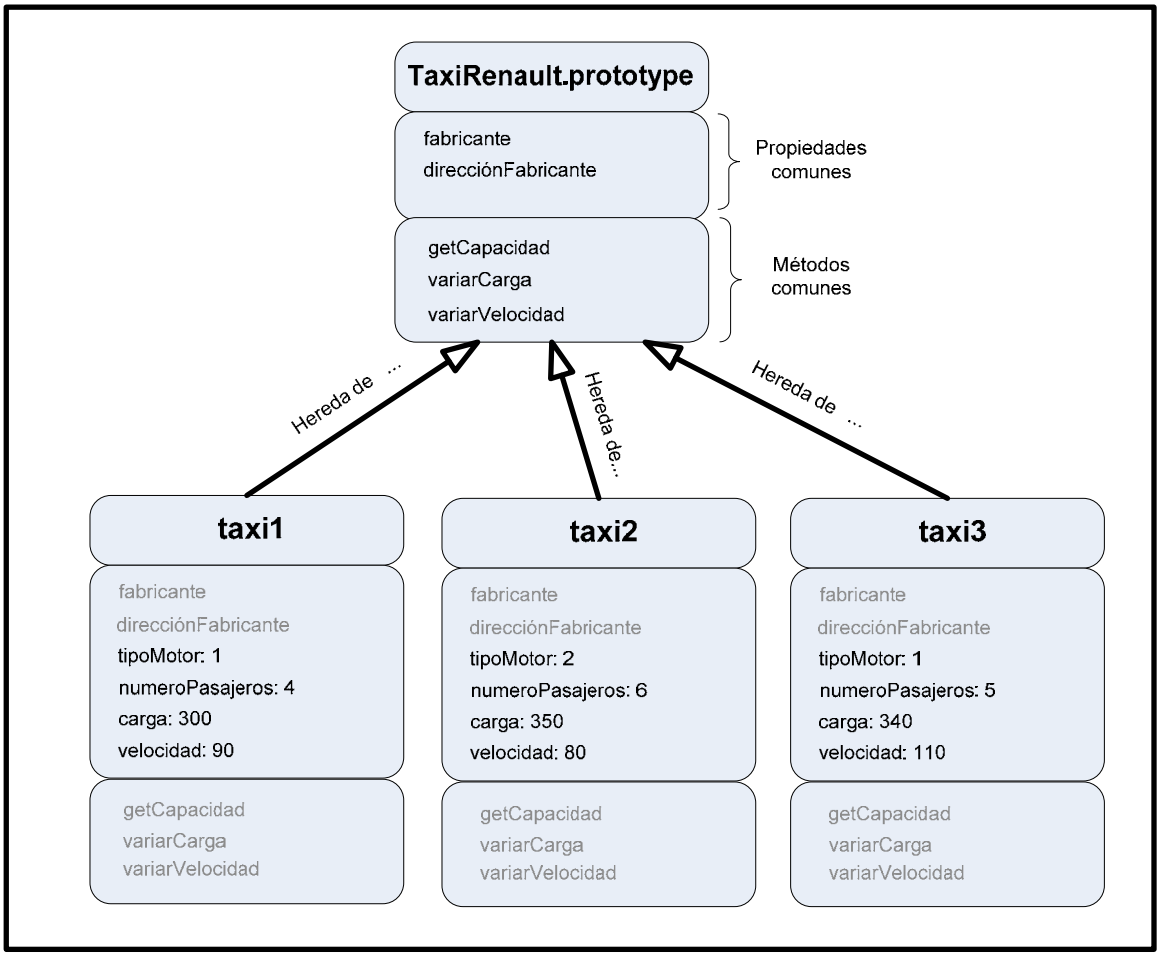
La idea de la herencia basada en prototipos JavaScript es definir un objeto (el objeto "padre" o prototipo) donde se aloja toda la información común que comparten todos los objetos de ese tipo (los objetos "hijos"). De esta manera se evita que cada objeto repita las propiedades y métodos comunes, lo cual ahorra memoria y agiliza la ejecución.

Las propiedades y métodos comunes se alojan en el prototipo u objeto padre, lo cual podemos representar con un esquema como este:



La ventaja de este esquema es que las propiedades y métodos comunes sólo existen una vez. Cuando un objeto invoca una propiedad, por ejemplo taxi1.fabricante, se comprueba si dicha propiedad está definida como propiedad específica del objeto, y si no es así, se busca en su prototipo de modo que si la propiedad existe en el prototipo, se devuelve esa propiedad de la misma manera que si fuera una propiedad del objeto. Si no se encontrara en el prototipo, se buscaría en el prototipo del prototipo (el padre del padre...) y así sucesivamente hasta encontrarla o no: a esto se le denomina cadena de prototipos o prototype chain y hablaremos sobre esto más adelante.

Al crear varios objetos TaxiRenault el esquema sería como este:



Lo que este esquema representa es que los objetos taxi1, taxi2 y taxi3 disponen de los atributos fabricante y direcciónFabricante, así como de los métodos getCapacidad, variarCarga y variarVelocidad. Pero estos atributos y métodos ya no están repetidos en cada uno de los objetos (no son propiedades y métodos de instancia) sino que están alojados en el prototipo, evitando así la repetición de información y sobrecarga de memoria.

Ten en cuenta que el uso de prototipos y herencia no es algo que haya que utilizar "para todo": sólo debemos pensar en ello cuando sea necesario. ¿Cuándo es necesario? Esta pregunta debe ser respondida para cada problema particular, pero en general piensa que no será necesario pensar en prototipos y herencia cuando sólo se vayan a crear unos pocos objetos. En cambio, en aplicaciones como juegos donde se usan intensivamente objetos (por ejemplo para representar meteoritos cayendo desde el espacio) será útil el uso de prototipos.

Si has trabajado previamente con otro tipo de lenguajes orientados a objetos (por ejemplo Java), posiblemente estés tratando de buscar las similitudes entre Java y JavaScript. Podríamos hacerlo, pero posiblemente esto nos generaría un lío de conceptos. Intenta olvidarte de las clases y mecanismos de herencia propios de otros lenguajes y cambia el chip: piensa en JavaScript.

A diferencia de otros lenguajes (como Java) donde existen clases y objetos, en JavaScript deberíamos decir que existen prototipos y objetos, aunque en muchos casos se usa la palabra clase por analogía.

DEFINIR PROTOTIPOS

En JavaScript el prototipo de un objeto está definido de forma predeterminada como una "propiedad oculta" de todo objeto, a la que podemos acceder con la sintaxis: nombreObjeto.prototype. El prototipo es a su vez un objeto (recuerda que una propiedad puede ser tanto un tipo primitivo como un objeto) y por tanto a dicho objeto se le pueden añadir propiedades y métodos.

Dado que JavaScript ofrece numerosas alternativas para trabajar con objetos, propiedades y métodos, existen numerosas alternativas para definir el prototipo de un objeto.

Alternativa 1: definir propiedades y métodos con la notación de punto:

```
function nombreObjeto (par1, par2, ..., parN) {
    this.propiedad1 = valorPropiedad1;
    this.propiedad2 = valorPropiedad2;
    ...
    }
nombreObjeto.prototype.propiedadComún1 = valorPropieadadComún1;
nombreObjeto.prototype.propiedadComún2 = valorPropieadadComún2;
...
nombreObjeto.prototype.métodoComún1 = function (par1, par2, ...) { ... }
nombreObjeto.prototype.métodoComún2 = function (par1, par2, ...) { ... }
...
```

Escribe este código y comprueba los resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros; this.carga = carga;
this.velocidad = velocidad;
}
TaxiRenault.prototype.fabricante = 'Renault, S.A.';
TaxiRenault.prototype.direccionFabricante = 'c/R, Paris';
TaxiRenault.prototype.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
TaxiRenault.prototype.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
TaxiRenault.prototype.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10); alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

Alternativa 2: definir un objeto y convertir una instancia de dicho objeto en prototipo:

```
function nombreObjeto (par1, par2, ..., parN) {
this.propiedad1 = valorPropiedad1; this.propiedad2 = valorPropiedad2; ...
}
function objetoDestinadoASerPrototipo (par1, par2, ...) { ... }
nombreObjeto.prototype = new objetoDestinadoASerPrototipo();
```

Escribe este código y comprueba los resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}
function prototipoTaxiRenault () {
this.fabricante = 'Renault, S.A.'; this.direccionFabricante = 'c/R, Paris';
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
this.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
this.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
}

TaxiRenault.prototype = new prototipoTaxiRenault();

function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10); alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Las propiedades y métodos de un prototipo a veces se dice que son propiedades y métodos de clase o estáticas tratando de buscar analogías con otros lenguajes.

Alternativa 3: definir un objeto y asignarle contenido a su propiedad prototype creando un objeto único con la sintaxis de literal (propiedades enumeradas con dos puntos y separadas por comas):

```
function nombreObjeto (par1, par2, ..., parN) {
this.propiedad1 = valorPropiedad1; this.propiedad2 = valorPropiedad2; ...
}
nombreObjeto.prototype = {
propiedadComún1: valorPropiedadComún1;
propiedadComún2: valorPropiedadComún2;
...
métodoComún1: function (par1, par2, ..., parN) { ... },
métodoComún2: function (par1, par2, ..., parN) { ... },
...
}
```

Escribe este código y comprueba los resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

TaxiRenault.prototype = {
fabricante: 'Renault, S.A.',
direccionFabricante: 'c/R, Paris',
getCapacidad: function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} },
variarCarga: function (variacion) { this.carga = this.carga + variacion; },
variarVelocidad: function (variacion) { this.velocidad = this.velocidad + variacion; }
}

function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10);
alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Podríamos especificar varias formas más de definir el prototipo de un objeto, pero con lo visto hasta ahora podemos seguir avanzando.

EJERCICIO

Define un tipo de objeto Cometa cuyas propiedades de instancia (específicas de cada objeto) sean diametro, temperatura y nombre. La temperatura será un valor numérico que suponemos está en grados centígrados. Como propiedad común a todos los objetos de tipo cometa define definicionSegunDiccionario (que debe contener la definición de cometa según el diccionario) y como métodos comunes obtenerRadio (que debe devolver el radio) y obtenerTemperaturaFarenheit (que debe devolver el valor de temperatura expresado en grados Farenheit). Crea tres objetos de tipo cometa y comprueba que puedes acceder tanto a las propiedades específicas como a las propiedades comunes y métodos comunes desde cada objeto.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01148E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

STATIC JAVASCRIPT. PROPIEDADES Y MÉTODOS ESTÁTICOS O "DE CLASE". EJERCICIO. CÓDIGO EJEMPLOS BÁSICOS (CU01148E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº48 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

STATIC JAVASCRIPT

A diferencia de otros lenguajes, JavaScript no utiliza la palabra clave static para definir variables o métodos estáticos, aunque sí provee de la posibilidad de definir propiedades y métodos únicos asociados a un tipo definido equivalentes a las propiedades y métodos estáticos.

Recordar que si escribimos una definición de tipo de objeto basada en function y dentro de ella un método o propiedad antecedida de la palabra clave this, cada vez que se crea una instancia con new se generan copias de las propiedades y métodos, cosa que ya hemos visto que puede resultar ineficiente. Hemos visto como alternativa, definir propiedades y métodos en el prototipo que son heredados por todas las instancias de ese tipo de objeto. De este modo, todos los objetos tienen acceso a esa propiedad o método por herencia prototípica: cuando se invoca la propiedad o método sobre el objeto y no se encuentra, se procede a la búsqueda en el objeto prototipo (el "padre" del objeto).

La ventaja de usar prototype es que estos propiedades y métodos sólo existen una vez en memoria y no generan duplicados para cada objeto, y así es más eficiente el código. Además, las propiedades y métodos pueden transmitirse a lo largo de una cadena de herencia y ser accesibles desde cualquier objeto.

SIMULAR PROPIEDADES Y MÉTODOS ESTÁTICOS

Tenemos aún otra manera de generar propiedades y métodos: declararlos como propiedades y métodos asociados al objeto que define el tipo (lo que llamaríamos "la clase"), de modo que sólo serán accesibles invocando al nombre del tipo de objeto, pero no a través de las instancias. Esto se asemeja mucho a lo que en otros lenguajes se denomina propiedades y métodos estáticos, de ahí que por analogía muchas veces se aluda a este tipo de propiedades y métodos como estáticos.

Para definir métodos y propiedades que simulan ser estáticos podemos hacerlo fuera de la función constructora con esta sintaxis:

```
function nombreObjeto (par1, par2, ..., parN) {
    this.propiedad1 = valorPropiedad1;
    this.propiedad2 = valorPropiedad2;
    ... }
nombreObjeto.nombrePropiedadEstática1 = valorPropiedadEstática1;
nombreObjeto.nombrePropiedadEstática1 = valorPropiedadEstática2;
...
nombreObjeto.métodoEstático1 = function (par1, par2, ...) { ... }
nombreObjeto.métodoEstático2 = function (par1, par2, ...) { ... }
...
```

También es sintácticamente posible incluir la propiedad o método estático dentro de la función constructora con esta sintaxis, que es análoga a la anterior con la diferencia de que la declaración se realiza dentro de la función:

```
function nombreObjeto (par1, par2, ..., parN) {
    this.propiedad1 = valorPropiedad1;
    this.propiedad2 = valorPropiedad2;
    ...
nombreObjeto.nombrePropiedadEstática1 = valorPropiedadEstática1;
nombreObjeto.nombrePropiedadEstática1 = valorPropiedadEstática2;
    ...
  nombreObjeto.métodoEstático1 = function (par1, par2, ...) { ... }
  nombreObjeto.métodoEstático2 = function (par1, par2, ...) { ... }
    ...
    }
```

Una propiedad o método estático no se duplica en cada objeto, sino que existe una única vez en memoria.

Escribe este código y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function Taxi (tipoMotor, numeroPasajeros, carga, velocidad) {
if (Taxi.numeroObjetos) {Taxi.numeroObjetos++;}
else {Taxi.numeroObjetos = 1; }
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
alert('Creado objeto número '+Taxi.numeroObjetos);
}

function ejemploObjetos() {
var taxi1 = new Taxi(1, 4, 300, 90);
var taxi2 = new Taxi(2, 5, 250, 100);
var taxi3 = new Taxi(1, 6, 400, 80);
alert('El número de objetos Taxi creados hasta el momento es ' + Taxi.numeroObjetos);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

La expresión << if (Taxi.numeroObjetos) >> devuelve true si numeroObjetos es una propiedad existente de Taxi y false en caso contrario. Si la propiedad estática no existe cuando se invoca la creación de un objeto con new, se crea con la sentencia Taxi.numeroObjetos = 1;

El resultado esperado es:

Creado objeto número 1, Creado objeto número 2, Creado objeto número 3. El número de objetos Taxi creados hasta el momento es 3.

Si volvemos a ejecutar el script el contador se sigue incrementando (4, 5, 6, si volvemos a ejecutar 7, 8, 9, etc.), excepto si volvemos a recargar la página web en nuestro navegador. Cuando tiene lugar una recarga, todo se inicializa, con lo cual volveríamos a empezar la cuenta del número de objetos por 1.

A continuación un ejemplo de declaración de una propiedad y método estático fuera de la función constructora. Escribe este código y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

TaxiRenault.fabricante = 'Renault, S.A.'; //Propiedad estática
TaxiRenault.mostrarMensaje = function () {alert('Soy un taxi Renault');} //Método estático

function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90);
alert ('La velocidad del taxi 1 es ' + taxi1.velocidad);
TaxiRenault.mostrarMensaje(); //Invocamos el nombre del tipo de objeto
alert ('La propiedad estática fabricante vale ' + TaxiRenault.fabricante);
alert ('Si intentamos obtener la propiedad fabricante para una instancia obtenemos: ' + taxi1.fabricante); //undefined
taxi1.mostrarMensaje(); //ERROR las instancias no tienen acceso a los métodos estáticos
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado de ejecución esperado es:

La velocidad del taxi 1 es 90
Soy un taxi Renault
La propiedad estática fabricante vale Renault, S.A.
Si intentamos obtener la propiedad fabricante para una instancia obtenemos: undefined
... (error, taxi1.mostrarMensaje(); no se ejecuta).

De este ejemplo obtenemos la siguiente conclusión:

No se puede acceder a una propiedad o método estático desde una instancia, hay que hacerlo invocando directamente sobre el nombre del objeto que define el tipo (lo que llamaríamos sobre "la clase").

Modifica el código anterior y escribe:

TaxiRenault.mostrarMensaje = **function** () {alert('Soy un taxi Renault con carga ' + **this**.carga);}

El resultado será Soy un taxi Renault con carga undefined ¿Por qué? Porque this hace referencia al objeto dentro del cual se encuentra la invocación y en este caso estamos trabajando con un método estático que no conoce las propiedades de un objeto en particular.

EJERCICIO

Define un tipo de objeto Meteorito cuyas propiedades de instancia (específicas de cada objeto) sean diametro, temperatura y nombre. La temperatura será un valor numérico que suponemos está en grados centígrados. Como propiedad estática del tipo meteorito define definicionSegunDiccionario (que debe contener la definición de meteorito) y como métodos estáticos obtenerRadio (que debe devolver el radio a partir de un parámetro diametro) y obtenerTemperaturaFarenheit (que debe devolver el valor de temperatura expresado en grados Farenheit a partir de un parámetro temperatura). Crea un objeto de tipo Meteorito con un diámetro, temperatura y nombre y comprueba que puedes acceder a las propiedades y métodos estáticos mostrando por pantalla la definición de meteorito, la superficie del objeto creado y la temperatura Farenheit del objeto creado.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01149E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

HERENCIA JAVASCRIPT: EJEMPLO CON CÓDIGO BÁSICO. JERARQUÍA DE CLASES EN CADENA DE PROTOTIPOS. (CU01149E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº49 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

HERENCIA Y OBJETO PROTOTYPE

JavaScript carece de clases como existen en otros lenguajes, por tanto no existe la herencia como existe en otros lenguajes. Coloquialmente nos referimos en ocasiones a "clases" entre comillas, pero hay que recordar que en JavaScript prácticamente todo con lo que se trabaja son objetos (instancias) y por tanto no es posible diferenciar entre clases e instancias.

No se puede crear por tanto una jerarquía de clases como se hace en otros lenguajes, aunque se puede crear una simulación basada en prototipos.

Todo objeto tiene una propiedad oculta denominada prototype que es un objeto de referencia. Este objeto de referencia actúa como si fuera la "clase padre" donde el objeto consulta atributos y métodos en caso de que no los encuentre definidos como atributos y métodos propios de sí mismo.

Ejemplo:

```
function Taxi (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

Taxi.prototype = { ruedas: 4, saludar: function() {
alert('Hola soy un taxi de ' + this.ruedas + ' ruedas y ' +this.numeroPasajeros +' pasajeros');} }

function ejemploObjetos() {
var taxi1 = new Taxi(1, 6, 300, 90);
taxi1.saludar();
}
```

En este ejemplo, se define la "clase" Taxi y se dota de contenido al objeto prototype asignándole propiedades y métodos. Se crea un objeto taxi1 y se invoca su método saludar(). En primer lugar este método se busca dentro de los métodos de "la clase" y al no encontrarse se busca en el objeto prototype, donde sí se encuentra y se devuelve. Al ejecutar el método saludar() se encuentra una invocación a this.ruedas. Esta propiedad se busca primero en la "clase" y al no encontrarse se busca en el prototipo. Lo mismo ocurre con numeroPasajeros, aunque en este caso sí se encuentra.

El objeto prototype existe como objeto vacío desde que se crea una instancia. ¿Por qué? Porque JavaScript lo crea en segundo plano, como si incluyéramos dentro del código de la clase la sentencia: prototype = {};

Por ello podemos acceder al objeto y asignarle propiedades y funciones y por ello este código de ejemplo es equivalente al anterior:

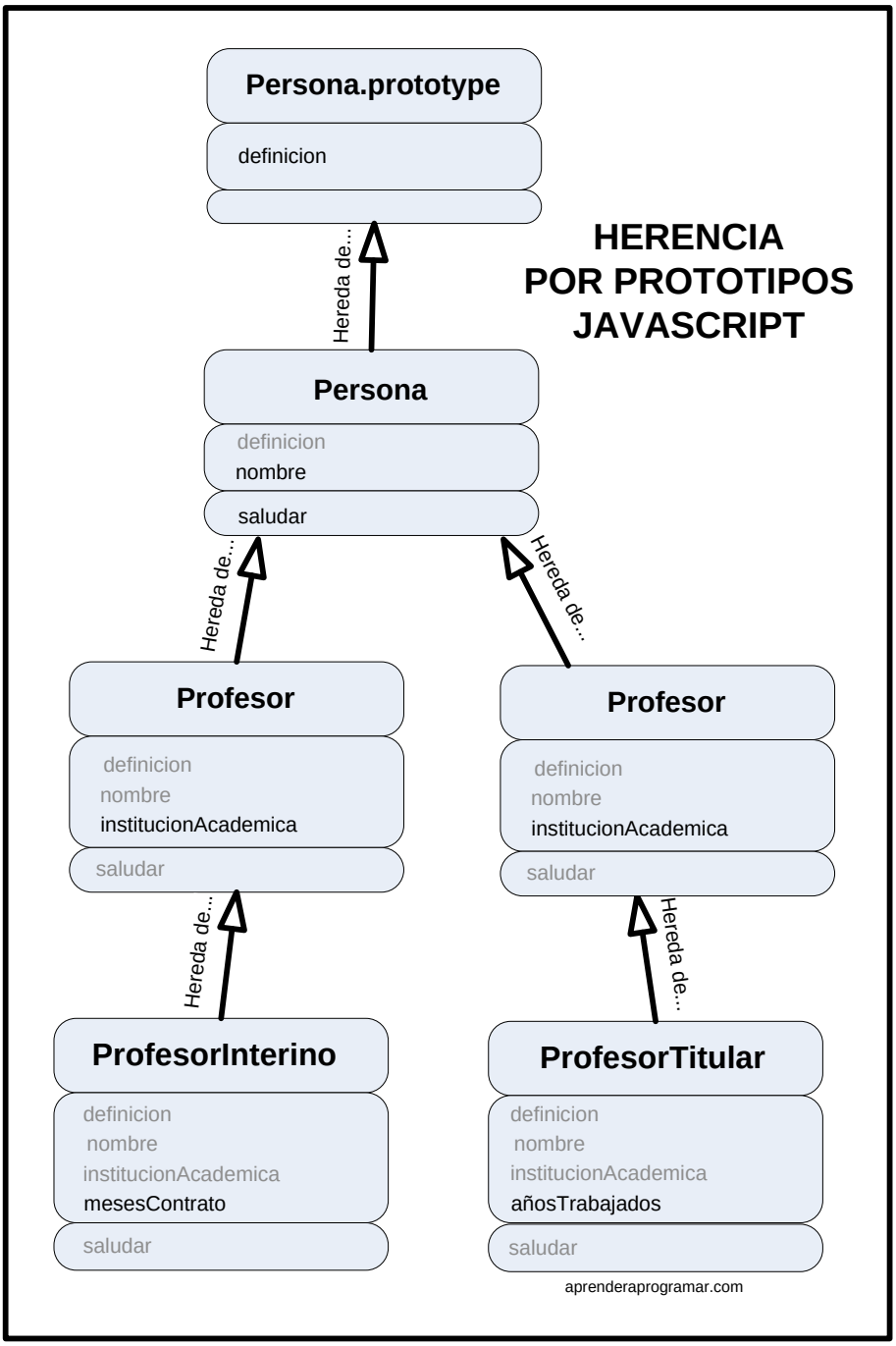
```
function Taxi (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

Taxi.prototype.ruedas = 4;
Taxi.prototype.saludar = function() {
alert('Hola soy un taxi de ' + this.ruedas + ' ruedas y ' +this.numeroPasajeros +' pasajeros');}

function ejemploObjetos() {
var taxi1 = new Taxi(1, 6, 300, 90);
taxi1.saludar();
}
```

Podemos hacer una simulación de "jerarquía de clases" basándonos en los objetos prototipo. Para ello hemos de crear prototipos en cadena. Veamos un código de ejemplo básico:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Persona () { this.nombre = 'Nombre desconocido';
this.saludar = function () {alert('Hola, soy ' + this.nombre);}
}
Persona.prototype.definicion = 'Ser humano';
function Profesor () { this.institucionAcademica = 'Institución desconocida'; }
Profesor.prototype = new Persona();
function ProfesorInterino() { this.mesesContrato = 0;}
ProfesorInterino.prototype = new Profesor();
function ProfesorTitular() { this.añosTrabajados = 0;}
ProfesorTitular.prototype = new Profesor();
function ejemploObjetos() {
var unProfesorTitular = new ProfesorTitular();
alert ('El nombre de unProfesorTitular es ' + unProfesorTitular.nombre);
unProfesorTitular.saludar();
unProfesorTitular.nombre = 'Juan';
unProfesorTitular.institucionAcademica = 'Universidad de Chapingo';
unProfesorTitular.añosTrabajados = 14;
var msg = 'El profesor titular creado tiene nombre ' + unProfesorTitular.nombre + ', trabaja en '
msg = msg + unProfesorTitular.institucionAcademica + ', tiene ' + unProfesorTitular.añosTrabajados + ' años trabajados';
msg = msg + ' y definición de ' + unProfesorTitular.definicion;
alert(msg);
unProfesorTitular.saludar();
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El código anterior se corresponde con este esquema de herencia, donde podemos observar que hay objetos Profesor duplicados ¿Por qué? Lo explicamos a continuación.

Nota: en gris señalamos propiedades o métodos accesibles a través del prototipo. Dichas propiedades o métodos no se encuentran en los objetos en sí, pero son accesibles.



El resultado de ejecución esperado para el código es:

El nombre de unProfesorTitular es Nombre desconocido
Hola, soy Nombre desconocido
El profesor titular creado tiene nombre Juan, trabaja en Universidad de Chapingo, tiene 14 años trabajados y definición de Ser humano
Hola, soy Juan

Analicemos el código paso a paso:

Se define profesorTitular de modo que toda instancia de profesorTitular se inicializará con la propiedad añosTrabajados = 0

Se define que el objeto prototipo de ProfesorTitular es un objeto Profesor.

Se define ProfesorInterino de modo que toda instancia de profesorInterino se inicializará con la propiedad mesesContrato = 0;

Se define que el objeto prototipo de ProfesorInterino es un objeto Profesor.

Se define Profesor de modo que toda instancia de Profesor se inicializará con la propiedad institucionAcademica = 'Institución desconocida';

Se define que el prototipo de Profesor es un objeto Persona.

Por último se define Persona de modo que toda instancia de Persona se inicializará con la propiedad nombre = 'Nombre desconocido'; y se define que el objeto prototipo de Persona (de tipo Object) tiene la propiedad << definicion = 'Ser humano'; >>

En el código de ejecución comenzamos por crear un objeto de tipo profesorTitular al que denominamos unProfesorTitular. Al invocar la propiedad nombre y el método saludar del objeto profesorTitular el intérprete JavaScript busca estas propiedades y métodos en el propio objeto, pero no los encuentra, ya que la única propiedad en el objeto es añosTrabajados. Al no encontrarlas, pasa a buscar las propiedades y métodos en el objeto prototipo (que es un objeto Profesor), pero en este objeto tampoco las encuentra. Finalmente, busca en el prototipo del objeto Profesor que es un objeto Persona. Como todo objeto Persona tiene inicialmente la propiedad nombre = 'Nombre desconocido' y dispone del método saludar, son éstos los que se devuelven. El resultado es que se muestra por pantalla <<El nombre de unProfesorTitular es Nombre desconocido. Hola, soy Nombre desconocido >>

Seguidamente se establecen valores para las propiedades nombre, institucionAcademica y añosTrabajados de unProfesorTitular y se muestra por pantalla un mensaje donde además de estas propiedades se muestra cómo unProfesorTitular dispone de la propiedad <<definicion>> que se obtiene recorriendo la cadena de prototipos.

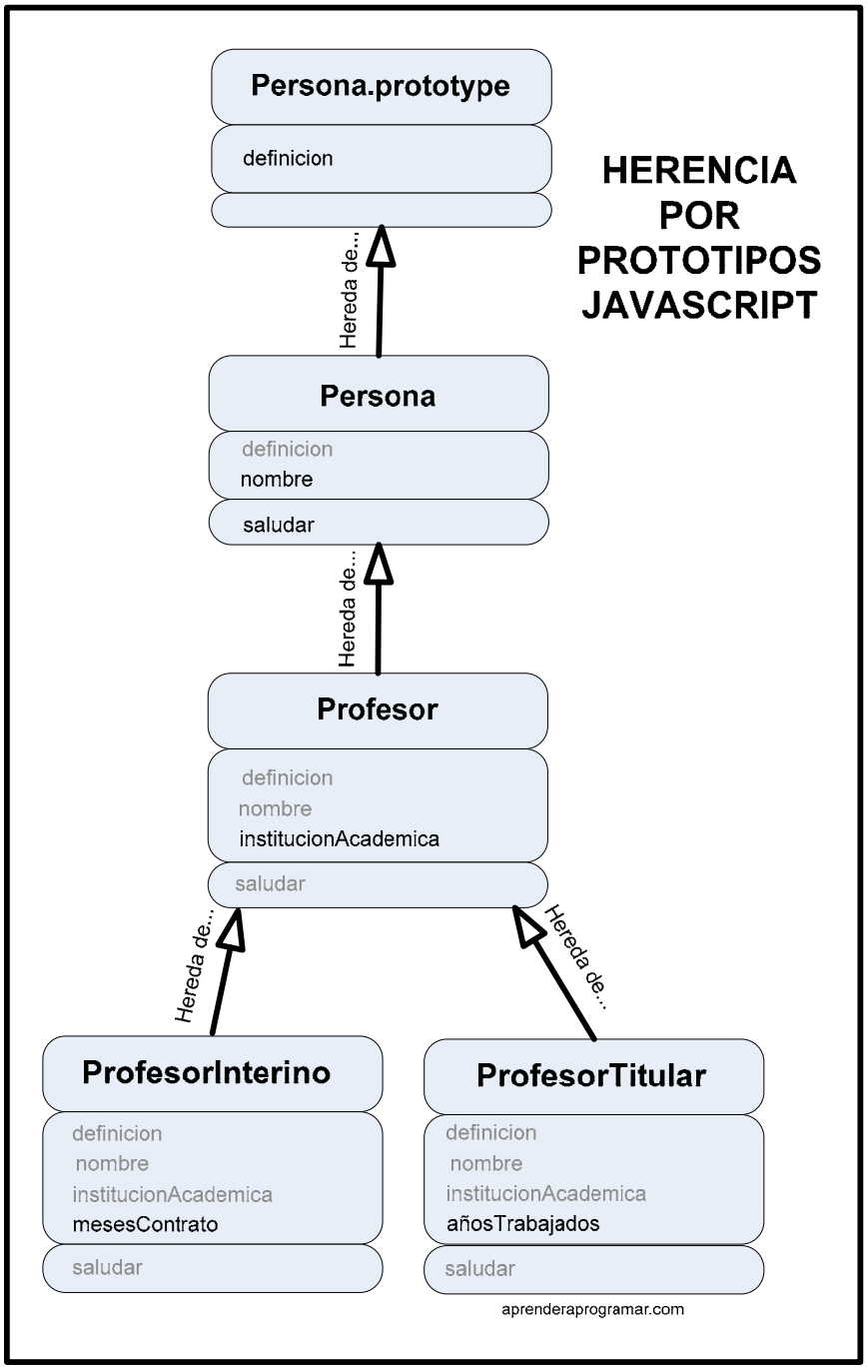
PENSAR EN OBJETOS Y NO EN CLASES

Muchas personas (normalmente los programadores que provienen de Java) pueden pensar que sería más correcto un esquema donde ProfesorInterino y ProfesorTitular dependan de un único Profesor.

Esto podría simularse haciendo que ProfesorInterino y ProfesorTitular compartan un mismo prototipo y esto podría lograrse con un código como este:

```
function Profesor () { this.institucionAcademica = 'Institución desconocida'; }
Profesor.prototype = new Persona();

function ProfesorInterino() { this.mesesContrato = 0;}
ProfesorInterino.prototype = ProfesorTitular.prototype;
```

El esquema sería este:



Esta concepción tiene un inconveniente importante. Dado que las propiedades y métodos comunes ("de clase") en JavaScript se establecen a través del prototipo, usar un mismo objeto como prototipo para los profesores interinos y los profesores titulares nos impediría diferenciar propiedades comunes exclusivas de los profesores interinos y propiedades comunes exclusivas de los profesores titulares. Si el prototipo es un mismo objeto, las propiedades comunes lo serían tanto para profesores interinos como para profesores titulares.

A modo de conclusión: cuando trabajes con JavaScript no pienses en clases, piensa en objetos.

CONSTRUCTORES CON HERENCIA

En el código de ejemplo visto anteriormente trabajamos con objetos que no reciben parámetros para su constructor, sino que inicializan sus propiedades a unos valores de defecto. ¿Cómo implementar herencia de modo que se pueda inicializar un objeto con unos valores concretos? Lo veremos en la próxima entrega.

EJERCICIO

Crea un esquema de herencia en JavaScript que refleje estos requisitos:

a) Hay tres tipos de hortalizas: zanahoria, lechuga y tomate. La zanahoria tiene como propiedad su valor calórico que es de 45 cal, mientras que la lechuga tiene 31 cal y el tomate 39 cal.

b) Toda hortaliza tiene como propiedad específica tipoHortaliza y su valor inicial debe ser “indefinido”.

c) Todas las hortalizas tienen una propiedad común: su componente principal: <<Agua>>

c) Una hortaliza es un tipo de planta cultivada. Una planta cultivada tiene como propiedad específica nombreCientifico y su valor inicial debe ser “desconocido”.

d) Una planta cultivada es un tipo de vegetal. Una propiedad de los vegetales es la movilidad y su valor es común para todos los vegetales: <<Ser vivo sin movilidad>>

Escribir el código correspondiente. Haciendo uso de la herencia por prototipos, crear un objeto de tipo tomate al que se denomine tomate1 y hacer que se muestre por pantalla <<tomate1 tiene la propiedad movilidad: Ser vivo sin movilidad>>.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01150E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CALL JAVASCRIPT. DIFERENCIAS CON APPLY. CONSTRUCTORES CON HERENCIA EN CADENA (PROTOTYPE CHAIN). EJEMPLOS (CU01150E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº50 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CADENAS DE HERENCIA JAVASCRIPT

JavaScript permite la herencia a través de prototipos, pero todavía no hemos visto cómo podemos crear un objeto de un subtipo que pueda ser instanciado para que se inicialice portando información tanto del subtipo como del supertipo. Antes de abordar esta cuestión estudiaremos las funciones call y apply.

CALL JAVASCRIPT

La función call permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, así como los parámetros adicionales que sean necesarios.

La sintaxis más básica es la siguiente:

```
function unaFuncion (par1, par2, ..., parN) {
// código
... }
unaFuncion.call (objetoQueActuaráComoThis, par1, par2, ... parN);
```

Veamos un ejemplo básico para comprender lo que significa y permite esta función. Escribe el código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar() { alert ('Hola, soy ' + this.nombre); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar();
saludar.call(unProfesor);
unProfesor.saludar(); //ERROR - NO PERMITIDO
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado de ejecución esperado es: Hola, soy undefined. >>> Hola, soy Carlos. >>> (Y un error: la línea unProfesor.saludar() no se ejecuta).

Analicemos paso a paso lo que hace el código:

Se define el tipo de objeto Profesor con los atributos nombre y salarioBase. Se define la función saludar() que carece de propiedades y que cuando se invoca muestra por pantalla un mensaje.

Se crea un objeto de tipo Profesor cuya propiedad nombre vale “Carlos”.

Se llama a la función saludar() y como resultado se obtiene “Hola, soy undefined”. ¿Por qué? Porque el objeto this en este caso es la propia función saludar y dicha función no tiene definido atributo nombre, por tanto al tratar de mostrar this.nombre muestra ‘undefined’.

A continuación se llama a la función saludar indicando que el objeto unProfesor actuará como objeto this para la ejecución de la función. Como consecuencia, this.nombre vale “Carlos” y se muestra por pantalla el mensaje “Hola, soy Carlos”.

El quid de la cuestión está en que cualquier función puede invocarse como método de cualquier objeto a través del método predefinido JavaScript call. En el ejemplo que hemos usado podemos destacar algunas cuestiones:

a) saludar() es lo mismo que saludar.call(). Al indicar saludar.call() y no pasar como parámetro ningún objeto, es la propia función saludar quien actúa como this.

b) La función saludar podría ser usada por diversos tipos de objetos que tuvieran un atributo name. La función actuaría así como método compartido por numerosos tipos de objetos, pudiendo estar fuera de la definición de los mismos y fuera de la cadena de herencia de los mismos.

Veamos ahora la sintaxis usando parámetros. Escribe este código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar(nombrePersona, modoSaludo) {
alert ('Hola, soy ' + this.nombre + ' y saludo a ' +nombrePersona+ ' con ' + modoSaludo); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar.call();
saludar.call(unProfesor, 'Ernesto', 'afecto');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado esperado es: Hola, soy undefined y saludo a undefined con undefined. >>> Hola, soy Carlos y saludo a Ernesto con afecto.

En la invocación saludar.call(unProfesor, 'Ernesto', 'afecto'); el parámetro unProfesor indica quién va a actuar como objeto this en la ejecución de la función saludar, mientras que 'Ernesto' y 'afecto' son los parámetros que se le pasan a la función.

APPLY JAVASCRIPT

La función apply permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, de la misma forma que con la función call. La diferencia de apply con call está en que los parámetros se pasan con un array en vez de separados por comas. De este modo apply consta exactamente de dos parámetros: el objeto que actuará como this y un array.

La sintaxis más básica es la siguiente:

```
function unaFuncion (par1, par2, ..., parN) {
// código
... }
unaFuncion.apply (objetoQueActuaráComoThis, arrayDeElementos);
```

Donde arrayDeElementos es un array. El array se puede haber declarado previamente, o bien declararse en el mismo momento de llamada de la función escribiendo [elemento1, elemento2, ..., etc.].

Veamos un ejemplo básico para comprender lo que significa y permite esta función. Escribe el código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar(nombrePersona, modoSaludo) {
alert ('Hola, soy ' + this.nombre + ' y saludo a ' +nombrePersona+ ' con ' + modoSaludo); }
function ejemploObjetos() { var unProfesor = new Profesor('Carlos');
saludar.apply();
var unArray = ['Christian', 'odio'];
saludar.apply(unProfesor, ['Ernesto', 'afecto']);
saludar.apply(unProfesor, unArray);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El resultado de ejecución esperado es: Hola, soy undefined y saludo a undefined con undefined >>> Hola, soy Carlos y saludo a Ernesto con afecto >>> Hola, soy Carlos y saludo a Christian con odio.

Call y apply son dos funciones muy similares. ¿Cuándo usar una y cuándo usar otra? En muchos casos será indistinto usar una y otra. Sin embargo apply nos provee de la potencia de los arrays, lo que la hace interesante cuando por algún motivo necesitemos crear bucles que recorran los parámetros pasados en el array, o cuando simplemente no se conozca a priori el número de parámetros que deben pasarse porque se establezcan de forma dinámica.

CREAR HERENCIA CON JAVASCRIPT

Sabemos cómo hacer para que todos los objetos de un tipo hereden propiedades y métodos comunes de su prototipo, pero ¿cómo crear herencia entre dos tipos de objetos?

Partimos de un ejemplo: tenemos como subtipo ProfesorInterino con la propiedad mesesContrato y como supertipo Profesor con las propiedades institucion y salarioBase como muestra el siguiente código.

```
function Profesor (institucion) { this.institucion = institucion || 'Desconocida'; this.salarioBase = 1200; }

function ProfesorInterino(mesesContrato) { this.mesesContrato = mesesContrato || -1;}
```

Ahora nos planteamos que los Profesores Interinos son un tipo de profesor, y por tanto al crear un profesor interino queremos inicializarlo de modo que disponga de las propiedades y métodos de Profesor. Al crear un profesor interino queremos poder especificar el atributo institucion o, si no lo especificamos, poder acceder a la propiedad institucion obteniendo como valor 'Desconocida' ¿Cómo hacerlo?

Una primera aproximación puede ser esta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function Profesor (institucion) {
this.institucion = institucion || 'Desconocida';
this.salarioBase = 1200;
}

function ProfesorInterino(mesesContrato, institucion) {
Profesor.call(this, institucion);
this.mesesContrato = mesesContrato || -1;}

function ejemploObjetos() {
var unProfesorInterino = new ProfesorInterino(4, 'Universidad de Chapingo');
var msg = 'El objeto unProfesorInterino tiene ' + unProfesorInterino.mesesContrato + ' meses de contrato';
msg = msg + ' y pertenece a la institución '+unProfesorInterino.institucion;
alert(msg);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado de ejecución esperado es: El objeto unProfesorInterino tiene 4 meses de contrato y pertenece a la institución Universidad de Chapingo

Analicemos lo que ocurre. Hemos modificado la función ProfesorInterino para que además de las propiedades intrínsecas (mesesContrato) pueda recibir la institución, propiedad de los objetos tipo Profesor.

Con la invocación Profesor.call(this, institucion); estamos haciendo que se ejecute la función Profesor pasándole como this el objeto ProfesorInterino y como parámetros la institución. Al ejecutarse la función Profesor, el objeto ProfesorInterino pasa a tener todos los atributos y métodos de un Profesor ya que se ejecuta this.institucion = institucion || 'Desconocida'; y this.salarioBase = 1200.

De este modo, el objeto ProfesorInterino dispone de las propiedades y métodos propios de los objetos Profesor.

Pero, ¿realmente se comporta el objeto unProfesorInterino como los objetos de tipo Profesor? La respuesta es que no: a través de la llamada call hemos logrado que las propiedades y métodos declarados con this para Profesor estén disponibles para el profesor interino, pero ¿qué ocurre con las propiedades y métodos comunes de los objetos Profesor, es decir, aquellas propiedades y métodos que hayamos especificado en el prototipo de Profesor. Lo que ocurre es que no están disponibles para unProfesorInterino. Comprobémoslo. Escribe este código y comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function Profesor (institucion) { this.institucion = institucion || 'Desconocida';
this.salarioBase = 1200; }
Profesor.prototype.saludar = function() {alert('Hola trabajo en '+ this.institucion + ' y mi salario base es
'+this.salarioBase);}

function ProfesorInterino(mesesContrato, institucion) { Profesor.call(this, institucion);
this.mesesContrato = mesesContrato || -1;}

function ejemploObjetos() {
var unProfesor = new Profesor(); unProfesor.saludar();
var unProfesorInterino = new ProfesorInterino(4, 'Universidad de Chapingo');
var msg = 'El objeto unProfesorInterino tiene ' + unProfesorInterino.mesesContrato + ' meses de contrato';
msg = msg + ' y pertenece a la institución '+unProfesorInterino.institucion;
alert(msg);
unProfesorInterino.saludar();
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El resultado esperado es: Hola trabajo en Desconocida y mi salario base es 1200 >> El objeto unProfesorInterino tiene 4 meses de contrato y pertenece a la institución Universidad de Chapingo >> **error** (la línea unProfesorInterino.saludar(); no se ejecuta).

Lo que comprobamos es que los objetos de tipo Profesor conocen el método saludar (que es un método común a todos los objetos de tipo Profesor definido a través de su prototipo), pero sin

embargo los objetos de tipo ProfesorInterino no conocen el método saludar, lo que demuestra que no es están comportando realmente como Profesor.

Para que los objetos de tipo ProfesorInterino dispongan de todo lo que dispone Profesor nos falta hacer que el prototipo de todo objeto ProfesorInterino sea un objeto Profesor. De este modo, cuando se busque el método saludar se buscará en primer lugar como método intrínseco de ProfesorInterino, al no encontrarse se buscará en el prototipo, que al ser un objeto Profesor sí podrá responder cuando se invoque el método saludar.

Para ello añadiremos la línea: ProfesorInterino.prototype = new Profesor();

Con esta línea ya los objetos de tipo ProfesorInterino conocen las propiedades y métodos comunes de Profesor.

Con esta línea ya se ejecuta la invocación unProfesorInterino.saludar() dando como resultado que se muestre por pantalla <<Hola trabajo en Universidad de Chapingo y mi salario base es 1200>>.

En resumen, para implementar una herencia completa y poder crear instancias pasando parámetros para generar objetos que hereden propiedades y métodos de un supertipo usaremos la invocación call al supertipo y además estableceremos que el prototipo es un objeto del supertipo.

CONSTRUCTORES CON HERENCIA

Con las herramientas que conocemos ya somos capaces de implementar cadenas de herencia con constructores que reciban parámetros de los supertipos. Supongamos que tenemos 4 clases en cadena y cada clase tiene una propiedad intrínseca. En la generación de un objeto del tipo inferior con herencia, tendríamos que pasarle 4 parámetros e invocar con 3 parámetros a su ascendiente. Luego el ascendiente será invocado con 2 parámetros y finalmente habrá una invocación con un parámetro.

A su vez, estableceremos como prototype de cada tipo de objeto a una instancia de su ascendiente.

ALTERNATIVA AL USO DE CALL

Podemos obtener los mismos efectos que con call usando esta construcción:

```
function unaFuncion (par1, par2, ..., parN) {
this.fun = Superclase;
this.fun (param1, param2, ..., paramN);
... }
```

Es decir, el código <<A>>: Profesor.call(this, institucion);

Da lugar al mismo efecto que el código <<B>>: this.fun = Profesor; this.fun(institucion);

En el primer caso (A) se ejecuta la función Profesor pasando como objeto this al objeto que lo llama y esto da lugar a que el objeto incorpore las propiedades y métodos del objeto llamado.

En el segundo caso (B) se define Profesor como un método propio del objeto y a continuación con la invocación this.prop(instituticion) se ejecuta dicho método.

En principio call o apply resultan más eficientes, ya que nos ahorramos crear el método prop para posteriormente ejecutarlo, pero a efectos prácticos suele resultar indistinto y este tipo de invocaciones es frecuente encontrarlo cuando se revisa código JavaScript.

RESUMEN

La herencia en JavaScript no funciona como en otros lenguajes y en cierta medida más que de herencia propiamente dicha podríamos hablar de simulación de herencia. El siguiente cuadro resume las vías principales para generar herencia en JavaScript:

GENERADOR DE HERENCIA	DESCRIPCIÓN APRENDERAPROGRAMAR.COM
prototype.nombrePropiedadOMetodo	Define propiedades y métodos comunes que son compartidos por un tipo de objetos.
subtipo.prototype = new supertipo()	Crea herencia del supertipo pero no permite especificar los parámetros para los constructores de los supertipos.
nombreFuncion.call(objetoThis, par1, par2, ...) dentro de una función declarativa de tipo.	Crea herencia de propiedades y métodos declarados con this en los supertipos y permite inicializar objetos pasando parámetros para los subtipos y los supertipos, pero no da lugar a que se conozcan las propiedades y métodos compartidos del supertipo.
subtipo.prototype = new supertipo() + nombreFuncion.call(objetoThis, par1, par2, ...) dentro de una función declarativa de tipo	Permite simular una herencia completa y la instanciación de un objeto con todos los parámetros propios y de supertipos.

EJERCICIO 1

Crea un esquema de herencia que cumpla con estos requisitos:

a) Un Médico especialista tiene una especialidad y es un tipo de Médico.

b) Un Médico trabaja en un centro de trabajo y es un tipo de Persona.

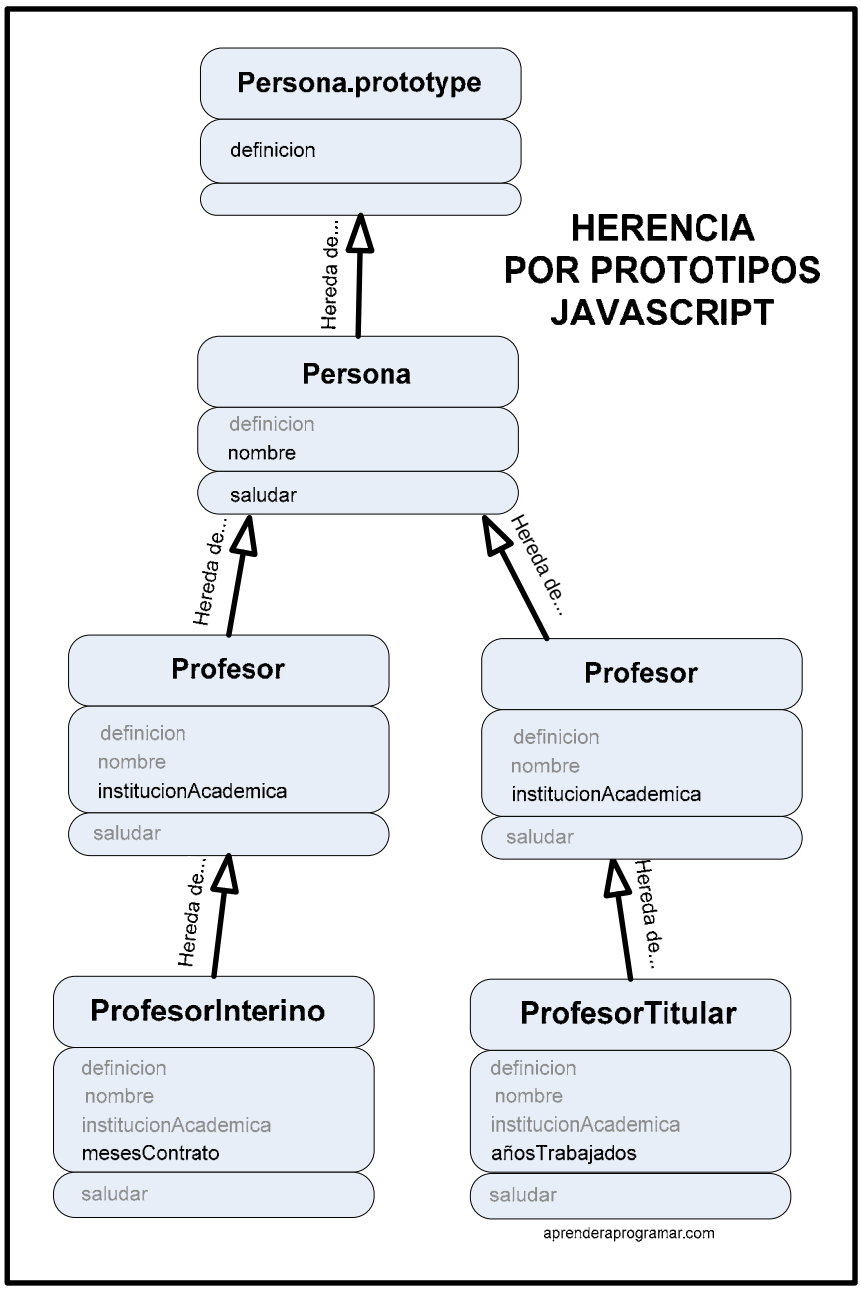
c) Una Persona tiene un nombre y una nacionalidad. Como método común a todas las personas tenemos mostrarNacionalidad, que muestra un mensaje informando de la nacionalidad.

Se desea crear un objeto de tipo MedicoEspecialista pasándole como parámetros para su creación además de sus propiedades intrínsecas las propiedades que hereda de sus supertipos y sobre este objeto invocar el método mostrarNacionalidad(), que deberá ser reconocido por herencia.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un código que represente el siguiente esquema de herencia permitiendo instanciar los subtipos pasándole los parámetros necesarios para inicializar las propiedades de los supertipos. Crea un objeto ProfesorTitular profesorTitular1 al que le pases como parámetros 8 (años trabajados), Universidad de León (institución académica), Juan (nombre), e invoca el método saludar sobre este objeto.



Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01151E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DEPURAR JAVASCRIPT. CONSOLE.LOG. DEBUG DE ERRORES CON FIREFOX (FIREBUG), CHROME, INTERNET EXPLORER. IDES. (CU01151E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº51 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

DEPURAR JAVASCRIPT

Si vienes siguiendo el curso de JavaScript y has ido realizando los ejercicios con seguridad te habrás enfrentado a problemas a la hora de ejecutar los scripts relacionados con errores en el código. Estos errores pueden dar lugar a que no se ejecute lo esperado, o simplemente que no se ejecute nada. La dificultad radica en que no tenemos ningún mensaje de aviso del error.

Por tratarse de un lenguaje interpretado y embebido JavaScript resulta un lenguaje relativamente difícil de depurar. No hay un método único o preferido por la comunidad de programadores, por ello hablaremos de alternativas o posibilidades para depurar JavaScript.

Agruparemos en primer lugar las alternativas en tres grandes grupos que vamos a citar superficialmente para luego centrarnos en una de estas alternativas:

a) **Herramientas de validación** del código (validadores) on-line o instaladas en un pc. Estas herramientas, en líneas generales, reciben un código JavaScript como entrada y devuelven una lista de los errores o problemas detectados en el código. Estos problemas pueden ser de sintaxis o de otro tipo. Como herramientas on-line podemos citar JSLint (http://www.jslint.com/) y JSHint (http://www.jshint.com/). Entre las herramientas para instalar en el computador de desarrollo tenemos Google Closure Linter (https://developers.google.com/closure/utilities/) y JavaScript Lint (http://www.javascriptlint.com/).

b) **IDEs para desarrollo web**. En general se trata de entornos de desarrollo que proveen algunas herramientas o extensiones capaces de detectar errores en el código ó al menos facilitar ayuda que debería dar lugar a que existan menos errores en el código. Son en general herramientas que proveen muchas funcionalidades y por tanto más pesadas que un simple editor de texto. Podemos citar dentro de este grupo Eclipse, IDE que permite la instalación de plugins entre los que cabe destacar JSDT (JavaScript Developer Tools), Komodo IDE, NetBeans, Microsoft Visual Studio con Visual Web Developer, WebStorm y Aptana Studio.

c) **Herramientas nativas** o que se integran en los navegadores web y que permiten el análisis del código y depuración de errores. Mozilla Firefox, es uno de los navegadores que ha sido tradicionalmente preferido por los programadores por las herramientas de análisis y depuración que provee y por atenerse a los estándares. De cara a la depuración de código JavaScript dispone de herramientas nativas (Developer tools, propia del explorador) como la consola y de diferentes plugins para desarrolladores entre los que destaca Firebug. El navegador Chrome de Google provee herramientas nativas en lo que se denomina Chrome Developer Tools. Internet Explorer de Microsoft ofrece también sus Developer Tools. En la mayor parte de los navegadores se accede a las developer tools pulsando F12.

¿QUÉ ALTERNATIVA ELEGIR?

Dado que existen numerosas alternativas la pregunta que nos planteamos es: ¿qué alternativa elegir? Para el desarrollo web profesional posiblemente debiéramos decantarnos por un IDE y conocer sus herramientas de depuración nativas o en forma de extensiones. Pero en este curso, donde nos centramos en comprender JavaScript y no en la productividad como programadores, vamos a recomendar que utilices las herramientas de desarrollador (developer tools) de tu navegador web, principalmente con el fin de tener ayuda a la hora de localizar errores en nuestro código JavaScript. Esta es una vía relativamente cómoda, rápida y sin necesidad de instalación de software adicional, para localizar errores en nuestro código.

¿Qué navegador ofrece mejores posibilidades de cara a la depuración? Preferimos no opinar al respecto. Firefox ha tenido siempre muy buena prensa entre los programadores e Internet Explorer al revés. Pero el mundo del desarrollo de aplicaciones web y apps para smartphones se mueve a gran velocidad y las opiniones que eran válidas hace unos años pueden no serlo en la actualidad.

Te recomendamos que optes por las herramientas de depuración de tu propio navegador web (el que vengas utilizando), y si consideras que no te responde adecuadamente, simplemente prueba otro.

UN CÓDIGO DE EJEMPLO

A modo de ejemplo consideraremos que hemos desarrollado un código JavaScript pero al tratar de ejecutarlo no hay respuesta alguna. Entendemos (suponemos) que hay un error y queremos que una herramienta de depuración nos facilite su localización.

El código que usaremos es este:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function saludar() { alert ('Hola, soy ' + this.nombre); }
function ejemploObjetos() {
var unObjeto = new Array();
unObjeto.saludar();
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

CACHÉ DE NAVEGADOR

Un problema que aparece en ocasiones es que el navegador guarda información en caché de modo que no se ejecuta el código que hemos modificado, sino código guardadado en la caché del navegador. Normalmente este problema se soluciona haciendo una recarga completa pulsando CTRL + F5 (teclas ctrl y F5 al mismo tiempo).

DESPLEGAR LA CONSOLA DEL NAVEGADOR

La mayoría de los navegadores permiten desplegar una consola o ventana auxiliar en la parte inferior del navegador, que sirve para fines de desarrollo de código y mostrar mensajes de error. Para activarla en tu navegador pulsa F12 (esto funciona en la mayor parte de los navegadores modernos, pero puede no estar disponible en algunos).

Una vez desplegada la consola, carga el archivo html con el código que contiene el error y pulsa en "Probar".

Por consola se muestra un mensaje como: TypeError: unObjeto.saludar is not a function ejemplo1.html:7

Este mensaje nos indica el tipo de error (se trata de invocar una función que no existe) y la línea donde se ha detectado el error dentro del archivo (línea 7).

El mensaje de error puede diferir según el navegador que estemos utilizando, por ejemplo otro navegador puede mostrar <<Uncaught TypeError: undefined is not a function ejemplo1.html:7>>

Este sencillo método para detectar y localizar errores te será muy útil para detectar y solucionar errores cuando estés creando código JavaScript.

Si tu navegador no te permite desplegar consola te recomendamos que lo actualices (puede que sea demasiado antiguo) o que te cambies de navegador.

CONSOLE.LOG

La consola tiene otra utilidad clave: permite mostrar mensajes informativos introducidos en el código pero no visibles como contenido de la página web.

Para mostrar mensajes a través de la consola usaremos la sintaxis: console.log(argumento), donde argumento es aquello que debe ser mostrado por consola (puede ser un texto, una variable, un objeto, etc.).

Por ejemplo:

```
var unObjeto = new Array();

console.log(unObjeto);
```

Nos devuelve a través de la consola (que tendremos que haber activado previamente pulsando F12) Array [ ] y la línea en que se genera el mensaje.

Console.log tiene un funcionamiento análogo a alert pero una ventaja importante respecto a esta función: por un lado, si olvidas en el código un console.log las repercusiones serán menores puesto que no será visible a menos que se active la consola, y los usuarios no navegan con la consola activada. Por otro lado, permite mostrar mensajes sin detener la ejecución del script, lo cual nos ahorra tiempo y facilita el trabajo.

FIREBUG: EXTENSIÓN DE FIREFOX

Debido a que es una herramienta con una larga tradición entre los desarrolladores web, merece la pena dedicarle unas líneas a la extensión FireBug de Mozilla Firefox.

Firebug es una extensión del navegador Firefox que nos permite editar webs y hojas de estilo, monitorizar tiempos de carga, depurar javascript y ver los errores y procesos de carga en la página además de explorar el DOM.

Firebug dispone de su propia consola donde igualmente podemos visualizar los mensajes de error. Si utilizas Firefox y estás interesado en conocer e instalar esta herramienta te recomendamos que leas el artículo Firebug, una extensión gratuita de Firefox para programadores y diseñadores web en la sección de Herramientas Informáticas de aprenderaprogramar.com.

RESUMEN

Existen múltiples vías para depurar código JavaScript y puedes optar por aquellas que mejor se adapten a tí. Como alternativa rápida y sencilla, te recomendamos que cuando estés probando código trabajes con la consola de tu navegador desplegada y que te valgas de console.log para mostrar mensajes que no detengan la ejecución de los scripts.

EJERCICIO

Ejecuta este script con la consola de tu navegador activada. ¿Qué mensajes aparecen en la consola antes y después de pulsar <<Probar>>? Corrige el error que contiene el código. ¿Qué mensajes aparecen ahora en la consola? ¿Qué interpretación podemos darle a los resultados que hemos obtenido?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() {
var unObjeto = new Array();
console.log('unObjeto es' + unObjeto);
console.log(unObjeto);
console.log('this es ' + this);
alert('Hello';
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01152E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

OPERADOR CONDICIONAL TERNARIO TIPO IF CON INTERROGACIÓN (JAVASCRIPT, JAVA, ETC.) Y DOS PUNTOS. EJEMPLOS (CU01152E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº52 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

OPERADOR CONDICIONAL TERNARIO

Para escribir condicionales la estructura básica de lenguajes como JavaScript, Java, PHP y otros es la sentencia if. Otro tipo de condicionales como el switch o el else if pueden ser reemplazados por if. Hay otra sintaxis de condicional que se denomina operador condicional ternario y que se escribe con expresiones que incluyen una interrogación y dos puntos como: a ? b : c;

La sintaxis y significado del operador condicional ternario (válida para diferentes lenguajes, como JavaScript, Java, PHP, etc.) es la siguiente:

expresiónConValorBooleano ? expresión1 : expresión2;

Interpretación: si la expresiónConValorBooleano es cierta se ejecuta la expresión1, y en caso contrario se ejecuta la expresión2.

Ejemplos. Supongamos que tenemos cuatro variables: A = 5, B = 3, C = -7 y D = 5

Expresión	Resultado
A==5 ? dispara(): espera();	Se ejecuta dispara()
A<B ? dispara(): espera();	Se ejecuta espera()
B<C ? dispara(): espera();	Se ejecuta espera()
A<B && B>C ? dispara(): espera();	Se ejecuta espera()
A<B && B>C || B==3 ? dispara(): espera();	Se ejecuta dispara()
A==5 ? A=20: A=1;	Si A valía 5 ahora vale 20, caso contrario ahora vale 1.
B = B==3 ? B*10: B*100;	Se trata de una asignación: si B valía 3, ahora B vale 10 veces lo que valía, caso contrario ahora B vale 100 veces lo que valía.

El operador ternario puede ser insertado en sentencias de ejecución donde no se permite la inserción de if, por ejemplo en una operación de asignación o en la variable de control de un bucle for. Por

ejemplo for (var i = A==5 ? k(): t(); i<8; i++) implica que el bucle comienza con i valiendo lo que retorne la función k() si A vale 5, o lo que retorne la función t si A no vale 5.

Usar el operador condicional ternario tiene ventajas e inconvenientes.

VENTAJAS E INCONVENIENTES DEL OPERADOR CONDICIONAL TERNARIO

En la siguiente tabla resumimos las ventajas e inconvenientes del operador condicional ternario comparándolo con el if tradicional.

Condicional ternario	If tradicional
Permite la escritura compacta, permitiendo ahorrar escritura de código.	Obliga a escribir más sentencias para conseguir el mismo resultado.
Resulta más difícil de leer, entender y depurar	Resulta más fácil de leer, entender y depurar
No todos los programadores lo usan, algunos ni siquiera lo conocen.	Todos los programadores lo usan y lo conocen.
Se admite en la sintaxis de los lenguajes en lugares donde no se admite la sentencia if	No es válido en ciertas ubicaciones donde sólo se admiten expresiones, pero puede hacerse la evaluación antes del punto donde sea necesario el condicional.
Criticado por algunos expertos, adorado por otros	Uso y aceptación cuasi-universal

EJEMPLOS DE USO EN JAVASCRIPT

Escribe este código, guárdalo como archivo html y comprueba los resultados de ejecución. Razona sobre los resultados obtenidos y trata de explicar paso a paso la lógica de todo lo que hace el código.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var A=5; var B=3; C=-7; D=5;
var msg = '';
A==5 ? msg=msg+'A es 5\n': msg=msg+'A no es 5\n';
A<B ? msg=msg+'A es menor que B\n': msg=msg+'B es mayor o igual que A\n';
B<C ? msg=msg+'B es menor que C\n': msg=msg+'C es mayor o igual que B\n';
A<B && B>C ? msg=msg+'Se cumple A<B y B>C\n': msg=msg+'No se cumple A<B y B>C\n';
A<B && B>C || B==3 ? msg=msg+'Se cumple la condición\n': msg=msg+'No se cumple la condición\n';
A==5 ? A=20: msg=msg+'A no es 5\n';
msg = msg + 'Si A valía 5 ahora A vale 20, si no sigue valiendo lo que valía\n';
msg = msg + 'Ahora A vale '+A+'\n';
```

```
A==5 ? k(): t();
B = B==3 ? B*10: B*100;
msg = msg + 'Después de ejecutar B = B==3 ? B*10: B*100; resulta que B vale ' + B + '\n';
for (var i = A==5 ? k(): t(); i<8; i++) { msg = msg + 'Bucle: '+i+' \n'; }
alert(msg);
}
function k() {alert('Ha Sido llamada la función k'); return 3;}
function t() {alert('Ha Sido llamada la función t'); return 5;}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado esperado es:

Ha sido llamada la función t >> Ha sido llamada la función t >>
A es 5
B es mayor o igual que A
C es mayor o igual que B
No se cumple A<B y B>C
Se cumple la condición
Si A valía 5 ahora A vale 20, si no sigue valiendo lo que valía
Ahora A vale 20
Después de ejecutar B = B==3 ? B*10: B*100; resulta que B vale 30
Bucle: 5
Bucle: 6
Bucle: 7

EJERCICIO

Ejecuta este script y responde a las siguientes preguntas (algunas de ellas corresponden a cuestiones que hemos visto a lo largo del curso):

a) ¿Qué significa el operador += que se emplea en el código?

b) ¿Por qué usamos [0] para establecer nodoBody?

c) ¿Qué ocurre si dejamos en blanco la respuesta cuando se pide un color? Razona por qué ocurre esto.

d) ¿Qué ocurre si escribimos pink cuando nos pide el color? Razona por qué ocurre esto.

e) Modifica el código para que usando el operador ternario, si el usuario no introduce como color red, yellow o blue aparezca el mensaje 'No eligió color o el color es no válido'

f) Reescribe el código inicial de partida sustituyendo el condicional ternario por if tradicionales.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var htmlADevolver = '';
var colorUsuario = prompt('Elija color red, yellow o blue');
htmlADevolver += colorUsuario ? '<h1 style="background-color:'+colorUsuario+
';"> Usted eligió '+colorUsuario+'</h1>' : '<h1>No eligió color</h1>';
var nodoBody = document.getElementsByTagName('body')[0];
nodoBody.innerHTML = nodoBody.innerHTML + htmlADevolver;
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01153E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FUNCIONES ARRAYS JAVASCRIPT. PUSH, SORT (ORDENAR NÚMEROS), CONCAT, JOIN, POP, SHIFT, SLICE, SPLICE. (CU01153E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº53 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

FUNCIONES PREDEFINIDAS PARA ARRAYS (ARREGLOS)

JavaScript proporciona una serie de funciones predefinidas para el manejo de arrays. Entre ellas podemos citar concat, join, split, pop, push, shift, unshift, reverse. Vamos a estudiarlas. Recordar también que para conocer el número de elementos en un array basta con invocar la propiedad length del array escribiendo nombreArray.length

En las siguientes tablas se resumen funciones que pueden ser útiles para trabajar con arrays en JavaScript. Algunas de ellas ya fueron nombradas cuando hablamos de funciones para cadenas de texto, ya que JavaScript asemeja las cadenas de texto como arrays de caracteres.

Supondremos que A1, A2, A3, A4 y A5 son arrays JavaScript.

FUNCIONES PARA AÑADIR ELEMENTOS O CONCATENAR ARRAYS

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com

concat(it1, it2, ... , itN)	Devuelve la concatenación de it1, it2, ..., itN con el array sobre el que se invoca. It1, it2, ..., itN pueden ser tipos primitivos u objetos.	var A3 = A1.concat(5, 9);
		var A4 = A1.concat(A2);
		var A5 = A1.concat([-4, 22, 11]);

push(x)	Añade x al final del array como nuevo (o nuevos) elemento, y devuelve la nueva longitud del array.	A1.push(55, 66); //Añade 55 y 66 al final
unshift(x)	Añade x al principio del array como nuevo (o nuevos) elementos.	A1.unshift([77, 88, 99]); //Ahora A1 tiene 3 elementos más, al principio
splice (ind, 0, it1, it2, ... , itN)	Modifica el array añadiendo los elementos it1, it2, ..., itN, que son insertados en la posición ind (desplazando a los existentes).	A1.splice(3, 0, 'xxx', 'yyy'); //Inserta xxx en posición 3, yyy en posición 4 y desplaza a los elementos existentes antes.
splice (ind, cuant, it1, it2, ... , itN)	Modifica el array eliminando cuant elementos e insertando it1, it2, ..., itN, desde el índice ind.	ejemplo.splice(3, 2, 'es', 'un', 'en'); //Se borran dos elementos y se insertan tres, con lo que la longitud del array es 1 más de la anterior.

FUNCIONES PARA EXTRAER ELEMENTOS O PARTES DE UN ARRAY CON ELIMINACIÓN

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
pop()	Elimina el último elemento del array y lo devuelve.	var ultimoElemento = A1.pop(); //Ahora A1 tiene 1 elemento menos
shift()	Elimina el primer elemento del array y lo devuelve.	var primerElemento = A1.shift(); //Ahora A1 tiene 1 elemento menos
splice (ind, cuant)	Modifica el array borrando cuant elementos a partir del índice ind.	A1.splice(3, 2);
splice (ind, cuant, it1, it2, ... , itN)	Modifica el array eliminando cuant elementos e insertando it1, it2, ..., itN, desde el índice ind.	ejemplo.splice(3, 2, 'es', 'un', 'en'); //Se borran dos elementos y se insertan tres, con lo que la longitud del array es 1 más de la anterior.
delete A[ind]	Elimina el elemento con índice ind del array A. El contenido a[ind] pasa a ser undefined.	delete A[7]; //Ahora A[7] contiene undefined

FUNCIONES PARA EXTRAER PARTES O ELEMENTOS DE UN ARRAY SIN ALTERARLO

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
slice (firstIn, lastOut)	Devuelve un array con los elementos extraídos entre los índices firstIn y lastOut-1. Es decir, el elemento en la posición firstIn se incluye y el elemento en la posición lastOut se excluye.	var result = [1, 2, 3, 4, 5].slice(1,4); //result contiene [2, 3, 4]
slice (firstIn)	Devuelve un array con los elementos extraídos entre el índice firstIn y el último elemento. Si se indica un valor negativo, se extrae un array con los firstIn últimos elementos.	var result = [1, 2, 3, 4, 5].slice(-2) //result contiene [4, 5]

FUNCIONES PARA RECUPERAR ÍNDICES DE POSICIONES

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
indexOf(x)	Busca x dentro del array y devuelve la posición de la primera ocurrencia.	var result = A1.indexOf(14);
lastIndexOf()	Busca x dentro del array empezando por el final y devuelve la posición de primera ocurrencia.	var result = A1.lastIndexOf(14);

FUNCIONES PARA TRANSFORMAR ARRAYS EN STRINGS O TIPOS PRIMITIVOS

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
join(separador)	Une los elementos de un array en una cadena de texto donde cada elemento está separado por 'separador'.	var frase = ['quiero', 'aprender'].join(' '); //frase es tipo String y contiene 'quiero aprender'
toString()	Une los elementos de un array en una cadena de texto donde cada elemento está separado por una coma.	var frase = ['quiero', 'aprender'].toString(); //frase es tipo String y contiene 'quiero,aprender'
valueOf()	Este método devuelve la representación como tipo primitivo de un objeto. En el caso de un array, hace lo mismo que el método toString().	alert (A1.valueOf()); //Mismo resultado que alert(A1);

FUNCIONES QUE PERMITEN ORDENAR O REORDENAR ARRAYS

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com
reverse()	Invierte el orden de los elementos en el array (el final pasa a ser el principio).	A1.reverse(); //Los elementos quedan ordenados al revés
sort()	Si no recibe parámetros, ordena los elementos del array **por orden alfabético** (que no coincide con el numérico), quedando el array modificado. Comentaremos esta función con más detenimiento.	var result = [2, 11, 111, 7].sort(); //result vale [11, 111, 2, 7] porque el orden es alfabético, no numérico.

LA FUNCIÓN SORT JAVASCRIPT. CÓMO ORDENAR NUMÉRICAMENTE

Hemos visto que [2, 11, 111, 7].sort() devuelve [11, 111, 2, 7] porque el array se ordena por orden alfabético y alfabéticamente cualquier número que contenga un 1 inicial se coloca antes que cualquier número que contenga un 2 inicial, independientemente de cuál de los dos números es mayor o menor.

Si el array contiene elementos undefined, estos se colocan al final del array.

Nos planteamos ahora, ¿cómo ordenar números de menor a mayor (o de mayor a menor)?

Para que la función sort ordene de forma distinta a la predeterminada se le debe pasar como parámetro una función de comparación. Por ejemplo:

[3, 5, 1, 7, 4].sort (funcionDeComparacion);

Donde funcionDeComparacion debe ser una función que reciba como parámetros dos elementos del tipo con el que trabaja el array y devuelva un número negativo si el primer elemento debe ordenarse antes que el segundo, cero si ambos elementos tienen igual orden, o un número positivo si el segundo elemento debe ordenarse antes que el primero.

Definir una función de comparación que permita ordenar un array numérico de menor a mayor es sencillo:

```
function funcionDeComparacion (elem1, elem2) {
return elem1 – elem2;
}
```

Esta función devuelve un número positivo si elem1 es mayor que elem2, con lo cual elem1 se colocará detrás. Si escribimos return elem2 – elem1 se ordenará de mayor a menor.

Escribe el siguiente código y comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var miArrayOriginal = [1, 4, 2, 9, 11, 44, 111, 7, 4];
var miArray = [1, 4, 2, 9, 11, 44, 111, 7, 4];
var msg ='El array original es: ' + miArrayOriginal +'\n';
msg = msg + 'Ordenado de mayor a menor es ' + miArray.sort(deMayorAMenor)+'\n';
msg = msg + 'Ordenado de menor a mayor es ' + miArray.sort(deMenorAMayor)+'\n';
alert(msg);
}
function deMenorAMayor(elem1, elem2) {return elem1-elem2;}
function deMayorAMenor(elem1, elem2) {return elem2-elem1;}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado esperado es:

El array original es: 1,4,2,9,11,44,111,7,4

Ordenado de mayor a menor es 111,44,11,9,7,4,4,2,1

Ordenado de menor a mayor es 1,2,4,4,7,9,11,44,111

EJERCICIO 1

Crea un script donde a partir del array [33, 2, 36, 55, 4, 1] se realicen los siguientes procesos:

a) Mostrar el array ordenado de menor a mayor y de mayor a menor usando la función sort y definiendo la función de ordenación de forma anónima, es decir, deberás invocar X.sort(function (...) { ... }) definiendo la función de forma anónima en vez de cómo función con nombre.

b) Crea una variable de nombre deMenorAMayor que contenga el resultado de ordenar de menor a mayor los elementos del array.

c) Crea una variable de nombre deMayorAMenor que contenga el resultado de ordenar de mayor a menor los elementos del array.

d) Muestra por pantalla el array original, la variable deMenorAMayor y la variable deMayorAMenor.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un script donde sea posible ordenar palabras por orden alfabético sin tener en cuenta la existencia de mayúsculas o minúsculas. Por ejemplo, declarar var miArray=['Moto', 'soto', 'Abaco', 'abeja', 'Sapo', 'nieve', 'Zumba, 'barco'] y tras ordenar obtener =['Abaco', 'abeja', 'barco', 'Moto', 'nieve', 'Sapo', 'soto', 'Zumba].

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01154E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EXPRESIONES REGULARES JAVASCRIPT. REGEX. NEW. CARÁCTER ESPECIAL. NÚMERO, LETRA, ESPACIO BLANCO. (CU01154E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº54 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

EXPRESIONES REGULARES JAVASCRIPT

En muchas ocasiones nos encontramos con la necesidad de hacer comprobaciones respecto a una cadena de texto. Por ejemplo comprobar si contiene letras mayúsculas, validar si es una dirección de correo electrónico correctamente formada… Estas comprobaciones también pueden servir para autocompletar cuando el usuario está escribiendo una entrada en un formulario.

Las comprobaciones sobre cadenas de texto se pueden hacer de muy distintas maneras y pueden tener muy distintas utilidades. En un nivel muy amplio, el estudio de cadenas, sus patrones, formas de construcción, etc. lo podríamos englobar dentro de la temática de "Lenguajes formales, expresiones regulares, autómatas y gramáticas".

Nosotros vamos simplemente a indicar las cuestiones más relevantes relacionadas con la posibilidad de uso de expresiones regulares dentro de JavaScript, sin entrar en su análisis desde el punto de vista formal o teórico.

Una expresión regular es un patrón: una expresión que define todas las cadenas o subcadenas que cumplen con ciertos criterios. Por ejemplo, un criterio puede ser "contener una a". Hemos de tener en cuenta que a efectos de análisis de cadenas de texto no es lo mismo a (minúscula) que A (mayúscula) ó á (con tilde). Es decir, una letra no es lo mismo que esa letra en mayúsculas o esa letra con tilde, diéresis o cualquier otro signo de puntuación. De hecho, quienes estén habituados a trabajar con códigos de caracteres como ASCII ó Unicode ya sabrán que una letra minúscula tiene distinto código numérico que esa misma letra mayúscula.

Una forma inicial de expresar un patrón podría ser la indicada en la siguiente tabla:

Patrón	Cadena a probar	¿Cumple el patrón?
Todas las cadenas que comienzan por una letra distinta de a y contienen al menos una a	Camión	Sí
	Abeto	No, dado que no contiene una a minúscula
	UEFA	No, ya que aunque contiene la A no cumple el requisito: contener una a minúscula.
	Abeja	Sí, ya que comienza con una letra A mayúscula y contiene una a minúscula
	BATHa	Sí, cumple los requisitos
	"apis mellifera sativa"	No, no cumple los requisitos*.
	\"apis mellifera sativa\"	Sí, cumple los requisitos*.

Hemos indicado "apis mellifera sativa" entrecomillada para indicar que la cadena está compuesta por varias palabras separadas por espacios en blanco. En este caso, la cadena contiene dos espacios en blanco. Hemos escrito \"apis mellifera sativa\" para representar una cadena donde las comillas forman parte de la cadena (pero no las barras inclinadas, que son el símbolo de escape que nos sirve para indicar que las comillas deben leerse literalmente). Esta cadena comienza con una comilla y contiene al menos una a minúscula, por lo que cumple con los requisitos.

Tener en cuenta que en una expresión regular los espacios en blanco cuentan. Por tanto no es lo mismo "apis mellifera sativa" que "apis mellifera sativa". Un solo espacio en blanco hace que dos cadenas se consideren distintas.

Las expresiones regulares son formas estandarizadas de expresar un patrón siguiendo ciertas reglas para definirlo. A un computador no podemos indicarle con una frase cuál es el patrón que queremos comprobar, tendremos que indicárselo con una sintaxis normalizada.

Es frecuente encontrar código donde se hace alusión a los siguientes términos:

regex: abreviatura de regular expression (expresión regular)

pattern: patrón

match: coincidencia (cadena que contiene el patrón).

Una expresión regular como “Todas las cadenas que comienzan por una letra distinta de a y contienen al menos una a” es relativamente compleja. Las expresiones regulares más sencillas son las que hacen alusión a si una cadena contiene un patrón. Por ejemplo: ¿Contiene una cadena la subcadena rio? Serían cadenas que contienen la subcadena rio y hacen match para la expresión regular indicada: “armariote”, “canario”, “rioboo”, “ariofonte”, etc. En cambio no contienen la subcadena y no hacen match cadenas como “CABRIOLET” (por estar en mayúsculas), “rico”, “dañino” ni “río”, en este último caso por llevar tilde.

EL OBJETO PREDEFINIDO REGEX

Para crear expresiones regulares JavaScript provee de una serie de símbolos especiales que permiten definir los patrones, como veremos más adelante.

Para introducir el concepto de símbolo especial y expresión regular vamos a usar primeramente sólo dos caracteres especiales: el carácter “punto” . , que es un simple punto, y el carácter ? que sirve para indicar que el carácter a continuación del símbolo es opcional, es decir, que puede aparecer 0 ó 1 vez.

En JavaScript el uso de expresiones regulares se basa en el uso de un objeto predefinido (existente) del lenguaje que podemos usar para construir nuestros scripts: el objeto RegExp. Este es uno de los objetos predefinidos de JavaScript, ya que existen otros (como Math, Date, etc.). RegExp nos provee de métodos útiles para trabajar con expresiones regulares.

JavaScript permite crear expresiones regulares de dos maneras:

a) En forma de literal: var miExpresionRegular = /as?.a/ representa a todas las cadenas que contienen una subcadena con la primera letra de la subcadena una a, seguida de una s, opcionalmente seguida de cualquier letra, y seguida de una a.

b) Instanciando el objeto RegExp: var miExpresionRegular = new RegExp("as?.a") representa lo mismo.

Una cadena como “casamentero” contiene el patrón: contiene una a, seguida de s, el carácter opcional no está presente, y seguida una a. c-**a-s-a**-m-e-n-t-e-r-o

Una cadena como “castaño” contiene el patrón: c-**a-s-t-a**-ñ-o contiene el patrón, siendo el carácter opcional la letra t.

También contienen el patrón asa, casta, masa, castañuela, casiarina, kaspamina ó asma.

Cadenas como “sabina”, “casualidad” ó “as” no contienen el patrón.

Escribe este código, guárdalo como archivo html y comprueba sus resultados. Observa cómo el método test aplicado a un objeto de tipo RegExp pasándole una cadena devuelve true si la cadena contiene el patrón definido por la expresión regular o false si la cadena no contiene el patrón.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var miExpReg = /as?.a/
var cadena = 'casamentero';
var msg = 'Patrón: as?.a \n';
msg = msg + '¿Contiene casamentero el patrón? : '+ miExpReg.test(cadena) +'\n';
msg = msg + '¿Contiene castaño el patrón? : '+ miExpReg.test('castaño') +'\n';
msg = msg + '¿Contiene sabina el patrón? : '+ miExpReg.test('sabina') +'\n';
msg = msg + '¿Contiene asa el patrón? : '+ miExpReg.test('asa') +'\n';
msg = msg + '¿Contiene as el patrón? : '+ miExpReg.test('as') +'\n';
alert(msg);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado esperado es: Patrón: as?.a >> ¿Contiene casamentero el patrón? : true

¿Contiene castaño el patrón? : true >> ¿Contiene sabina el patrón? : false

¿Contiene asa el patrón? : true >> ¿Contiene as el patrón? : false

CARACTERES ESPECIALES EN EXPRESIONES REGULARES

Los principales caracteres especiales para construir expresiones regulares JavaScript están reflejados en la siguiente tabla. Los paréntesis tienen un significado especial que veremos en la siguiente entrega.

Carácter	Significado	Ejemplo aprenderaprogramar.com
****	Backslash o carácter de escape. Sirve para anular un carácter especial y hacer que se considere como si fuera un carácter normal.	Ver más abajo
a	Donde a es una letra cualquiera. Indica que el patrón incluye una a en el orden especificado.	barco Cadenas que tienen a 'barco' como subcadena (incluido barco mismo)
{n}	El carácter anterior aparece exactamente n veces, siendo n un entero positivo.	ca{3}t{3}e Hace match si una subcadena es caaattte.
{n,}	El carácter anterior aparece n o más veces, siendo n un entero positivo.	ca{3,}te Hace match con caaate, caaaate, etc. pero no con cate
{n,m}	El carácter anterior aparece un mínimo de n veces y un máximo de m veces, siendo n y m enteros positivos.	ca{2,5}te Hace match con caate, caaate, caaaate y caaaaate
^	El símbolo ^ (denominado "exponente") indica comienzo de la cadena por ese símbolo.	/^a/ Cadenas que empiezan por a minúscula.
.	El símbolo punto indica existencia de cualquier carácter.	ca.e Cadenas que contienen subcadenas con caxe donde x es cualquier carácter.
$	El símbolo dólar indica que la letra anterior ha de ser obligatoriamente última letra de la cadena	^a..e$ Cadenas que empiezan por a y terminan con e, y que contienen exactamente cuatro caracteres (la a inicial, dos intermedios, más la e final)
*****	El símbolo asterisco indica que la subcadena contiene el símbolo al que precede cero o más veces.	^at*e$ Cadenas que empiezan por a, terminan por e, y entre ambos tienen cero, una o muchas t´s (pero no ningún otro carácter).
{0,}	Equivalente al símbolo asterisco	^at{0,}e$ Igual que el anterior
+	El símbolo más indica que la subcadena contiene el símbolo al que precede una o más veces.	^at+e$ Cadenas que empiezan por a, terminan por e, y entre ambos tienen una o muchas t´s (pero no ningún otro carácter).
{1,}	Equivalente al símbolo +	^at{1,}e$
?	El símbolo interrogación indica opcionalidad: el carácter que lo precede puede aparecer 0 ó 1 vez en la subcadena.	^at?e$ Cadenas que empiezan por a, terminan por e, y entre ambos tienen una ninguna t.
{0,1}	Equivalente al símbolo ?, es decir, el carácter anterior puede aparecer 0 ó 1 vez.	^at{0,1}e$
a|b	El símbolo or genera un match si se encuentra a ó b siendo a ó b dos caracteres cualesquiera que se indiquen.	^a(t|g)e$ Cadenas que empiezan por a, terminan por e, y entre ambos hay una t ó una g.

Carácter	Significado	Ejemplo aprenderaprogramar.com
sub1(?=sub2)	Hace match sólo si existe una subcadena donde la subcadena sub1 está antes de la subcadena sub2	mahatma(?= ghandi) Hace match con mahatma ghandi pero no con mahatma prasha
sub1(?!sub2)	Hace match en todas las subcadenas donde la subcadena sub2 no está después de la subcadena sub1	mahatma(?! ghandi) Hace match con mahatma para, mahatma gon, etc. pero no con mahatma ghandi
[abc]	Hace match con cualquiera de los caracteres indicados dentro de los corchetes (conjunto de caracteres)	^[aeiou] Cadenas que empiezan por una vocal.
[a-z]	Rango de caracteres. Hace match con cualquier carácter comprendido entre el inicial y el final, en orden alfabético. Se pueden definir varios conjuntos uno detrás de otro, por ejemplo: [A-Za-z] indica “letra mayúscula o minúscula”.	^[c-k] Cadenas que empiezan por c, d, e, f, g, h, i, j ó k.
[^abc] ó [^a-z]	Complementario o negado de un conjunto de caracteres. Hace match con cualquier carácter distinto de los definidos en el conjunto de caracteres	[^au]$ Hace match con cualquier cadena que no termina en a ni en u
\d	Hace match con cualquier número entre 0 y 9	^\d*\d$ Hace match si la cadena es un número seguido del símbolo * y seguido de otro número. * no es aquí especial por estar precedido del escape \.
[0-9]	Equivalente a \d	^[0-9]* [0-9]$
\D	Hace match con cualquier carácter que no sea un dígito. Equivale a [^0-9]	^ [^0-9] Cadena que no empieza por un número.
\s	Hace match con un espacio en blanco (incluye tabuladores y saltos de línea entre otros).	\s\s Hace match si encuentra dos espacios en blanco
\S	Hace match con cualquier carácter que no sea espacio en blanco, tabulador, salto de línea...	[\s\S] Hace match con cualquier carácter
\w	Hace match con cualquier letra mayúscula, minúscula, número o guión bajo. Equivale a [A-Za-z0-9_]. Tener en cuenta que las letras con tilde quedan fuera y habría que añadirlas si queremos.	\w{3} Hace match si existe una subcadena con 3 caracteres válidos seguidos.
\W	Hace match con cualquier carácter que no sea letra mayúscula, minúscula, número o guión bajo. Equivale a [^A-Za-z0-9_]	\W{2,3} Hace match si existe una subcadena con 2 ó 3 caracteres seguidos que no son letras ni números.
Otros	Existen otros símbolos especiales y combinaciones de símbolos especiales para lograr determinados fines.	

EJERCICIO

Dada la expresión regular de JavaScript /^[A-C]\w+\ses\s\w+/ indicar cuáles de las siguientes cadenas hacen match con la expresión regular, y en caso de hacer match, qué parte o partes son las que hacen match:

a) Juan es guapo

b) Adriano no es feo

c) Adriano deja de ser guapo

d) Adriano ya es guapo

e) No es ahora

f) Ahora es no

g) Adriano es guapo

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01155E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

PARÉNTESIS EN EXPRESIONES REGULARES JAVASCRIPT. FLAGS. MÉTODOS EXEC, TEST, MATCH, SEARCH, REPLACE, SPLIT. EJEMPLOS (CU01155E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº55 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

PARÉNTESIS Y FLAGS EN EXPRESIONES REGULARES

Las expresiones regulares JavaScript son una potente herramienta que nos permitirá resolver problemas diversos. Ya conocemos aspectos básicos como la instanciación de objetos RegExp y los caracteres especiales disponibles. Vamos a ampliar conocimientos sobre expresiones regulares.

EXPRESIONES REGULARES CON FLAGS

JavaScript permite especificar modos de búsqueda de expresiones regulares a través de lo que denomina flags (banderas o indicadores).

Los flags se indican en el momento de definir la expresión regular:

a) Si se define en forma de literal: var miExpresionRegular = /as?.a/**flagsAIncluir**

b) Instanciando el objeto RegExp: var miExpresionRegular = new RegExp("as?.a", " **flagsAIncluir**")

En la siguiente tabla se indican los principales flags, cómo usarlos y ejemplos.

Flag	Significado	Ejemplo sintaxis
g	Búsqueda global de todas las coincidencias (no se detiene al encontrar la primera coincidencia)	/as?.a/**g** new RegExp("as?.a", "**g**")
i	No diferenciar entre mayúsculas y minúsculas	/as?.a/**i** new RegExp("as?.a", "**i**")
m	Búsqueda multilínea. Para los indicadores de comienzo de cadena ^ y fin de cadena $, si se aplica esta bandera, se tienen en cuenta matches en todos los principios y finales de línea, no sólo al principio y final de cadena.	/as?.a/**m** new RegExp("as?.a", "**m**")
Combinaciones	Se pueden especificar varios flags	var miExpReg = /^i/**mi** Indica que hará match con cualquier comienzo de línea por una i minúscula ó I mayúscula (búsqueda multilínea y sin diferenciar mayúsculas y minúsculas).

MÉTODOS DE LOS OBJETOS TIPO EXPRESIÓN REGULAR

Los objetos de tipo expresión regular proveen de una serie de métodos útiles. En la siguiente tabla vemos cuáles son, su significado y un ejemplo de uso.

Método	Significado	Ejemplo de uso

exec	Ejecuta una búsqueda del patrón y devuelve un array cuyo índice 0 contiene el valor encontrado. Si está activado el flag g, la repetición de la búsqueda devuelve la siguiente coincidencia.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = '';
		while ((matches = miExpReg.exec(txt)) !== null) {
		msg = msg + 'Encontrado: '+matches[0]+'\n'; }
		alert(msg);

test	Comprueba si se verifica el patrón y devuelve true o false	var miExpReg = /p.sto/g var txt = 'el pasto es pisto pero no pesto'; msg = ''; alert('Hay match: '+ miExpReg.test(txt));

MÉTODOS DE LOS OBJETOS STRING RELACIONADOS CON EXPRESIONES REGULARES

Los objetos de tipo String proveen de una serie de métodos útiles. En la siguiente tabla vemos cuáles son, su significado y un ejemplo de uso.

Método	Signficado	Ejemplo de uso

match	Devuelve un array con las coincidencias encontradas, o null si no hay coincidencias.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var matches = txt.match(miExpReg);
		if (matches != null) {
		for (var i=0; i<matches.length; i++){
		msg = msg + 'Encontrado: '+matches[i]+'\n'; }
		} else {msg = 'No se encontraron coincidencias';}
		alert(msg);

search	Devuelve la posición de comienzo de la primera coincidencia dentro del string, o -1 si no hay coincidencia. Recordar que la primera posición posible es cero.	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var posicion = txt.search(miExpReg);
		if (posicion != -1) {
		msg = msg + 'Encontrado patrón en posición: '+posicion+'\n';
		} else {msg = 'No se encontraron coincidencias';}
		alert(msg);

replace	Devuelve un nuevo String (sin modificar el original) donde se reemplaza una o varias coincidencias por lo especificado (una cadena o una función que devuelve la cadena).	var miExpReg = /p.sto/g
		var txt = 'el pasto es pisto pero no pesto';
		msg = ''; var nuevoTxt = txt.replace(miExpReg, 'coco');
		msg = 'txt vale ' + txt + ' y nuevoTxt vale ' +nuevoTxt;
		alert(msg);

Método	Signficado	Ejemplo de uso

split	Devuelve un array con las subcadenas resultantes de dividir la cadena original en subcadenas delimitadas por el carácter separador especificado (que queda excluido). Si se indican unas comillas vacías se extraen todos los caracteres a un array.	var miExpReg = /\d/g
		var txt = 'un 3 de bastos gana a un 5 de copas pero no a un 7 de oros';
		msg = ''; var splits = txt.split(miExpReg);
		msg = 'splits contiene ' + splits;
		alert(msg);

USAR PARÉNTESIS PARA AGRUPAR FRAGMENTOS

Los paréntesis sirven para agrupar varios caracteres y afectarlos por un carácter especial.

Por ejemplo var miExpReg = /(p.s)?to/g

var txt = 'el pasto es pisto y eso es todo';

var resultado = txt.match(miExpReg);

Hace que resultado contenga ['pasto ', 'pisto ', 'to '] ya que hay un fragmento de la expresión que se ha agrupado e indicado que es opcional.

Los paréntesis nos permiten no tener que repetir. Por ejemplo en vez de /estereotipo|estereoforma/ podemos escribir /estereo(tipo|forma).

USAR PARÉNTESIS PARA IDENTIFICAR SUBEXPRESIONES CON MATCH

Los paréntesis generan otro efecto. Cuando se invoca la función match buscando la primera coincidencia (es decir, búsqueda no generalizada sin el flag g) y existen subexpresiones entre paréntesis, el array generado guarda en su índice cero la primera coincidencia mientras que en los sucesivos índices guarda las coincidencias de las subexpresiones.

Cuando se escribe una expresión regular, los elementos que se incluyen entre paréntesis pueden ser recuperados por separado invocándolos luego como $1, $2, $3…. Estas variables persisten normalmente hasta que se invoca una nueva expresión regular.

Veámoslo con un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var expReg = /(\w+)\s(\w+)\s(\w+)/;
```

```
var txt = "Era una noche negra y sola";
var newTxt = txt.match(expReg);
console.log(newTxt); //Activar la consola para que se visualice
var cambiada = newTxt[0].replace(newTxt[3], 'madrugada');
console.log(cambiada); //Activar la consola para que se visualice
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es:

Array ["Era una noche", "Era", "una", "noche"]

"Era una madrugada"

Analicemos lo que hace el código: en el array newTxt queda almacenado en su posición cero la coincidencia global con la expresión regular. En su índice uno, queda almacenado el match de la primera subexpresión entre paréntesis, es decir, la primera palabra (sin tildes). En su índice dos, la segunda palabra, y en su índice 3, la tercera palabra. A continuación creamos una cadena formada por las tres palabras con la tercera de ellas reemplazada por 'madrugada'.

USAR PARÉNTESIS PARA IDENTIFICAR SUBEXPRESIONES CON REPLACE

Dentro de una sentencia replace, los matches de subexpresiones entre paréntesis pueden ser invocadas usando unas variables temporales especiales que se numeran como $1, $2, $3... etc. de acuerdo con la cantidad de subexpresiones entre paréntesis que existan. Este ejemplo nos muestra cómo se pueden usar:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var expReg = /(\w+)\s(\w+)\s(\w+)/;
var txt = "Alberto Flores Rubalcaba: tiene calificación de 10";
var cambiada = txt.replace(expReg, '$3 $2, $1');
console.log(cambiada); //Activar la consola para que se visualice
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

En este ejemplo el match de la expresión regular es <<Alberto Flores Rubalcaba>>, y cada palabra es una subexpresión por encontrarse entre paréntesis en la expresión regular. En el replace podemos identificar cada subexpresión como $1 (primera subexpresión), $2 (segunda) y $3 (tercera).

En el replace indicamos que la coincidencia de la expresión regular sea sustituida por la cadena formada por la tercera palabra seguida de la segunda, una coma y la primera palabra.

De este modo el resultado esperado es que se muestre por consola: "Rubalcaba Flores, Alberto: tiene calificación de 10".

EJERCICIO

Crea un documento HTML (página web) donde exista un formulario que se envíe por el método GET. Se pedirá al usuario que introduzca nombre, apellidos y correo electrónico. Define dentro de la etiqueta form que cuando se produzca el evento onsubmit (pulsación del botón de envío del formulario) se ejecute una función a la que denominaremos validacionConExpReg que no recibe parámetros.

La función validar debe realizar estas tareas y comprobaciones **utilizando expresiones regulares**:

a) Comprobar que el nombre contiene al menos tres letras. Si no es así, debe aparecer un mensaje por pantalla indicando que el texto no cumple tener al menos tres letras. Por ejemplo si se trata de enviar Ka como nombre debe aparecer el mensaje: “El nombre no cumple tener al menos tres letras”.

b) Comprobar que el correo electrónico contiene el carácter @ (arroba) y el carácter . (punto). De no ser así, deberá aparecer un mensaje indicando que al correo electrónico le falta uno o ambos caracteres. Por ejemplo si se trata de enviar pacogmail.com deberá aparecer el mensaje: “Falta el símbolo @ en el correo electrónico”.

c) Antes de enviarse los datos del formulario a la página de destino, todas las letras del correo electrónico deben transformarse a minúsculas. Por ejemplo si se ha escrito PACO@GMAIL.COM debe enviarse paco@gmail.com

d) Antes de enviarse los datos del formulario a la página de destino, si el correo electrónico contiene la subcadena “ at “ debe reemplazarse por el símbolo @. Por ejemplo si se ha escrito paco at gmail.com debe enviarse paco@gmail.com

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01156E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CONVERTIR STRING A NUMBER JAVASCRIPT. REDONDEAR. TOFIXED, ISNAN, TOPRECISION, VALUEOF. PARSEINT Y PARSEFLOAT (CU01156E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº56 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

FUNCIONES PARA TRABAJAR CON NÚMEROS

JavaScript proporciona una serie de funciones predefinidas y palabras clave o constantes para trabajar con números. Entre ellas podemos citar isNan, Number(n), toFixed(n), toExponential(n), toPrecision(n), valueOf(), toString().

El trabajo con números en JavaScript se puede abordar desde distintas perspectivas.

En primer lugar, existen dos variables predefinidas globales que ya hemos mencionado:

a) NaN (Not-a-Number) que se usa para representar un valor numérico no válido (p.ej. 0/0).

b) Infinity que se usa para representar un valor numérico positivo infinito, o su variante -Infinity, que representa un valor numérico negativo infinito.

El valor NaN es un valor peculiar en JavaScript desde el momento en que no permite su comparación con el operador ==. Por ejemplo, var x = NaN; var y = NaN; alert (x==y); devuelve false a pesar de que ambas variables contienen Nan. Para evaluar si una variable contiene Nan ha de usarse la expresión x != x, que devolverá true únicamente si x contiene NaN. Dado que la aplicación de estos operadores es un tanto confusa con variables que pueden contener NaN, se recomienda usar la función isNaN que comentaremos un poco más adelante, y que permite evaluar si un valor es NaN.

Una expresión como 1/Infinity se evalúa a cero.

PROPIEDADES DEL OBJETO NUMBER

JavaScript, al igual que dispone de un objeto predefinido RegExp para trabajar con expresiones regulares, dispone de un objeto predefinido denominado Number que permite trabajar con números. Este objeto predefinido tiene propiedades y métodos. Entre sus propiedades podemos citar las siguientes:

PROPIEDAD	REPRESENTA	EJEMPLOS aprenderaprogramar.com
POSITIVE_INFINITY	Infinito positivo (igual que Infinity)	var num = Number.POSITIVE_INFINITY; //num toma valor Infinity
NEGATIVE_INFINITY	Infinito negativo (igual que -Infinity)	var num = Number.NEGATIVE_INFINITY; //num toma valor -Infinity

MAX_VALUE	Representa el valor numérico más grande con el que puede trabajar JavaScript	var num = Number.MAX_VALUE; // num vale 1.7976931348623157e+308 u otra cifra igualmente gigantesca, dependiendo del sistema
MIN_VALUE	Representa el valor numérico más pequeño con el que puede trabajar JavaScript.	var num = Number.MIN_VALUE; // num vale num vale 5e-324 u otra cifra también aproximadamente cero, dependiendo del sistema
NaN	Representa el valor NaN	var num = Number.NaN; // num vale NaN

Las propiedades de Number son de sólo lectura, es decir, no podemos establecer valores distintos de los predefinidos para ellas.

FUNCIONES DEL OBJETO NUMBER

FUNCIÓN	UTILIDAD	EJEMPLOS aprenderaprogramar.com

toFixed(n)	Devuelve un String con el número o variable sobre el que se invoca el método con tantos decimales como indique el parámetro n. Si n es cero o vacío, redondea al entero más próximo. Si es x.50 devuelve el entero inmediato superior si x es positivo o inferior si x es negativo. No genera notación exponencial.	var num = 3.1416;
		alert('num vale ' + num.toFixed(2));
		// num vale "3.14 "

isNaN(x)	Evalúa a true si x es un valor NaN o a false en caso contrario.	alert('¿0/0 vale NaN?: ' + isNaN(0/0)); // true
toPrecision(n)	Devuelve un String con el número o variable sobre el que se invoca el método con un número de dígitos significativos especificado por el número de decimales n. Si n es vacío devuelve el número sin modificar. No admite n cero. Si el número es demasiado grande para representar su parte entera, genera notación exponencial.	var num = (3.1416).toPrecision(2); // num vale 3.1, hay dos dígitos significativos.
valueOf(x)	Este método devuelve la representación como tipo primitivo de un objeto. En el caso de un Number, hace lo mismo que el método toString().	alert (A1.valueOf()); //Mismo resultado que alert(A1);
toString()	Devuelve la representación como String de un objeto Number.	alert (A1.toString()); //Mismo resultado que alert(A1);

DIFERENCIAR EL OBJETO NUMBER DE OBJETOS NUMBER

Debemos diferenciar entre el objeto predefinido Number, y los objetos Number que podamos crear nosotros como programadores (que también dispondrán de métodos heredados del objeto Number). Para crear objetos Number usamos la sintaxis var num = new Number(valorNumerico);

NUMBER COMO FUNCIÓN PARA CONVERTIR OTROS TIPOS DE DATOS

Dado una variable de otro tipo a la que denominaremos unaVarOtroTipo, podemos usar Number como función para realizar la conversión de dicha variable a un valor numérico usando esta sintaxis:

var valorNumerico = Number(unaVarOtroTipo);

Donde unaVarOtroTipo puede ser un String, un objeto Date, o cualquier otro objeto del que se pueda realizar una representación numérica.

FUNCIONES GLOBALES PARSEINT Y PARSEFLOAT

Existen dos funciones globales útiles para convertir cadenas en valores numéricos: parseInt y parseFloat.

La sintaxis básica de parseFloat es la siguiente:

```
var numerica = parseFloat(cadena);
```

La función parseFloat toma la cadena y trata de retornar un valor numérico decimal. Si la conversión no se puede completar por aparecer algún carácter extraño, devuelve el valor numérico que pueda extraer hasta la aparición de dicho valor. Si la cadena no empieza con un valor válido para un número (es decir, un +, un – ó un número, un signo . ó una letra e indicadora de exponente) la función devuelve NaN.

Veámoslo con un ejemplo:

```
var cadena1 = '82.35 % de aprobados';
var cadena2 = '% de aprobados: 82.35';
alert('cadena1 a numero: ' + parseFloat(cadena1) + ' ; cadena2 a numero: ' + parseFloat(cadena2));
```

Este ejemplo da lugar a que se muestre por pantalla:

cadena1 a numero: 82.35 ; cadena2 a numero: NaN

En el primer caso, extrae el valor numérico hasta allí donde le es posible. En el segundo caso, al no comenzar la cadena con algo que pueda convertirse a numérico, devuelve NaN.

La sintaxis básica de parseInt es la siguiente:

```
var numericaInt = parseInt(cadena);
```

Hay otra forma de invocación consistente en pasar dos parámetros: parseInt (cadena, raíz), donde raíz es un parámetro opcional que por defecto es 10 e indica la base numérica empleada. Por defecto es la decimal (10), pero podría ser 8 (valor numérico octal) ó 16 (valor numérico hexadecimal).

El funcionamiento de parseInt es similar al de parseFloat con la diferencia de que parseInt extrae únicamente aquella parte de la cadena que pueda reconocerse como un entero. Así parseInt('15.67 metros') devuelve 15, parseInt ('.1 metros') devuelve NaN (no admite que un entero comience con un punto), y parseInt ('0.1 metros') devuelve 0.

EJERCICIO

Crea un script donde:

a) Se pida al usuario que introduzca un número superior a 10000000 y se muestre por pantalla ese número con tres dígitos significativos. Si el número introducido no cumple la condición, se volverá a solicitar que se introduzca.

b) Se pida al usuario que introzca un número con 4 decimales y se muestre por pantalla ese número redondeado a dos decimales. Si el número introducido no cumple la condición, se volverá a solicitar que se introduzca (para ello habrá que analizar que la cadena introducida cuente con cuatro dígitos después del carácter de punto decimal).

c) Se pida al usuario que introduzca una frase que comience por un número que puede ser entero o decimal. Para dicha frase, se devolverá el número entero que se pueda extraer, y el número decimal que se pueda extraer. Deberá analizarse la entrada con una expresión regular, de modo que si no cumple la condición, se vuelva a solicitar la introducción de la frase.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01157E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EVENTOS JAVASCRIPT. TIPOS. PROPAGACIÓN. MODELOS. MANEJADOR O EVENT HANDLER. CONFIRM. EJEMPLO. (CU01157E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº57 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CONCEPTO DE EVENTO JAVASCRIPT

Un evento es algo que ocurre y puede ser detectado. Esta definición de partida es quizás demasiado simple. Podríamos decir que si un usuario hace click sobre un botón en una página web se produce un evento (el usuario ha hecho click), pero también que si el usuario estornuda se produce un evento, lo cual realmente no entra dentro del ámbito de la programación JavaScript, por lo que hemos de acotar esta definición.

JavaScript es un lenguaje que puede realizar tareas propias de la programación tradicional como ejecuciones secuenciales de instrucciones o bucles iterativos, pero está especialmente preparado para ser un lenguaje de respuesta dinámica a los eventos que genera el usuario. Por ello muchas veces se alude a JavaScript como un lenguaje dirigido por eventos o como un tipo de programación basada en eventos. Según este modelo de programación, la aplicación web está "a la espera" de que el usuario haga algo (pulsar un botón, un link, cerrar una ventana, etc.) y es cuando existe una acción del usuario cuando se ejecuta un fragmento de código. Esto permite que las aplicaciones web sean interactivas en base al ciclo acción del usuario – reacción de la aplicación.

Podríamos hablar de distintos tipos de eventos:

a) Eventos interceptables y eventos no interceptables: un evento interceptable es aquel que puede ser detectado por el sistema operativo o el navegador y generar la ejecución de un fragmento de código JavaScript como consecuencia de ello. Estos son los eventos JavaScript con los que trabajaremos. Un evento no interceptable sería aquel que no puede ser detectado, como que el usuario se rasque la cabeza o que reciba un mensaje de correo electrónico en una aplicación local.

b) Eventos disparados por el usuario y eventos disparados por el sistema: cada vez que el usuario realiza una acción interceptable, como hacer click ó pulsar una tecla, decimos que se dispara un evento generado por el usuario. Pero además de eventos disparados por el usuario, existen eventos disparados por el sistema operativo o por el navegador. Por ejemplo, cuando termina la carga de una página web en el navegador se dispara el evento de sistema <<onload>> (carga terminada). También podemos incluir instrucciones específicas JavaScript para que se ejecuten cuando tenga lugar un evento de sistema.

TIPOS DE EVENTOS. OBJETIVO O TARGET DE EVENTO.

Cada evento tiene un **nombre**, al que también se llama "**tipo de evento**". Por ejemplo al evento correspondiente a hacer click se le identifica como "onclick". Normalmente los nombres se forman anteponiendo el prefijo on con el nombre de lo que ocurre en inglés. Así por ejemplo "onload" indica "cuando termina la carga del documento HTML".

Cada evento está asociado a un objeto que es quien recibe o detecta el evento, al que se denomina target u objetivo del evento. Así, el evento onclick del ejemplo anterior tiene como objetivo un nodo HTML de imagen, y un nodo HTML div. El evento onload tiene como target el objeto window u objeto global.

MODELOS DE EVENTOS. PROPAGACIÓN.

Al igual que definir la estructura de un documento HTML es complejo y por ello se creó el DOM (Document Object Model), para poder describirla y manejarla, existen modelos de eventos para describir y manejar los eventos.

El problema reside en que a lo largo del tiempo se crearon distintos modelos de eventos propuestos por distintas organizaciones (Netscape, Microsoft, W3C, etc.). Distintos navegadores adoptaron distintos modelos de eventos, lo cual hacía difícil que el código programado y útil en un navegador funcionara en otro que usaba un modelo de eventos distinto.

Un modelo de eventos implica como aspecto importante qué eventos son detectados y cuáles son sus nombres. Pero esta no es la única cuestión relacionada con los eventos. Otra cuestión importante es en qué orden se ejecutan dos eventos que suceden simultáneamente. Por ejemplo, si tenemos un evento onclick definido para un div, y otro evento onclick definido para una imagen dentro del div, al hacer click sobre la imagen se producen dos eventos simultáneamente: el evento onclick correspondiente al div y el evento onclick correspondiente a la imagen. ¿Qué código de respuesta se debería ejecutar primero, el correspondiente al div o el correspondiente a la imagen? Al orden en que se van disparando los eventos se le denomina propagación de eventos. La propagación de eventos puede ser desde dentro hacia fuera (p.ej. primero el evento de la imagen y luego el del div), desde fuera hacia dentro (p.ej. primero el evento del div y luego el de la imagen), o para casos más complejos, según otras formas de establecer el orden.

Cuestiones como esta también están comprendidas en el modelo de eventos. Al existir distintos modelos de eventos, el resultado era (y en cierta medida es) que el orden en que se ejecutaban los eventos dependía del navegador que estuviéramos utilizando.

Con el paso del tiempo se ha ido produciendo cierta estandarización entre navegadores, aunque esta todavía no es completa. Nosotros en este curso no vamos a entrar a estudiar en profundidad ningún modelo de eventos en concreto, sólo haremos referencias puntuales a ellos cuando lo consideremos necesario. Nos vamos a limitar a describir la forma más estandarizada y aceptada por la industria (los fabricantes de navegadores) y la comunidad de programadores para trabajar con eventos.

Pero debes tener en cuenta lo siguiente: distintos navegadores pueden responder de distinta manera a un mismo código. Algunos navegadores pueden no reconocer el código que sí es reconocido por otros navegadores. Algunos navegadores pueden considerar como eventos algo que otros navegadores simplemente ignoran y no son capaces de capturar.

¿Debemos preocuparnos por esto? Pues la verdad es que no, ya que eso no nos aportará ninguna solución. La estrategia que recomendamos es tratar de usar el código más inter-compatible que podamos y sepamos, siendo conscientes de que hoy por hoy es imposible conseguir que una página web se muestre exactamente igual y responda exactamente igual en todos los navegadores.

Conociendo que no podemos cambiar esta situación, nuestro objetivo será crear páginas web cuyo funcionamiento sea correcto en el mayor número de navegadores posible.

MANEJADOR DE EVENTOS O EVENT HANDLER COMO ATRIBUTO DE ELEMENTO HTML. CONFIRM.

¿Cómo indicar en el código que una serie de instrucciones deben ejecutarse cuando tiene lugar un evento?

La forma más tradicional sería la consistente en añadir un atributo a la etiqueta HTML que recibe el evento, también llamado "captura de eventos en línea":

```
<etiqueta nombreEvento="acción a ejecutar"> ... </etiqueta>
```

Donde acción a ejecutar es un fragmento de código JavaScript, que en caso de comprender varias sentencias deben ir separadas por punto y coma.

```
<a onclick="alert('Cambiamos la url')" href="http://aprenderaprogramar.com">Aprende a programar</a>
```

En este ejemplo podemos decir que onclick es el nombre del evento (aunque a veces se prefiere decir que el nombre del evento es click). También podemos decir que es un atributo de la etiqueta <a>, y podemos decir que es el manejador del evento: el elemento que dice qué ha de ejecutarse cuando se produzca el evento.

Una posibilidad interesante de este tipo de manejadores de eventos es que podemos pasar a una función el elemento HTML que recibe el evento usando la palabra clave this. Por ejemplo:

```
<a onclick="llamarFuncion(this)" href="http://aprenderaprogramar.com">Aprende a programar</a>
```

Al invocar a this estamos pasando el elemento HTML, nodo del DOM, <a>, lo cual puede ser de interés en numerosas ocasiones en que nos interese ejecutar una serie de acciones sobre el elemento que recibe la acción.

Con esta forma de manejo nos podemos plantear una pregunta: si hay una acción predeterminada cuando se pulsa el link (dirigirnos a la url de destino) ¿Qué se ejecuta primero, el código de respuesta o la acción predeterminada? Se estandarizó que en primer lugar se ejecuta el script de respuesta al evento, y luego la acción predeterminada.

Pero como en programación todo es posible, se planteó la cuestión: ¿Por qué no idear una forma de hacer que la ejecución predeterminada se produzca sólo cuando nosotros queramos?

La solución que se le dio a esto fue permitir que el manejador del evento devolviera un valor booleano: true (por defecto) si se debía ejecutar la acción predeterminada, o false para evitar que se ejecutara la acción predeterminada.

```
<etiqueta nombreEvento="acción a ejecutar; return valorBooleano"> ... </etiqueta>
```

```
<a onclick="alert('Link desactivado'); return confirm('Va a acceder a aprenderaprogramar.com')"
href="http://aprenderaprogramar.com">Aprende a programar</a>
```

La ejecución de la acción predeterminada se podía dejar en manos del usuario a través de un mensaje de confirmación usando la función global de JavaScript confirm, de modo que el manejador devuelve true si el usuario pulsa “Aceptar” o false si el usuario pulsa en “Cancelar”.

```
<a onclick="return confirm('Va a acceder a aprenderaprogramar.com ¿Está seguro?')"
href="http://aprenderaprogramar.com">Aprende a programar</a>
```

Algunos eventos, como el cierre de una ventana, pueden generar la ejecución de un código JavaScript, pero no se puede impedir en este caso la acción predeterminada porque sería “ir contra los deseos del usuario”. Quizás se pudiera impedir, pero no tendría lógica.

Esta forma de introducir manejadores de eventos, digamos que la más antigua de todos, es reconocida por todos los navegadores, al igual que los cuatro eventos “mágicos” (en el sentido de que cuando se introdujeron fueron un gran avance en el desarrollo web) onclick, onload, onmouseover, onmouseout.

Desde la época del navegador Netscape en que se introdujeron los manejadores de eventos básicos y los eventos básicos hasta hoy en día, ha habido una larga y complicada evolución.

Hoy día podemos decir que la captura de eventos en línea, aunque es compatible con todos los navegadores, tiene un serio inconveniente: nos obliga a mezclar aspectos de comportamiento o respuesta dinámica con aspectos de estructura del documento HTML. Como sabemos, hoy día hay consenso en que es ventajoso separar la estructura (HTML) de la presentación (CSS) y del comportamiento (JavaScript).

MANEJADORES DE EVENTOS COMO PROPIEDADES DE OBJETOS

Podemos independizar los manejadores de eventos usando código JavaScript para crearlos manejando la propiedad de nodos HTML del DOM que representa el atributo de la etiqueta HTML.

Lo entenderemos mejor con un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
</head>
```

```
<body>
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
<a href="http://aprenderaprogramar.com">Aprende a programar.</a>
<a href="http://aprenderaprogramar.es">Inténtalo, será divertido.</a>
</body>
</html>
<script type="text/javascript">
var elemsA = document.getElementsByTagName('a');
for (var i=0;i<elemsA.length;i++) {
elemsA[i].onclick = function() {return confirm('Ud. va a ser transferido de url hacia ' + this.href);}
}
</script>
```

Sobre este ejemplo tenemos que llamar la atención sobre algunas cuestiones:

a) El código JavaScript está incluido detrás del código HTML. ¿Por qué? Porque si es incluido antes del código HTML, se ejecuta antes de que se haya cargado la página, y al no haberse cargado los elementos HTML no es posible establecer la propiedad onclick de los mismos (es decir, getElementsByTagName('a') nos devuelve cero elementos cuando todavía no se ha cargado la página web.

b) A cada nodo de tipo link (a) se le asigna como propiedad onclick una función, en este caso haciendo uso de return y confirm, de modo que el usuario puede elegir entre aceptar el ir al link elegido o cancelar.

c) this dentro de la función anónima a ejecutar ante un evento onclick sobre un elemento a hace referencia al elemento propietario de la función, en este caso el propio nodo de tipo link a, de modo que podemos rescatar la url de destino escribiendo this.href.

Una forma alternativa de escribir el script sería:

```
<script type="text/javascript">
var elemsA = document.getElementsByTagName('a');
for (var i=0;i<elemsA.length;i++) {
elemsA[i].onclick = transferir;}
function transferir() {
return confirm('Ud. va a ser transferido de url hacia ' + this.href);
}
</script>
```

En este caso usamos una asignación a la función. Tener en cuenta que no escribimos elemsA[i].onclick = transferir();} porque esto daría lugar a que se ejecutar la función (invocación). Al escribir la asignación sin paréntesis la función queda asignada pero no se ejecuta.

Finalmente tenemos una alternativa "más lógica" que nos permite no tener que poner el script después del código HTML. La idea es la siguiente: el evento onload nos informa de cuándo se ha completado la carga de la página web. En ese momento dispararemos una función que se encargará de introducir los manejadores de eventos que queramos. Como en el momento en que se dispara la función la carga de

la página ya se ha realizado, no es necesario que el código JavaScript para realizar este proceso quede "debajo" del código HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
window.onload = function () {
var elemsA = document.getElementsByTagName('a');
for (var i=0;i<elemsA.length;i++) {
elemsA[i].onclick = transferir;}
function transferir() { return confirm('Ud. va a ser transferido de url hacia ' + this.href);}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
<a href="http://aprenderaprogramar.com">Aprende a programar.</a>
<a href="http://aprenderaprogramar.es">Inténtalo, será divertido.</a>
</body>
</html>
```

Recordar que al escribir window.onload = algo(); la función algo() se ejecutará porque mediante esta instrucción estamos indicando que esa función debe ejecutarse cuando se produzca el evento onload sobre el objeto window.

Por ejemplo window.onload = alert ('La página ha terminado de cargar'); nos permite mostrar un mensaje de alerta advirtiendo de que la página ha terminado de cargar.

EJERCICIO

A partir del siguiente código HTML, crea un script que cumpla los requisitos indicados más abajo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html> <head><title>Portal web - aprenderaprogramar.com</title>
<meta name="description" content="Portal web aprenderaprogramar.com">
<meta name="keywords" content="aprender, programar, cursos, libros"><meta charset="utf-8">
</head>
 <body> <p><a href="principal.html" title="Página principal" >Ir a la pagina principal</a></p>
<h1>Novedades</h1>
<p>Aquí presentamos las novedades del sitio.</p>
<h3>Lanzamos el producto X-FASHION</h3>
<p>Este producto permite estirar la piel hasta dejarla como la de un bebé.</p>
<p><img src="http://i.imgur.com/tq5Bs.png" title="Imagen del producto" alt="imagen producto x-fashion" /></p>
<h3>Mejoramos el producto T-MOTION</h3>
<p>Hemos lanzado una nueva versión del producto T-MOTION</p>
<p><img src="http://i.imgur.com/SpZyc.png" title="Imagen del proudcto tmotion" alt="imagen producto tmotion" /></p>
</body>
</html>
```

a) Mediante el control de los eventos onmouseover y onmouseout, debemos hacer que cuando el usuario pase el ratón sobre las etiquetas h1 y h3, el color del texto pase a ser orange y cuando deje de pasarlo el texto quede en marrón.

b) Mediante el control del evento onmouseover, debemos hacer que cuando el usuario pase el ratón sobre un párrafo, el color de fondo del párrafo sea amarillo y cuando deje de pasarlo no haya color de fondo.

c) El código JavaScript debe estar situado entre las etiquetas <head> ... </head>

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01158E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ADDEVENTLISTENER JAVASCRIPT. REMOVE. ATTACHEVENT. THIS EN EVENTOS. PROPAGACIÓN. BUBBLING. CAPTURA. (CU01158E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº58 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

ADDEVENTLISTENER JAVASCRIPT

Pueden ser objetivos (target) de eventos el objeto window, el objeto document y todos los nodos de tipo Element de un documento HTML. Todos estos objetos disponen de un método predefinido denominado addEventListener() que permite agregarles uno, dos o más manejadores de eventos.

La sintaxis a emplear para addEventListener es básicamente la siguiente:

```
nodoObjetivo.addEventListener ("nombreDeEventoSinPrefijo", funcionAEjecutar,
parametroBooleanoOpcional)
```

Donde nodoObjetivo será un nodo que habremos obtenido con un método como getElementById ó similar, nombreDeEventoSinPrefijo es el nombre del evento sin el prefijo on, por ejemplo click (en lugar de onclick como se escribiría tradicionalmente).

funcionAEjecutar es la función que se ejecutará cuando se produzca el evento.

Finalmente, el parámetroBooleanoOpcional especifica si el evento debe ser capturado (true) o no debe serlo (false, que es el valor que se aplica por defecto si no se especifica este parámetro. El tercer parámetro no lo especificaremos, o lo haremos como false. El motivo por el que se encuentra ahí es más histórico que una necesidad real y no vamos a detenernos a hablar sobre este parámetro.

Veamos un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
window.onload = function () {
var elemsH = document.querySelectorAll("h1, h2, h3, h4, h5, h6");
for (var i=0;i<elemsH.length;i++) {
elemsH[i].addEventListener("mouseover", cambiarColor1);
elemsH[i].addEventListener("mouseout", cambiarColor2);}
function cambiarColor1() { this.style.color = 'orange';}
function cambiarColor2() { this.style.color = 'brown';}
}
</script>
</head>
<body><p><a href="principal.html" title="Página principal" >Ir a la pagina principal</a></p>
<h1>Novedades</h1><p>Aquí presentamos las novedades del sitio.</p>
<h3>Lanzamos el producto X-FASHION</h3>
<p>Este producto permite estirar la piel hasta dejarla como la de un bebé.</p>
```

```
<p><img src="http://i.imgur.com/tq5Bs.png" title="Imagen del producto" alt="imagen producto x-fashion" /></p>
<h3>Mejoramos el producto T-MOTION</h3>
<p>Hemos lanzado una nueva versión del producto T-MOTION</p>
<p><img src="http://i.imgur.com/SpZyc.png" title="Imagen del proudcto tmotion" alt="imagen producto tmotion" /></p>
</body>
</html>
```

El resultado esperado es que cuando se pase el mouse por encima de un elemento de título <h1>, <h2>, etc. el color cambie a naranja. Una vez el mouse deja de estar encima del elemento, el color del texto cambiará a marrón. Ten en cuenta que en algunos navegadores antiguos el código puede que no funcione.

PALABRA CLAVE THIS

Recordar que dentro de la función de respuesta a un evento, la palabra clave this hace referencia al elemento que es quien recibe (objetivo o target) el evento. En el ejemplo anterior vemos un ejemplo con el código this.style.color = 'orange'; que se encarga de cambiar el color del texto del elemento HTML donde se ha generado el evento.

REMOVEEVENTLISTENER

Al igual que podemos añadir manejadores de eventos con addEventListener, podemos eliminarlos con removeEventListener, cuya sintaxis básica es la siguiente:

```
nodoObjetivo.removeEventListener ("nombreDeEventoSinPrefijo", funcionAEjecutar,
parametroBooleanoOpcional)
```

Donde los parámetros tienen el mismo signficado que con addEventListener.

Un ejemplo nos aclarará las ideas. En este caso, vamos a plantear una variante del código de ejemplo que vimos para addEventListener. Ese código daba lugar a que cuando el usuario pasaba el ratón sobre un elemento de tipo título <h1>, <h2>, … el texto tomara color naranja, y cuando dejara de estar encima de ese elemento tomara color marrón. Pero al repetirse el paso por un elemento que estaba en color marrón, volvía a ponerse naranja porque los manejadores de eventos seguían actuando de la misma manera. Una forma de anular ese cambio de color es eliminar los manejadores de eventos después de que el texto haya cambiado a color marrón. Al anular los manejadores de eventos, una vez el texto toma el color marrón, quedará ya en ese color definitivamente. El cambio a introducir es en la función cambiarColor2, dejando el código como sigue:

```
function cambiarColor2() { this.style.color = 'brown';
this.removeEventListener("mouseover", cambiarColor1);
this.removeEventListener("mouseout", cambiarColor2);
}
```

ATTACHEVENT() Y DETACHEVENT()

Estos métodos fueron usados en el pasado por algunos navegadores como equivalentes a addEventListener y removeEventListener. Desde el momento en que todos los navegadores modernos soportan los métodos addEventListener y removeEventListener, estos métodos no deben ser usados excepto en casos muy concretos en que tengamos que resolver situaciones muy específicas.

PROPAGACIÓN DE EVENTOS. BUBBLING.

Ya hemos indicado que ciertas acciones del usuario generan varios eventos simultáneos. Por ejemplo si tenemos una imagen dentro de un div, y tenemos definida la captura del evento click para ambos, cuando se hace click sobre la imagen se producen en realidad dos eventos: el evento click sobre la imagen y el evento click sobre el div.

La propagación se dice que consta de varias fases: la captura sobre los elementos que sufren el evento detectada por el navegador (que va desde fuera del DOM hacia dentro), el envío del evento al manejador del evento (donde el evento es enviado como un objeto), y el burbujeo, que da lugar a la ejecución del código de respuesta asociado al evento (que va desde dentro del DOM hacia fuera).

El comportamiento que secuencia la ejecución del código de respuesta con un orden determinado se llama “bubble” o “burbujeo”, de modo que desde el elemento más interior del DOM se va ejecutando la respuesta al evento y seguidamente se va expandiendo la ejecución hacia los elementos más externos. En nuestro ejemplo, se ejecutaría en primer lugar la respuesta al evento sobre la imagen por estar dentro del div, y a continuación la respuesta el evento sobre el div. El burbujeo continua sobre todos los elementos del DOM hasta llegar a document y al objeto global window.

Casi todos los eventos burbujean, pero algunos eventos no lo hacen. Por ejemplo el evento focus (onfocus, obtener el foco un campo input) no lo hace. Algunos eventos tienen un burbujeo especial, por ejemplo el evento load (onload, carga de un elemento) burbujea hasta el elemento document, pero no alcanza al elemento window.

¿QUÉ FORMA DE INTRODUCIR MANEJADORES DE EVENTOS ES MEJOR?

Hemos visto tres maneras diferentes de introducir manejadores de eventos: en línea dentro del código HTML, como propiedades de nodos del DOM y a través de la función addEventListener. ¿Cuál es mejor?

Haremos las siguientes consideraciones:

- Introducir los manejadores de eventos en línea es simple pero tiene inconvenientes serios: no separa la estructura de la página web (HTML) del código JavaScript. Esto dificulta el mantenimiento del código y se considera una práctica no adecuada. Por otro lado, se generan conflictos de nombres.

- Introducir manejadores de eventos como propiedades puede resultar adecuado, pero cuando hay que introducir varios manejadores para un mismo evento puede "oscurecer" (hacer más difícil de leer) el código. Por ejemplo para introducir dos manejadores para el evento onclick escribiríamos element.onclick = function () {dispararA(); dispararB()}.

- Introducir manejadores de eventos a través de addEventListener, a pesar de que ha tenido algunos problemas de compatibilidad en el pasado, es un modo que se está estandarizando y se considera también adecuado.

EJERCICIO

Crea un documento HTML donde dentro del elemento body tengamos un div con id "principal", dentro de principal otro div denominado "secundario", y dentro de secundario otro div con id "terciario". Dentro de terciario debe existir un párrafo con el texto: Ejemplo de bubbling (burbujeo). Añade eventListeners con el evento click para los párrafos y todos los elementos div, document y window, y una función de respuesta común para todos ellos que emita el mensaje de alerta <<Soy un nodo tipo NombreDelNodo y estoy burbujeando>>.

Ejemplo: al hacer click sobre el texto <<Ejemplo de bubbling (burbujeo)>> deberán empezar a aparecernos mensajes como: Soy un nodo tipo P y estoy burbujenado. Soy un nodo tipo DIV y estoy burbujeando. Soy un nodo tipo DIV y estoy burbujeando...

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01159E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

LISTA DE EVENTOS JAVASCRIPT. (ON) CLICK, DBLCLICK, MOUSEOVER, MOUSEOUT, CHANGE, SUBMIT, KEYPRESS, ETC. (CU01159E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº59 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

LISTA DE EVENTOS JAVASCRIPT

Ya sabemos cómo añadir manejadores de eventos para ejecutar un fragmento de código JavaScript cuando tenga lugar un determinado evento. Pero, ¿cuáles son los eventos reconocibles? ¿cuántos hay? ¿cómo podemos saber sus nombres? ¿A qué elementos HTML son aplicables?

Estas preguntas que a priori parecen sencillas, en realidad son complicadas de responder debido a diversos motivos como:

a) El número de eventos reconocibles se está ampliando constantemente. Inicialmente había 6 u 8 eventos reconocibles, luego fueron 15 ó 20, luego 50 ó 60, luego 100 ó 120... Por tanto no podemos responder a la pregunta de cuántos eventos existen, porque el número de eventos reconocibles cambia con frecuencia.

b) Algunos eventos sólo son reconocidos por algunos navegadores pero no por otros. Nosotros vamos a tratar de centrarnos en los eventos que son reconocidos por todos los navegadores.

c) No existe un organismo único encargado de definir los eventos reconocibles, sino varios. Es decir, los eventos con los que puede trabajar un navegador son el resultado de la definición de eventos realizada por diferentes especificaciones oficiales (DOM, HTML5, CSS Animations, especificaciones oficiales de diversas APIs, etc.) que van siendo integradas en los navegadores.

Por tanto más que pretender conocer todos los eventos y en base a ello escribir nuestro código, más bien debemos preguntarnos qué es lo que queremos hacer y en base a ello buscar el evento que nos permite hacerlo. Una vez localizado el evento, deberemos documentarnos en internet para comprobar cuál es la compatibilidad con los distintos navegadores. Recomendamos usar sólo aquellos eventos que tengan compatibilidad contrastada en la mayoría de los navegadores.

La siguiente lista por tanto sólo pretende ser orientativa de los eventos disponibles de uso más habitual.

Son eventos que se usan con mucha frecuencia onload, para realizar acciones una vez termina la carga de la página web, así como onclick, onmouseover, onmouseout para control de la apariencia de la página web según por donde pase el usuario el ratón (mouse), y onchange, onsubmit y onreset para control de los formularios.

Tener en cuenta que no todos los eventos hacen bubbling y no todos tienen acción por defecto, o si la tienen, no para todos puede ser evitada. Cuando trabajemos con un evento es conveniente buscar documentación adicional sobre su comportamiento y respuesta en los distintos navegadores.

LISTA DE EVENTOS JAVASCRIPT

Tipo de evento	Nombre con prefijo on (eliminar cuando proceda)	Descripción aprenderaprogramar.com
Relacionados con el ratón	onclick	Click sobre un elemento
	ondblclick	Doble click sobre un elemento
	onmousedown	Se pulsa un botón del ratón sobre un elemento
	onmouseenter	El puntero del ratón entra en el área de un elemento
	onmouseleave	El puntero del ratón sale del área de un elemento
	onmousemove	El puntero del ratón se está moviendo sobre el área de un elemento
	onmouseover	El puntero del ratón se sitúa encima del área de un elemento
	onmouseout	El puntero del ratón sale fuera del área del elemento o fuera de uno de sus hijos
	onmouseup	Un botón del ratón se libera estando sobre un elemento
	contextmenu	Se pulsa el botón derecho del ratón (antes de que aparezca el menú contextual)
Relacionados con el teclado	onkeydown	El usuario tiene pulsada una tecla (para elementos de formulario y body)
	onkeypress	El usuario pulsa una tecla (momento justo en que la pulsa) (para elementos de formulario y body)
	onkeyup	El usuario libera una tecla que tenía pulsada (para elementos de formulario y body)
Relacionados con formularios	onfocus	Un elemento del formulario toma el foco
	onblur	Un elemento del formulario pierde el foco
	onchange	Un elemento del formulario cambia
	onselect	El usuario selecciona el texto de un elemento input o textarea
	onsubmit	Se pulsa el botón de envío del formulario (antes del envío)
	onreset	Se pulsa el botón reset del formulario

Tipo de evento	Nombre con prefijo on (eliminar cuando proceda)	Descripción aprenderaprogramar.com
Relacionados con ventanas o frames	onload	Se ha completado la carga de la ventana
	onunload	El usuario ha cerrado la ventana
	onresize	El usuario ha cambiado el tamaño de la ventana
Relacionados con animaciones y transiciones	animationend, animationiteration, animationstart, beginEvent, endEvent, repeatEvent, transitionend	
Relacionados con la batería y carga de la batería	Chargingchange, chargingtimechange, dischargingtimechange levelchange	
Relacionados con llamadas tipo telefonía	alerting, busy, callschanged, connected, connecting, dialing, disconnected, disconnecting, error, held, holding, incoming, resuming, statechange	
Relacionados con cambios en el DOM	DOMAttrModified, DOMCharacterDataModified, DOMContentLoaded, DOMElementNameChanged, DOMNodeInserted, DOMNodeInsertedIntoDocument, DOMNodeRemoved, DOMNodeRemovedFromDocument, DOMSubtreeModified	
Relacionados con arrastre de elementos (drag and drop)	drag, dragend, dragenter, dragleave, dragover, dragstart, drop	
Relacionados con video y audio	audioprocess, canplay, canplaythrough, durationchange, emptied, ended, ended, loadeddata, loadedmetadata, pause, play, playing, ratechange, seeked, seeking, stalled, suspend, timeupdate, volumechange, waiting, complete	
Relacionados con la conexión a internet	disabled, enabled, offline, online, statuschange, connectionInfoUpdate	
Otros tipos de eventos	Hay más tipos de eventos: relacionados con la pulsación sobre pantallas, uso de copy and paste (copiar y pegar), impresión con impresoras, etc.	

Esta cantidad de eventos puede resultar desconcertante por excesiva. No te preocupes por entender ahora el significado de cada uno de ellos. Simplemente ten unas nociones básicas que te permitan resolver los retos que como programador te puedan ir surgiendo.

EJERCICIO 1

Crea un documento HTML que conste de un título h1 con el texto <<La web para aprender programación>>, dos párrafos de texto y una imagen. Utiliza los eventos onmouseover para que la imagen original sea reemplazada por otra imagen cuando el usuario pasa el ratón sobre ella. Utiliza el evento onmouseout para hacer que cuando el usuario salga del espacio de la imagen, vuelva a aparecer la imagen original.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un documento HTML que conste de un título h1 con el texto <<La web para aprender programación>>, dos párrafos de texto y un formulario con los campos nombre, apellidos, edad y correo electrónico, así como con botones enviar y cancelar. Utiliza el evento onsubmit y otras herramientas para impedir que se envíe el formulario si se produce alguna de estas circunstancias:

a) El nombre está vacío ó el correo electrónico está vacío.

b) El correo electrónico no contiene los símbolos @ (arroba) y . (punto). Por ejemplo juan arroba gmail.com no es un correo válido.

c) No existe al menos una letra precediendo el símbolo @ del correo electrónico y una letra después de este símbolo. Por ejemplo a@.com no es un correo válido.

d) No existen al menos dos letras después del punto en el correo electrónico. Por ejemplo juan@gmail.c no es un correo válido.

e) La edad es cero o menor de cero.

En caso de producirse una de estas circunstancias, debe mostrarse el campo del formulario de un color distinto y un mensaje de advertencia. El color de advertencia debe desaparecer (dinámicamente) cuando el campo que tenía un problema tome el foco (para ello usa el evento onfocus).

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01160E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

OBJETO EVENT JAVASCRIPT. TYPE, CURRENTTARGET, TARGET, TIMESTAMP, CLIENTX, SCREENX, PAGEX, BUTTON. (CU01160E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº60 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

OBJETO EVENT. INFORMACIÓN DEL EVENTO.

Podemos añadir manejadores de eventos a los elementos del DOM y hacer que exista uno o varios manejadores de eventos para gestionar la respuesta ante dicho evento. Además en ocasiones necesitaremos información relacionada con el evento: por ejemplo en qué coordenadas de la pantalla se ha hecho click, o qué tecla ha sido pulsada. Para ello JavaScript dispone del objeto Event.

El objeto Event es un objeto predefinido de JavaScript que almacena información sobre un evento y es pasado, para cada evento que tiene lugar, como argumento a la función o funciones que gestionan el evento.

Las propiedades y métodos del objeto Event presentan la misma problemática que los eventos en relación a la estandarización y compatibilidad entre navegadores. No todos los navegadores reconocen la misma información en el objeto Event. También las propiedades del objeto Event se están ampliando continuamente, con lo cual es difícil decir qué cantidad de propiedades (información) está disponible en el objeto Event. Y también no existe una única especificación oficial de propiedades del objeto Event. Nosotros intentaremos abstraernos de esa situación y centrarnos en comprender los conceptos y conocer las propiedades y métodos del objeto event más usados y estandarizados. Comprendiendo esto, cuando necesitemos alguna información específica podremos buscar información para resolver un problema concreto. Esta es una mejor estrategia que tratar de conocer de antemano y tener un listado preciso de propiedades y métodos del objeto Event.

Las propiedades y métodos del objeto Event se resumen en la siguiente tabla. Veremos su aplicación práctica en el código de ejemplo que incluimos posteriormente:

Tipo	Nombre	Descripción aprenderaprogramar.com
Propiedades de control del evento	type	Devuelve el tipo de evento producido, sin el prefijo on (p.ej. click)
	target	Devuelve el elemento del DOM que disparó el evento (inicialmente)
	currentTarget	Devuelve el elemento del DOM que está disparando el evento actualmente (no nesariamente el elemento que disparó el evento, ya que puede ser un disparo debido a burbujeo)
Otras propiedades de control del evento	eventPhase (indica en qué fase de tratamiento de evento estamos, 1 captura, 2 en objetivo, 3 burbujeo), bubbles (booleana, indica si es un evento que burbujea o no), cancelable (booleana, devuelve si el evento viene seguido de una acción predeterminada que puede ser cancelada), cancelBubble (booleana, devuelve si el evento actual se propagará hacia arriba en la jerarquía del DOM o no).	
Propiedad temporal	timeStamp	Devuelve una medida de tiempo en milisegundos desde un origen temporal determinado.

Tipo	Nombre	Descripción aprenderaprogramar.com
Propiedades de localización del puntero del ratón	clientX, clientY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la ventana del navegador y se expresan en pixeles.
	screenX, screenY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la pantalla y se expresan en pixeles.
	pageX, pageY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda del documento, que pueden ser distintas a las de la ventana si el usuario ha hecho scroll sobre el documento.
Propiedad para detectar el botón del ratón pulsado	button	Normalmente empleado para el evento mouseup (liberación de botón del ratón) para detectar cuál ha sido el botón pulsado. Contiene un valor numérico: 0 para click normal (botón izquierdo), 1 para botón central (botón en el scroll), 2 para botón auxiliar (botón derecho).
Propiedades relacionadas con el teclado	Para determinar qué tecla ha sido pulsada	Lo estudiaremos por separado en la siguiente entrega del curso
Propiedades relacionadas con drag and drop	Algunas no estandarizadas	dataTransfer, dropEffect, effectAllowed, files, types
Otras propiedades varias	Otras (algunas no estandarizadas)	x, y, layerx, layery, offsetX, offsetY, wheelDelta, detail, relatedNode, relatedTarget, view, attrChange, attrName, newValue, prevValue, data, lastEventId, origin, source
Método	stopPropagation()	Detiene la propagación del evento
Método	preventDefault()	Cancela (si es posible) la acción de defecto que debería ocurrir después del evento (equivale a return false para cancelar la acción).
Otros métodos	initEvent(a, b, c) se usa para definir eventos. No lo estudiaremos.	

EJEMPLO CON TRATAMIENTO DE MANEJADORES DE EVENTOS COMO PROPIEDADES

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var tiempoAnterior;
```

```
function ejemploJS1 (evobject) {
alert ('La función de gestión ha recibido objeto Event con tipo: ' + evobject.type);
evobject.target.style.color = 'red'; //Cambiamos el color si se ha producido el evento
if (tiempoAnterior) {
alert('Han transcurrido ' + (evobject.timeStamp-tiempoAnterior)/1000 +' segundos desde el click anterior');}
tiempoAnterior = evobject.timeStamp;
}
function ejemploJS2 (evobject) {alert ('¡Haz hecho doble click!'); evobject.target.style.color = 'red';}
window.onload = function() {
document.getElementById('pulsador1').onclick = ejemploJS1;
document.getElementById('pulsador2').ondblclick = ejemploJS2;}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="pulsador1" > Haz click aquí </div>
<div style="color:blue;" id ="pulsador2" > Haz doble click aquí </div>
</body>
</html>
```

El resultado esperado es que cuando se haga click sobre el texto “Haz click aquí” se muestre el mensaje: La función de gestión ha recibido objeto Event con tipo: click. Además si es la segunda o posterior vez que se hace click aparecerá el tiempo que ha pasado desde el anterior click, por ejemplo: Han transcurrido 5.117 segundos desde el click anterior.

En caso de hacerse doble click sobre el texto “Haz doble click aquí”, se espera que aparezca el mensaje ¡Haz hecho doble click!

Del código anterior podemos destacar lo siguiente:

evobject.target nos devuelve el elemento HTML que disparó el evento. Valiéndonos de esto, cambiamos el color del texto justo después de mostrar los mensajes correspondientes.

Hemos escrito function ejemploJS1 (evobject) donde evobject representa el objeto Event que se recibe y que es un argumento que se pasa automáticamente cuando se invoca la función al dispararse el evento. El nombre de este argumento es el que nosotros decidamos, es decir, podemos llamarlo evobject, eventoDatos, miEvento u objetoRepresentaEvento, etc., según prefiramos. Incluso podríamos escribir function ejemploJS1 (), en este caso no podríamos recuperar información del evento ya que para hacerlo necesitamos hacerlo incluyendo un parámetro en la función.

La función que detecta el evento click es distinta de la que gestiona el evento dblclick ¿Podría ser la misma? Esta pregunta tiene dos respuestas: si la pregunta se refiere a si una función puede gestionar varios eventos distintos la respuesta es sí. Si la pregunta es si una función puede distinguir si se ha hecho click o se ha hecho doble click la respuesta es, inicialmente, que no es fácil. Decimos que inicialmente que no es fácil porque entre los eventos click y dblclick hay una dificultad especial que no existe entre otro tipo de eventos. La dificultad radica en que el evento click es “parte” o “comienzo” del evento dblclick. Por ello no es evidente cómo diferenciar entre dos clicks y un doble click, o cómo detener la ejecución de la respuesta a click cuando se hace doble click. En realidad, es posible que una función diferencie entre click y doble click basándonos en el intervalo de tiempo que transcurre, pero esto requiere un poco más de código y de momento no vamos a detenernos a estudiarlo.

EJEMPLO CON TRATAMIENTO DE MANEJADORES DE EVENTOS COMO ATRIBUTOS

En este caso la diferencia radica en que al indicar el atributo del elemento HTML debemos pasar como argumento event. La sintaxis es:

```
onNombreDelEvento = "nombreFuncion(event)"
```

Donde event representa el objeto Event que se pasa a la función. Por ejemplo podríamos escribir onmousedown="showCoords(event)" que indica que cuando se pulse un botón del ratón sobre el elemento que recibe el evento, debe ejecutarse la función showCoords, que recibirá el objeto Event con la información correspondiente gracias a que indicamos <<event>> como parámetro que se le pasa a la función.

INFORMACIÓN SOBRE EVENTOS DEL TECLADO

La información sobre eventos del teclado la abordaremos en la próxima entrega del curso.

EJERCICIO

Crea un documento HTML que conste de un título h1 con el texto <<Posición x: - Posición y>>, y un div con ancho 500 px y alto 400 px. Utiliza el evento onmouseover para mostrar dinámicamente en el título h1 las coordenadas del puntero del ratón a medida que se desplaza por el elemento div.

Por ejemplo, cuando el usuario coloca el ratón sobre el div se deberá mostrar: Posición x: 244 Posición y: 188, al mover el ratón las coordenadas indicadas cambiarán a otro valor, por ejemplo Posición x: 322 Posición y: 228, y así sucesivamente.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01161E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CAPTURAR EVENTOS DE TECLADO JAVASCRIPT. TECLA PULSADA. FROMCHARCODE. KEYCODE, WICH. (CU01161E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº61 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

EVENTOS DE TECLADO E INFORMACIÓN RELACIONADA

Los eventos de teclado han resultado históricamente difíciles de trabajar con JavaScript. Algunos especialistas han llamado a esto "el caos de JavaScript en el manejo de eventos de teclado". Vamos a tratar de ver los aspectos clave relacionados con la gestión de eventos de teclado.

Los eventos de teclado presentan especial dificultad debido a que existen diferentes juegos de caracteres, diferentes sistemas operativos, diferentes configuraciones de teclado, teclas especiales, caracteres no visibles, caracteres que se obtienen mediante pulsación de dos teclas y por supuesto diferentes idiomas que tienen caracteres no comunes con otros idiomas.

A pesar de todo, no debería ser difícil determinar qué tecla ha sido pulsada y sin embargo históricamente sí ha sido difícil, lo cual es poco entendible en una época en que la tecnología inunda nuestras vidas. Estas dificultades han provenido principalmente de la falta de acuerdo entre los fabricantes de los distintos navegadores para alcanzar una estandarización adecuada. Hoy día podemos decir que esta falta de estandarización está en vías de resolverse (por fortuna no nos enfrentamos a las incompatibilidades que existían hace tan solo unos pocos años).

Hemos de tener en cuenta que existen tres eventos distintos relacionados con pulsar una tecla. Cuando pulsamos una tecla se producen tres eventos, que enunciados por orden son: keydown, keypress, keyup. Los eventos keydown y keyup podemos traducirlos como "la tecla baja" y "la tecla sube", mientras que keypress sería "la tecla es pulsada".

Hay dos tipos de códigos diferentes: los códigos de teclado (que representan la tecla del teclado que ha sido pulsada, tecla física) y los códigos de caracteres (número asociado a cada carácter, según el juego de caracteres Unicode). Tener en cuenta que desde el punto de vista de qué tecla física ha sido pulsada es lo mismo escribir una a que una A, ya que en ambos casos pulsamos la misma tecla en el teclado. En cambio desde el punto de vista del carácter, a es un carácter y A es otro carácter diferente (y cada uno tiene su código numérico diferente).

Para cada evento de teclado las propiedades del objeto Event relacionadas con el teclado pueden tener distinto signficado, por ello hemos de atenernos a unas reglas para poder operar con seguridad. Estas reglas son recomendaciones (nuestras), no se trata de reglas de obligado cumplimiento:

Regla 1: para determinar el carácter resultado de la pulsación de una tecla simple o de una combinación de teclas usaremos el evento keypress (no los eventos keydown ni keyup), determinando la tecla pulsada con la propiedad wich del objeto event.

Regla 2: para determinar si ha sido pulsada una tecla no imprimible (como flecha de cursor, etc.), usaremos el evento keyup, determinando la tecla pulsada con la propiedad keyCode (código de la tecla física).

Regla 3: no intentes controlar todo lo que ocurre con el teclado (porque quizás te vuelvas loco). Controla únicamente aquello que sea necesario y útil.

Tener en cuenta que hay algunas teclas no imprimibles que no generan evento keypress, sino únicamente eventos keydown y keyup. En cambio, algunas teclas no imprimibles sí generan evento keypress.

RECUPERAR UN CARÁCTER A PARTIR DE SU CÓDIGO NUMÉRICO

Cada carácter tiene un código numérico. Guiándonos por el juego de caracteres Unicode los códigos numéricos son 65 para la letra A (mayúscula), 66 para la letra B, ..., 97 para la letra a (minúscula). Teclas especiales como ENTER también tienen asignados códigos numéricos: 13 para ENTER, 37 flecha izquierda del cursor, 38 flecha arriba del cursor, 39 flecha derecha del cursor, 40 flecha derecha del cursor, etc.

Para recuperar el carácter asociado a un código usaremos un método estático del objeto predefinido String, con la siguiente sintaxis:

```
String.fromCharCode(num1, num2, ..., numN);
```

Donde num1, num2, ..., numN son los códigos numéricos a convertir. Si escribimos por ejemplo alert(String.fromCharCode(97,98,99)); obtendremos como resultado <<abc>>

PROPIEDADES Y MÉTODOS DEL OBJETO EVENT RELACIONADOS CON TECLADO

Las propiedades y métodos del objeto Event relacionados con el teclado se resumen en la siguiente tabla. Veremos su aplicación práctica en el código de ejemplo que incluimos posteriormente:

Tipo	Nombre	Descripción aprenderaprogramar.com
Propiedades relacionadas con el teclado	altKey	Contiene el valor booleano true si la tecla ALT estaba pulsada cuando se produjo el evento, o false en caso contrario.
	ctrlKey	Contiene el valor booleano true si la tecla CTRL estaba pulsada cuando se produjo el evento, o false en caso contrario.
	shiftKey	Contiene el valor booleano true si la tecla SHIFT estaba pulsada cuando se produjo el evento, o false en caso contrario.
	charCode	Devuelve el código del carácter unicode generado por el evento keypress. Se recomienda usar wich en lugar de charCode.

Tipo	Nombre	Descripción aprenderaprogramar.com
	keyCode	Devuelve el código de tecla pulsada para eventos keydown y keyup
	wich	Devuelve el código del carácter unicode generado por el evento keypress.
	Otras propiedades	data, metakey
	Otras propiedades no estandarizadas	altLeft, ctrlLeft, shiftLeft, isChar

EJEMPLO: TRABAJANDO CON EVENTOS DE TECLADO

Escribe el siguiente código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var eventoControlado = false;
window.onload = function() { document.onkeypress = mostrarInformacionCaracter;
document.onkeyup = mostrarInformacionTecla; }
function mostrarInformacionCaracter(evObject) {
        var msg = ''; var elCaracter = String.fromCharCode(evObject.which);
        if (evObject.which!=0 && evObject.which!=13) {
        msg = 'Tecla pulsada: ' + elCaracter;
        control.innerHTML += msg + '----------<br/>'; }
        else { msg = 'Pulsada tecla especial';
        control.innerHTML += msg + '----------<br/>';}
        eventoControlado=true;
}
function mostrarInformacionTecla(evObject) {
        var msg = ''; var teclaPulsada = evObject.keyCode;
        if (teclaPulsada == 20) { msg = 'Pulsado caps lock (act/des mayúsculas)';}
        else if (teclaPulsada == 16) { msg = 'Pulsado shift';}
        else if (eventoControlado == false) { msg = 'Pulsada tecla especial';}
        if (msg) {control.innerHTML += msg + '----------<br/>';}
        eventoControlado = false;
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplo JavaScript: pulse una
tecla</h3></div>
<div id="control"> </div>
</body></html>
```

El resultado esperado es que al pulsar una tecla como a ó b se muestre por pantalla Tecla pulsada: a---------- Tecla pulsada: b----------, o que al pulsar shift se muestre Pulsado shift----------, o Pulsada tecla especial para otras teclas especiales.

Ten en cuenta lo siguiente:

Se controla si el evento ha sido detectado con keypress mediante la variable booleana eventoControlado. Si el evento ha sido detectado con keypress, no lo tratamos con keyup (evento que tiene lugar después de keypress). De esta forma tratamos de distinguir, de forma aproximada, entre teclas especiales y normales.

Podrían existir algunas divergencias entre diferentes navegadores o diferentes computadores (diferentes teclados).

Este código trata de ser un ejemplo de cómo trabajar con eventos de teclado, pero puedes crear tus propios trucos para trabajar con el teclado, o usar trucos o bibliotecas creadas por otros programadores que están disponibles en internet.

EJERCICIO 1

Crea un documento HTML que conste de un título h1 con el texto <<Pulsa una tecla>>, y un div central de 400 por 400 px con el borde marcado y márgenes de 100px en todas direcciones. Utiliza el evento keypress para determinar el carácter que el usuario ha escrito utilizando el teclado y haz que se muestre dentro del div con un tamaño de fuente de 250px.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un documento HTML que conste de un título h1 con el texto <<Pulsa una flecha>>, y un div central de 300 por 300 px con borde color negro, color de fondo amarillo y márgenes de 200px en todas direcciones. Inicialmente el cuadrado que define el div debe estar alineado en el centro de la ventana. Utiliza el evento keyup para determinar si el usuario pulsa una tecla del cursor, y en ese caso utilizando CSS haz que el cuadrado que define el div se desplace 20px en la dirección de flecha elegida por el usuario. Por ejemplo, si el usuario pulsa la flecha derecha, el div debe desplazarse 20 px dentro de la ventana, hacia la derecha. Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01162E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DATE JAVASCRIPT. DATE.NOW, DATE.PARSE, DATE.UTC. DIFERENCIAS ENTRE GMT Y UTC Ó LOCAL. GETMONTH, GETDATE, GETDAY. (CU01162E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº62 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

OBJETO DATE JAVASCRIPT

Con frecuencia cuando creamos webs o apps tenemos que trabajar con fechas y calendarios. Por ejemplo en la página web de un hotel o un restaurante es posible que trabajemos con fechas y horas de comienzo de reserva, de fin de reserva, etc. JavaScript dispone del objeto predefinido Date para facilitar el trabajo con fechas.

El manejo del "tiempo" es un aspecto controvertido en la programación. La primera dificultad y más obvia es que existen cientos de dispositivos electrónicos y no todos manejan la misma fecha. Por ejemplo mi computador personal puede indicar que hoy es lunes 25 de enero de 2048 y la hora es las 12:35:44 mientras que el servidor al que estoy conectado, suponiendo que está en mi misma ciudad, puede indicar que hoy es lunes 25 de enero de 2048 y la hora es las 12:33:21. Otro servidor, en otro país, puede indicar que la fecha es domingo 24 de enero de 2048 y la hora es las 23:51:15 debido a la diferencia horaria entre países. Aquí, claro está, nos estaríamos refiriendo a horas locales. Para poder disponer de horas de referencia globales para todo el mundo se crearon estándares como:

a) GMT o tiempo medio de Greenwich: ha sido un estándar ampliamente usado. GMT era el tiempo medido en el observatorio británico de Greenwich, que se definió como meridiano cero de la tierra. Así cada país podía expresar su hora en función de la hora del meridiano de Greenwich. Por ejemplo GMT+0 indicaba que la hora era la misma que la hora oficial en el meridiano de Greenwich, mientras que GMT+3 indicaba que la hora eran 3 horas más que en el meridiano de Greenwich (es decir, si es Greenwich eran las 09:00 en un país cuya hora fuera GMT+3 serían las 12:00).

b) UTC o tiempo universal coordinado: es el estándar que se ha adoptado como referencia que indica un tiempo único independientemente de en qué lugar del planeta nos encontremos, basado en las mediciones de relojes atómicos distribuidos por distintos países. Es el estándar que se está imponiendo en los sistemas informáticos.

La circunferencia terrestre se dividió en 24 husos horarios de modo que cada huso quedaba referenciado a un tiempo común (lo que se denominaba el tiempo de Greenwich).

Aunque desde el punto de vista de su definición técnica UTC y GMT no son lo mismo, a efectos prácticos hablar de GMT+3 es lo mismo que hablar de UTC+3.

A pesar de este gran avance, el trabajo con el tiempo sigue presentando grandes dificultades en los sistemas informáticos, tanto por la falta de sincronización entre dispositivos como por la disparidad en cuanto a cómo medir el tiempo y con qué grado de precisión, existencia de horarios de verano e invierno, etc. Por ello encontrarás que es relativamente frecuente encontrar que a medida que los lenguajes de programación evolucionan vayan introduciendo cambios en la forma de manejar el tiempo.

Una cuestión a tener en cuenta es que el tiempo oficial de Greenwich, tiempo GMT ó UTC, no se corresponde con el tiempo local el Greenwich debido a la existencia del horario de verano. Por ejemplo, Lisboa se encuentra en el huso horario de Greenwich, pero en verano la hora local está adelantada una hora respecto al tiempo oficial de Greenwich por motivos de ahorro energético. Esto da lugar a que el 5 de agosto a las 21:00 en Lisboa se corresponda con el 5 de agosto 20:00 UTC debido al horario de verano, a pesar de que Lisboa se encuentre en el mismo huso horaria que Greenwich.

OBJETO DATE JAVASCRIPT

JavaScript nos provee de un objeto predefinido (existente) del lenguaje que podemos usar para trabajar con fechas en nuestros scripts: el objeto Date. Este es uno de los objetos predefinidos de JavaScript, ya que existen otros (como Math, RegExp, etc.). Además de un objeto predefinido Date define un tipo de dato en JavaScript.

MÉTODOS DEL OBJETO PREDEFINIDO DATE

El objeto Date (predefinido) nos proporciona estos métodos:

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
Date.now()	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas UTC hasta el instante actual UTC según el tiempo del sistema.	alert('Milisegundos desde 1 de enero de 1970: '+Date.now());
Date.parse(fecha)	Transforma el String fecha en una fecha y devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC hasta el instante actual UTC según el tiempo del sistema.	alert('Milisegundos desde 1 de enero de 1970: '+Date.parse('1970-01-02T00:00:00+0000')); //Devuelve 86400000
Date.UTC(año,mes, dia, hora, min, seg, ms)	Año y mes son obligatorios y el resto opcionales. El año ha de ser mayor que 1900. Los meses van de 0 a 11 y los días de 1 a 31. Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC hasta el instante UTC indicado en los argumentos.	alert('Milisegundos desde 1 de enero de 1970 son: '+Date.UTC(1970, 0, 2, 0, 0, 0)); //Devuelve 86400000

El método toString() aplicado a estos métodos devuelve un valor numérico (milisegundos).

CONSTRUCTORES DE OBJETOS TIPO DATE

Además del objeto predefinidio Date, podemos crear objetos Date, que representan "un instante" en el tiempo. Hay cuatro formas distintas de crear objetos Date (constructores):

```
new Date();
new Date(valorEntero);
new Date(fechaString);
new Date(año, mes, día, hora, minuto, segundo, milisegundo);
```

El constructor sin parámetros crea un objeto Date donde el instante de tiempo que representa es el tiempo del sistema en el momento de la creación. El tiempo del sistema es el tiempo de acuerdo con nuestro computador (sistema operativo), y no necesariamente coincidirá con el de otro computador.

Si se invoca Date() sin la palabra new, se obtiene la representación del tiempo actual del sistema. No hay que confundir esta invocación (que sería a una función que devuelve un valor de tipo Date correspondiente al tiempo actual y que cambiará en cada invocación) con la invocación new Date(), que crea un objeto Date que representa el tiempo actual, que queda almacenado en el objeto.

El constructor que recibe un valor entero crea un objeto Date cuyo instante de tiempo está definido por los milisegundos que hayan pasado desde el origen de tiempo (el origen de tiempo en JavaScript está fijado el 1 de enero de 1970 a las 00:00:00 horas, medianoche). Dado que un día tiene 24 horas, una hora 60 minutos, un minuto 60 segundos y un segundo 1000 milisegundos, para representar el 2 de enero de 1970 usaríamos new Date(86400000); , donde el entero entre paréntesis son los milisegundos que contiene un día.

Ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var capturaTiempo = new Date();
alert('capturaTiempo es ' + capturaTiempo);
var unaFecha = new Date(86400000);
alert('El valor numérico 86400000 representa la fecha '+unaFecha);
alert('Date tiempo actual es ' +Date() + ' mientras que capturaTiempo sigue valiendo'+capturaTiempo+
' y unaFecha sigue valiendo '+unaFecha);
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado (ten en cuenta que puede diferir según el navegador y sistema operativo que estés usando) es que se muestre:

capturaTiempo es Mon Aug 11 2094 17:22:48 GMT+0100 >> El valor numérico 86400000 representa la fecha Fri Jan 02 1970 00:00:00 GMT+0000 (Hora de verano GMT) >> Date tiempo actual es Mon Aug 11 2094 17:23:31 GMT+0100 mientras que capturaTiempo sigue valiendoMon Aug 11 2094 17:22:48 GMT+0100 y unaFecha sigue valiendo Fri Jan 02 1970 00:00:00 GMT+0000 (Hora de verano GMT)

Comprobamos que el método toString() aplicado a objetos de tipo Date devuelve una representación como cadena de texto de la fecha.

El constructor que recibe un String crea un objeto Date a partir de una cadena de texto que representa una fecha construida según las formas normalizadas válidas. Hay múltiples formas normalizadas (europeas, norteamericanas) y algunos navegadores reconocen más formas que otros, por lo que no vamos a estudiar todas ellas, sino a mostrar ejemplos en una forma recomendada consistente en indicar año-mes-diaThh:mm+xxyy donde +xxyy indica una diferencia horaria respecto a la hora UTC, que puede ser negativa (hora local retrasada respecto a la UTC) o positiva (hora local adelantada respecto a la UTC).

Ejemplo	Comentarios aprenderaprogramar.com
new Date ('2050-12-25T00:00:00+0000') **ó new Date ('2050-12-25')**	25 de diciembre de 2050 a las 00:00:00 horas locales, estando en una zona horaria sin diferencia respecto a la hora UTC (Greenwich)
new Date ('2050-12-25T00:00+0300')	25 de diciembre a las 00:00:00 horas locales estando en una zona horaria con 3 horas de adelanto respecto a la hora UTC, por tanto representa la fecha 24 de diciembre de 2050 a las 21:00:00 horas UTC
new Date ('2050-12-25T12:00-0600')	25 de diciembre a las 00:00:00 horas locales estando en una zona horaria con 6 horas de retraso respecto a la hora UTC, por tanto representa la fecha 24 de diciembre de 2050 a las 18:00:00 horas UTC

El constructor new Date(año, mes, día, hora, minuto, segundo, milisegundo); trabaja de la misma forma que el método Date.UTC: año y mes son obligatorios y el resto opcionales. El año ha de ser mayor que 1900. Los meses van de 0 a 11 y los días de 1 a 31. Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC hasta el instante UTC indicado en los argumentos. Ejemplo:

```
var miFecha = new Date(1970, 0, 2, 0, 0, 0);

alert('Milisegundos desde 1 de enero de 1970 son: '+(miFecha-0));
```

Escribimos una operación aritmética para forzar a que el método toString devuelva el valor numérico en lugar de una representación del tipo << Fri Jan 02 1970 00:00:00 GMT+0000 (Hora de verano GMT)>>

MÉTODOS GETTERS DE OBJETOS TIPO DATE

Los objetos de tipo Date disponen de estos métodos:

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com

getFullYear()	Devuelve un entero de cuatro dígitos, año correspondiente a la fecha según el tiempo local del sistema.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-31T23:00:00-0600');
		var miDia = miFecha.getFullYear(); //Tenemos 2050

getUTCFullYear()	Devuelve un entero de cuatro dígitos, año correspondiente a la fecha según el tiempo UTC.	Independientemente de en qué zona horaria estemos si ejecutamos:
		var miFecha = new Date('2050-12-31T23:00:00-0600');
		var miDia = miFecha.getUTCFullYear(); //Tenemos 2051

getMonth() y **getUTCMonth()**	Devuelve un entero (entre 0 y 11) correspondiente al mes según el tiempo local del sistema ó según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-31T23:00:00-0600');
		var miDia1 = miFecha.getMonth(); //Tenemos 11
		var miDia2 = miFecha.getUTCMonth(); //Tenemos 0

getDate() y **getUTCDate()**	Devuelve un entero, día del mes correspondiente a la fecha según el tiempo local del sistema ó según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-24T23:00:00-0600');
		var miDia1 = miFecha.getDate(); //Tenemos 24
		var miDia2 = miFecha.getUTCDate(); //Tenemos 25

getDay() y **getUTCDay()**	Devuelve un entero, día de la semana (entre 0 y 6, siendo el 0 el domingo), correspondiente a la fecha según el tiempo local del sistema ó según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-24T23:00:00-0600');
		var miDia1 = miFecha.getDay(); //Tenemos 6 (sábado)
		var m = miFecha.getUTCDay(); //Tenemos 0 (domingo)

getHours() y **getUTCHours()**	Devuelve un entero (entre 0 y 23), correspondiente a la hora según el tiempo local del sistema ó según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-24T23:00:00-0600');
		var miDia1 = miFecha.getHours(); //Tenemos 23
		var m = miFecha.getUTCHours(); //Tenemos 5

getMinutes() y **getUTCMinutes()**	Devuelve un entero (entre 0 y 59) correspondiente al minuto según el tiempo local del sistema o según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-24T23:35:00-0600');
		var miDia1 = miFecha.getMinutes(); //Tenemos 35
		var m = miFecha.getUTCMinutes(); //Tenemos 35

getSeconds() y **getUTCSeconds()**	Devuelve un entero (entre 0 y 59) correspondiente al segundo según el tiempo local del sistema o según el tiempo UTC.	Suponiendo que tenemos como hora local del sistema UTC-6 si ejecutamos:
		var miFecha = new Date('2050-12-24T23:35:15-0600');
		var miDia1 = miFecha.getSeconds(); //Tenemos 15
		var m = miFecha.getUTCSeconds(); //Tenemos 15

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
getMilliseconds() y getUTCMilliseconds()	Devuelve un entero (entre 0 y 999) correspondiente a los milisegundos según el tiempo local del sistema ó según el tiempo UTC.	Sólo útil para aplicaciones muy específicas.

getTime()	Devuelve un entero, número de milisegundos transcurridos desde el 1 de enero de 1970 00:00:00 UTC. El valor es negativo para fechas anteriores a esta fecha.	var miFecha = new Date('1970-01-02T00:00:00+0000');
		var milisec = miFecha.getTime();
		alert('Tenemos: '+milisec); // Tenemos 86400000

getTimezoneOffset()	Devuelve un entero, valor en minutos de la operación tiempo UTC – tiempo local del sistema. Valor negativo para zonas horarias + y positivo para zonas horarias –.	Si tenemos hora local del sistema GMT+0100:
		var desfase = new Date().getTimezoneOffset();
		alert('Tenemos: '+desfase); // Tenemos -60

MÉTODOS SETTERS DE OBJETOS TIPO DATE

Los objetos de tipo Date disponen de estos métodos para establecer los datos correspondientes a una fecha:

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
setFullYear()	Establece un entero de cuatro dígitos, año correspondiente a la fecha según el tiempo local del sistema.	miFecha.setFullYear(2050);
setUTCFullYear()	Establece un entero de cuatro dígitos, año correspondiente a la fecha según el tiempo UTC.	miFecha.setUTCFullYear(2050);

setMonth() y setUTCMonth()	Establece un entero (entre 0 y 11) correspondiente al mes según el tiempo local del sistema ó según el tiempo UTC.	miFecha.setMonth(11); //El mes es diciembre
		miFecha.setUTCMonth(11); //El mes es diciembre

setDate() y setUTCDate()	Establece un entero, día del mes correspondiente a la fecha según el tiempo local del sistema ó según el tiempo UTC.	miFecha.setDate(24); miFecha.setUTCDate(24);
setHours() y setUTCHours()	Establece un entero (entre 0 y 23), correspondiente a la hora según el tiempo local del sistema ó según el tiempo UTC.	miFecha.setHours(23); miFecha.setUTCHours(23);

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
setMinutes() y setUTCMinutes()	Establece un entero (entre 0 y 59) correspondiente al minuto según el tiempo local del sistema o según el tiempo UTC.	miFecha.setMinutes(35); miFecha.setUTCMinutes(35);
setSeconds() y setUTCSeconds()	Establece un entero (entre 0 y 59) correspondiente al segundo según el tiempo local del sistema o según el tiempo UTC.	miFecha.setSeconds(15); miFecha.setUTCSeconds(15);
setMilliseconds() y setUTCMilliseconds()	Establece un entero (entre 0 y 999) correspondiente a los milisegundos según el tiempo local del sistema ó según el tiempo UTC.	Sólo útil para aplicaciones muy específicas.
setTime()	Establece un entero, número de milisegundos transcurridos desde el 1 de enero de 1970 00:00:00 UTC. El valor es negativo para fechas anteriores a esta fecha.	miFecha.setTime(86400000); //La fecha es 2 de enero de 1970 00:00:00 +0000

PARADOJAS CON FECHAS Y RECOMENDACIÓN

Suponiendo que estamos en México D.F. (zona horaria) UTC-6 si ejecutamos:

var miFecha1 = new Date('2050-12-24T23:00:00-0600');

var miDia = miFecha.getDate(); da lugar a que miDia contenga 24

Suponiendo que estamos en Lisboa (zona horaria UTC+0000) si ejecutamos:

var miFecha1 = new Date('2050-12-24T23:00:00-0600');

var miDia = miFecha.getDate();da lugar a que miDia contenga 25

¿Por qué en un caso obtenemos 24 y en otro 25? Primeramente partimos de la fecha '2050-12-24T23:00:00-0600', esta fecha son las 5 de la mañana del 25 de diciembre de 2050 hora UTC. El día del mes expresado como día local depende de en qué zona del planeta estemos: si estamos en México D.F. a esa hora UTC serán las 23:00 del 24 de diciembre, por ello el método getDate() devolvería 24 en un computador situado en México D.F. Si estamos en Lisboa, a esa hora UTC son las 5 de la mañana del 25 de diciembre y por ello el método getDate() devolvería 25 en un computador situado en Lisboa. Cuando decimos situado nos referimos a que tenga como hora local del sistema la hora local del sitio donde está ubicado el computador (aunque no necesariamente tendría por qué ser así).

Esta aparente paradoja (que no es tal) podría traernos complicaciones si no tenemos en cuenta los detalles. A modo de recomendación, recomendamos trabajar siempre bajo la referencia UTC y hacer conversiones a fechas locales de forma controlada.

EJERCICIO

Crea un documento HTML que conste de un título h1 con el texto <<Calendario>>, y un div central de 400 por 400 px con el borde marcado y márgenes de 100px en todas direcciones. Dentro del div central crea una tabla de 7 columnas por 7 filas (total 49 celdas) con ancho de tabla 300 píxeles y tamaño de fuente en la tabla 24 píxeles. La primera columna corresponderá a lunes y la última a domingo. Usa un método JavaScript para determinar el mes actual. Mediante código JavaScript, haz que aparezca dinámicamente como texto encima de la tabla el mes y año de que se trate (por ejemplo <<MAYO DE 2050>>). Haz que cada celda de la primera fila se rellene indicando el día de la semana (Lu – Ma – Mi – Ju – Vi –Sa –Do). Haz que la tabla se rellene dinámicamente (al cargar la página) con:

a) Espacio en blanco si no corresponde ningún día.

b) El número del día del mes según corresponda (28, 30 ó 31 días según de qué mes se trate).

El aspecto, suponiendo que te encuentras en el mes de mayo de 2050, sería el siguiente:

MAYO DE 2050

Lu	Ma	Mi	Ju	Vi	Sa	Do
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Ten en cuenta que debe generarse el calendario del mes en que te encuentres según la hora local del sistema (de tu computador).

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01163E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FORMATO DE FECHAS JAVASCRIPT. TOSTRING, TOLOCALEDATESTRING, TOLOCALETIMESTRING, TOTIMESTRING, ETC. (CU01163E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº63 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MÉTODOS PARA DAR FORMATO A FECHAS JAVASCRIPT

Los objetos de tipo Date disponen de otros métodos además de los que ya conocemos. Estos métodos nos permitirán representar la fecha como cadena de texto, especificando el formato en que queremos que se muestre (12 ó 24 horas, lenguaje, calendario empleado, etc.).

FORMATOS DE FECHAS JAVASCRIPT

Como métodos que facilitan mostrar información sobre los objetos tipo Date en un formato amigable para los humanos tenemos los indicados en la siguiente tabla. Para los ejemplos, hemos supuesto que nuestro computador tiene como hora local del sistema la de Moscú (Rusia), que es UMT+0300.

Como fecha de partida tomamos var fecha2050 = new Date('2050-12-25T23:30:00-0600'); que se corresponde con la hora de México D.F.

Esta fecha expresada como UMT es '2050-12-26T05:30:00+0000' y como hora local del sistema (considerando que es Moscú) '2050-12-26T08:30:00+0300'. Podemos comprobar que según el sistema con el que expresemos la fecha nos encontramos en unos casos en el día 25 de diciembre y en otros casos en el día 26 de diciembre.

Para buscar la lógica al manejo de fechas con JavaScript te recomendamos que tomes como referencia la fecha UTC, y a partir de ahí generar las horas locales (de sistema) que te resulten de interés.

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
toString()	Devuelve una representación de la fecha donde día y mes están expresados en inglés	fecha2050.toString(); // Mon Dec 26 2050 08:30:00 GMT+0300 (Hora de verano de Rusia)
toDateString()	Devuelve una representación abreviada de la fecha donde día y mes están expresados en inglés.	fecha2050.toDateString(); // Mon Dec 26 2050
toISOString()	Devuelve una representación de la fecha en formato ISO, que se corresponde con YYYY-MM-DDTHH:mm:ss.sssZ donde Y es año, M mes, D día, T un separador, H hora, m minuto, s segundos y milisegundos y Z indica que es un tiempo UTC.	fecha2050.toISOString() // 2050-12-26T05:30:00.000Z
toLocaleDateString() **(*Sin Parámetros)**	Devuelve una representación de la fecha (día, mes y año) según la forma de expresar fechas que venga determinada por el tiempo del sistema del computador donde se ejecute.	toLocaleDateString() // 26/12/2050 Nota: supuesto ejecutado en un computador con configuración estándar europea.

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com

toLocaleDateString(localizacion , options)
(* Con parámetros)
	Localizacion (opcional) indica un array de datos de referencia para mostrar la fecha, por ejemplo es-ES para España, es-AR para Argentina... También se puede especificar calendario y otros aspectos.
	Options (opcional) es un array que indica opciones
		var options = {weekday: "long", year: "numeric", month: "long", day: "numeric"};
		alert(fecha2050.toLocaleDateString("es-ES", options));
		// lunes, 26 de diciembre de 2050

toLocaleString() **(*Sin Parámetros)**	Devuelve una representación de la fecha según la configuración local del computador donde se ejecute.	fecha2050.toLocaleString(); // 26/12/2050 9:30:00 Nota: puede variar según configuración computador.

toLocaleString(localizacion , options)
(* Con parámetros)
	Localizacion (opcional) indica un array de datos de referencia para mostrar la fecha, por ejemplo es-ES para España, es-AR para Argentina... También se puede especificar calendario y otros aspectos.
	Options (opcional) es un array que indica opciones
		var options = {weekday: "long", year: "numeric", month: "long", day: "numeric", hour: "numeric", hour12:"false"};
		alert(fecha2050.toLocaleString("es-ES", options));
		// lunes, 26 de diciembre de 2050 9 a. m.

toLocaleTimeString() **(*Sin Parámetros)**	Devuelve una representación de la hora según la configuración local del computador donde se ejecute.	fecha2050.toLocaleTimeString(); // 9:30:00 Nota: puede variar según configuración computador.
toLocaleTimeString() **(* Con Parámetros)**	Localizacion (opcional) indica un array de datos de referencia para mostrar la fecha, por ejemplo es-ES para España, es-AR para Argentina... También se puede especificar calendario y otros aspectos. Options (opcional) es un array que indica opciones	var options = {hour: "numeric", hour12:"false"}; alert(fecha2050.toLocaleTimeString("es-ES", options)); // 9 a. m.
toTimeString()	Devuelve una representación de la hora según el sistema americano.	fecha2050.toTimeString() // 08:30:00 GMT+0300 (Hora de verano de Rusia)
toUTCString()	Devuelve la fecha UTC en formato dependiente de configuración del sistema, normalmente en inglés.	fecha2050.toUTCString() ; // Mon, 26 Dec 2050 05:30:00 GMT
valueOf()	Devuelve un valor numérico que representa el número de milisegundos transcurridos desde el1 de enero de 1970 a las 00:00 UTC.	fecha2050.valueOf(); // 2555645400000

EJERCICIO 1

Crea un script donde se calcule el tiempo en minutos entre que se formula primera petición al usuario y este responde, y entre una segunda petición al usuario y este responde. Ejemplo:

Introduzca su nombre: Alfredo >> Introduzca su país: Colombia >> Han pasado 0.122 minutos entre su primera y segunda respuesta.

Otro ejemplo: Introduzca su nombre: Juan >> Introduzca su país: Chile >> Han pasado 0.73 minutos entre su primera y segunda respuesta.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un script donde pida al usuario que introduzca una primera fecha (fecha1) en formato dd-mm-yyyy, y una segunda fecha en el mismo formato y calcule los días que han pasado entre las dos fechas . Ejemplo: Introduzca la fecha 1: 05-09-2076 >> Introduzca la fecha 2: 09-09-2076 >> Entre las 00:00 horas del primer día a las 00:00 del segundo hay 4 días.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01164E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EJEMPLO RELOJ JAVASCRIPT. SETTIMEOUT, CLEARTIMEOUT, SETINTERVAL, REQUEST ANIMATIONFRAME. (CU01164E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº64 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

TIMERS JAVASCRIPT

Además de Date en JavaScript disponemos de otros objetos y métodos que nos permiten realizar tareas relacionadas con el tiempo. En concreto el objeto global window dispone de varios métodos pensados para ejecutar una función con un cierto retardo, o periódicamente con cierto intervalo de tiempo entre cada ejecución.

TIMERS: SETTIMEOUT, SETINTERVAL, REQUESTANIMATIONFRAME

Normalmente el código JavaScript se ejecuta secuencialmente, pero existen funciones especiales denominadas timers que permiten establecer la ejecución de funciones en determinados momentos del tiempo.

Los timers son funciones predefinidas del objeto window (por tanto se pueden invocar usando window.nombreDeLaFuncion(...) o simplemente usando nombreDeLaFuncion(...).

SINTAXIS TIMER	UTILIDAD	EJEMPLOS aprenderaprogramar.com
var referenciaTimer = setTimeout(nombreFuncion, tiempoMiliseg); *** Para función sin parámetros**	La función nombreFuncion se ejecutará con un retraso de tiempoMiliseg respecto a lo que sería su ejecución natural.	var ejecT = setTimeout(mostrarAlerta, 5000); //mostrarAlerta se ejecuta con 5 segundos de retraso
var referenciaTimer = setTimeout(function() {nombreFuncion (par1, par2, ..., parN)},tiempoMiliseg); *** Para función con parámetros**	La función nombreFuncion se ejecutará con un retraso de tiempoMiliseg respecto a lo que sería su ejecución natural.	var control = setTimeout(function() {ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar)}, 1500); //La función se ejecuta con 1.5 s de retraso
clearTimeOut(referenciaTimer)	Detiene la ejecución programada por referenciaTimer mediante setTimeOut (si se ejecuta antes del tiempo programado)	clearTimeOut(ejecT)
var referenciaTimer = setInterval(nombreFuncion(par1, par2, ... parN), tiempoMiliseg); *** Para función sin parámetros**	Ejecuta periódicamente la función nombreFuncion con un intervalo entre ejecuciones de tiempoMiliseg.	setInterval(mostrarAlerta, 5000); //mostrarAlerta se ejecuta periódicamente cada 5 segundos

SINTAXIS TIMER	UTILIDAD	EJEMPLOS aprenderaprogramar.com
var referenciaTimer = setInterval(function() {nombreFuncion(par1, par2, ... parN)}, tiempoMiliseg); *** Para función con parámetros**	Ejecuta periódicamente la función nombreFuncion con un intervalo entre ejecuciones de tiempoMiliseg.	var t = setInterval(function(){reloj('Chile') },2000); //reloj se ejecuta periódicamente cada 2 s
clearInterval(referenciaTimer)	Detiene la ejecución programada por referenciaTimer mediante setInterval	clearInterval(ejecT)
var referenciaTimer = requestAnimationFrame(nombreFuncion) *** Para función sin parámetros**	Crea un bucle de repintado delegando el control del mismo en el navegador.	globalID = requestAnimationFrame(animacionRepetimos);
var referenciaTimer = requestAnimationFrame(function(){nombreFuncion(par1, par2, ..., parN)}); *** Para función con parámetros**	Crea un bucle de repintado delegando el control del mismo en el navegador.	globalID = requestAnimationFrame(function(){animacionRepetimos('estiloDivertido')});
cancelAnimationFrame(referenciaTimer)	Detiene la ejecución programada por referenciaTimer mediante RequestAnimationFrame.	Ver ejemplo más abajo

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function alertaTras5seg() { setTimeout(mostrarAlerta, 5000); }
function mostrarAlerta() {alert('Han pasado 5 segundos desde la carga de la página');}
</script>
</script></head>
<body onload="alertaTras5seg()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

El resultado esperado es que cuando transcurran 5 segundos desde la carga de la página se muestre el mensaje “Han pasado 5 segundos desde la carga de la página”.

Si queremos introducir la repetición periódica de la ejecución de la función podemos escribir una llamada con setTimeOut dentro de la propia función. Por ejemplo:

```
function mostrarAlerta() {
alert('Han pasado 5 segundos');
setTimeout(mostrarAlerta, 5000);
}
```

Hará que se repita periódicamente la alerta indicando que han pasado 5 segundos.

Aunque si se quiere una repetición periódica será más cómodo usar setInterval:

```
function alertaTras5seg() {setInterval(mostrarAlerta, 5000);}
```

REQUESTANIMATIONFRAME

Tradicionalmente los efectos de progresión o animación incremental con JavaScript se lograban con un código de este tipo:

```
setInterval(function() {  // Código a ejecutar}, 1000/60);
```

Con este código se lograba ejecutar 60 repintados por segundo de una función que por ejemplo iba dibujando una línea. Esta cadencia daba lugar a que pareciera que la línea se dibujaba progresivamente.

A partir de cierto momento, se introdujo una forma de crear estos efectos (puede que no funcione en algunas versiones de navegadores) que trataba de trasladar el control de la cadencia de la animación al propio navegador con varios objetivos:

a) Permitir que el navegador detuviera el proceso en una pestaña si esta pasaba a estar inactiva (dejando así de consumir recursos).

b) Permitir que el navegador optimizara el redibujado, optimizando los recursos y evitando bloqueos.

c) Buscar un menor consumo de energía (CPU).

Algunos programadores adoran requestAnimationFrame y otros lo ignoran o simplemente no lo usan.

El esquema básico para trabajar con requestAnimationFrame es el siguiente:

```
var globalID;
function animacionRepetimos() {
  //Código a ejecutar
  globalID = requestAnimationFrame(animacionRepetimos);
}
function detener() {  cancelAnimationFrame(globalID);}
```

La animación comienza cuando se invoca a una función a la que hemos denominado (a modo de ejemplo) animacionRepetimos. Dentro de esta función existe una llamada recursiva que da lugar a que

el navegador ejecute el repitando periódico ejecutando previamente la función. La función puede ser detenida usando cancelAnimationFrame(nombreDelIdentificadorEmpleado);

Ejecuta este código y comprueba sus resultados (ten en cuenta que en algunos navegadores puede no funcionar, especialmente si son más antiguos):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
button {margin:10px;}
#cabecera{position:absolute; color:white; margin:5px; font-family:verdana, arial;}
.relleno { width: 5px; height: 20px; background: black; float: left;}
</style>
<script type="text/javascript">
window.requestAnimationFrame = window.requestAnimationFrame || window.mozRequestAnimationFrame ||
                    window.webkitRequestAnimationFrame || window.oRequestAnimationFrame;
var globalID;
function animacionRepetimos() {
  var nuevoDiv = document.createElement("div");
  document.getElementById('oculto').appendChild(nuevoDiv);
  nuevoDiv.innerHTML='<div class="relleno"  \> </div>';
  globalID = requestAnimationFrame(animacionRepetimos);
}
function empezar() {globalID = requestAnimationFrame(animacionRepetimos);}
function detener() {  cancelAnimationFrame(globalID);}
</script></head>
<body >
<button id="start" onclick="empezar()">Empezar</button><button id="stop" onclick = "detener()">Detener</button>
<div id="oculto"></div>
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

El resultado esperado es que una el color negro se vaya extendiendo de izquierda a derecha rellenando líneas (al acumularse div de fondo negro) sobre la pantalla haciendo visible el texto blanco que no se desplaza por tener la propiedad CSS position: absolute;

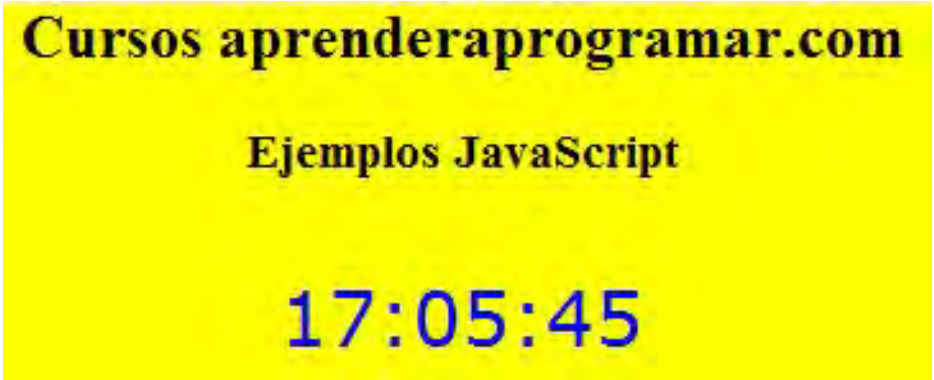
CREAR UN RELOJ CON JAVASCRIPT

Con las herramientas de que disponemos estamos en disposición de crear un reloj con JavaScript.

Ejecuta este código y comprueba sus resultados (ten en cuenta que en algunos navegadores puede no funcionar, especialmente si son más antiguos):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">

function reloj() {
   var hoy=new Date(); var h=hoy.getHours(); var m=hoy.getMinutes(); var s=hoy.getSeconds();
   m = actualizarHora(m);    s = actualizarHora(s);
   document.getElementById('displayReloj').innerHTML = h+":"+m+":"+s;
   var t = setTimeout(function(){reloj()},500);
}

function actualizarHora(i) {
   if (i<10) {i = "0" + i};  // Añadir el cero en números menores de 10
   return i;
}
</script>

</script>
</head>
<body onload="reloj()" >
<div style="text-align:center;">
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; font-family: verdana, arial; font-size:30px; padding:15px;" id ="displayReloj" >   </div>
</div>
</body>
</html>
```

El resultado esperado es que se muestre un reloj marcando los segundos.



EJERCICIO 1

Crea un reloj JavaScript que marque los segundos usando el método setInterval.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un reloj JavaScript que marque los segundos usando el método requestAnimationFrame.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 3

Crea un reloj JavaScript que marque inicialmente 01:00 (es decir, 1 minuto) y que marche hacia atrás (00:59, 00:58, 00:57 ... etc.) hasta llegar a 00:00. Cuando llegue a 00:00 debe deternerse y mostrar el mensaje: "Tu tiempo ha terminado".

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01165E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

MATH JAVASCRIPT. GENERAR NÚMEROS ALEATORIOS RANDOM. REDONDEAR. FUNCIONES TRIGONOMÉTRICAS Y CONSTANTES. (CU01165E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº65 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MATH JAVASCRIPT

JavaScript nos provee de un objeto predefinido con propiedades y métodos útiles de cara a realizar operaciones matemáticas: el objeto Math. Gracias a este objeto podemos acceder a constantes de uso habitual como el número pi, o realizar cálculos trigonométricos, redondeos, logaritmos, potencias, etc.

La precisión de los resultados obtenidos con el objeto Math puede diferir según las características y configuración local del computador donde se ejecuten los scripts.

PROPIEDADES DEL OBJETO MATH

Las propiedades del objeto Math portan constantes que pueden ser útiles para cálculos matemáticos.

Propiedad (constante matemática)	Significado
Math.E	Constante de Euler, base de los logaritmos neperianos, aproximadamente 2.718
Math.LN2	Valor de logaritmo neperiano de 2, aproximadamente 0.693
Math.LN10:	Valor de logaritmo neperiano de 10, aproximadamente 2.303.
Math.LOG2E:	Logaritmo en base 2 de la constante de Euler, aproximadamente 1.443
Math.LOG10E	Logaritmo en base 10 de la constante de Euler, aproximadamente 0.434
Math.PI	Número pi, aproximadamente 3.14159
Math.SQRT1_2	Raíz cuadrada de 1/2, aproximadamente 0.707
Math.SQRT2	Raíz cuadrada de 2, aproximadamente 1.414

MÉTODOS GENERALES DEL OBJETO MATH

Los métodos del objeto Math son útiles para diversos cálculos matemáticos.

SINTAXIS MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
Math.abs(x)	Devuelve el valor absoluto de x	'Valor absoluto de -2 es ' +Math.abs(-2); // 2
Math.ceil(x)	Devuelve el entero más próximo mayor o igual al valor de x (redondeo hacia arriba)	Math.ceil(3.33) // 4 Math.ceil(-3.33) // -3
Math.floor(x)	Devuelve el entero más próximo menor o igual al valor de x (redondeo hacia abajo)	Math.floor(3.33) // 3 Math.floor(-3.33)) // -4
Math.exp(x)	Devuelve e elevado a x, donde e es la constante de Euler (2.718...)	Math.exp(1) //2.718281828459045
Math.log(x)	Devuelve el logaritmo neperiano (con base e) de x	Math.log(Math.E) // 1
Math.max(num1, num2, ..., numN)	Devuelve el máximo valor entre una serie de n números	Math.max(-15, 20) // 20
Math.min(num1, num2, ..., numN)	Devuelve el mínimo valor entre una serie de n números	Math.min(-15, 20) // -15
Math.max.apply(null, nombreArray)	Construcción que permite obtener el máximo valor dentro de un array	Math.max.apply(null, [3, 44, 2]) //44
Math.min.apply(null, nombreArray)	Construcción que permite obtener el mínimo valor dentro de un array	Math.min.apply(null, [3, 44, 2]) //2
Math.pow(x,y)	Devuelve el resultado de elevar el número x al exponente y	Math.pow(2,4) // 16
Math.random()	Devuelve un número pseudo aleatorio comprendido entre 0 (incluido) y 1 (excluido).	alert('Aleatorio entre 1 y 10: ' +Math.floor((Math.random() * (11 - 1)) + 1)); // Devuelve un entero entre 1 y 10
Math.round(x)	Devuelve el entero más próximo a x. Si se trata de número cuya parte decimal es .5 (equidistante), se redondea al entero superior.	Math.round(3.33) // 3
Math.sqrt(x)	Devuelve la raíz cuadrada (valor positivo) de x	Math.sqrt(36) // 6

A Math.max y Math.min no se le puede pasar directamente un array tratando de obtener el máximo o mínimo valor dentro del array, porque la función espera un número o serie de números, y no un objeto array.

MÉTODOS TRIGONOMÉTRICOS DEL OBJETO MATH

Las funciones trigonométricas (sin, cos, tan, asin, acos, atan, atan2) operan en radianes. Pi radianes son 180 grados sexagesimales. Para obtener los grados sexagesimales que son numRad radianes usamos una regla de tres: Math.Pi/180 = numRad / numGrad. numGrad = numRad * (180/ Math.PI).

Para obtener los radianes que son numGrad grados sexagesimales usamos: Math.Pi/180 = numRad / numGrad y obtenemos numRad = numGrad * (Math.Pi/180)

Como mostramos en los ejemplos, se pueden producir errores de precisión (obtener un decimal próximo a cero en lugar de cero, u obtener un número muy grande en lugar de indeterminación por infinito).

SINTAXIS MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com

Math.sin(x)	Devuelve el seno de x (donde x está expresado en radianes)	alert('Seno de Pi es ' +Math.sin(Math.PI));
		// Seno de Pi es 1.2246467991473532e-16 *
		* Error de precisión, resultado real debe ser 0

Math.cos(x)	Devuelve el coseno de x (donde x está expresado en radianes)	alert ('Coseno de Pi es ' +Math.cos(Math.PI)); // Coseno de Pi es -1

Math.tan(x)	Devuelve la tangente de x (donde x está expresado en radianes)	alert('Tangente de 90 o Pi es ' +Math.tan(Math.PI/2));
		// Tangente de 90 o Pi es 16331239353195370 *
		* Error de precisión, resultado real debe ser infinito

Math.asin(x)	Devuelve el arcoseno de x (donde x está expresado en radianes)	alert('Arcoseno de 1 es (90 grados o pi/2) ' +Math.asin(1)); //Arcoseno de 1 es 1.5707963267948966
Math.acos(x)	Devuelve el arcocoseno de x (donde x está expresado en radianes)	alert('Arcocoseno de -1 es ' +Math.acos(-1)); // Arcocoseno de -1 es 3.141592653589793
Math.atan(x)	Devuelve la arcotangente de x (donde x está expresado en radianes)	alert('Arcotangente de 90 es ' +Math.atan(Infinity)); // Arcotangente de 90 es 1.5707963267948966 (este valor son pi/2)
Math.atan2(x, y)	Devuelve la arcotangente del cociente entre x e y (donde el cociente representa un valor en radianes)	alert('Arcotangente de 9/3 es' +Math.atan2(9, 3)); // Arcotangente de 9/3 es1.2587542052323633

EJERCICIO 1

Crea un script que genere un número aleatorio entre 1 y 100. A continuación debe pedir al usuario que adivine el número. Si el usuario responde un número menor al número aleatorio, debe mostrarse un mensaje “El número es mayor. Inténtelo de nuevo” y dar opción a responder de nuevo. Si el usuario responde un número mayor debe mostrarse un mensaje “El número es menor. Inténtelo de nuevo”. Si el usuario acierta debe mostrarse “Enhorabuena. Ha acertado”. El programa debe terminar si el usuario

acierta o si se superan los 30 intentos sin acertar. En caso de superarse los 30 intentos debe mostrarse el mensaje “Ha superado 30 intentos. El programa termina”.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea un script que pida al usuario el dato de radio en metros de la base de un cilindro y el dato de altura del cilindro. Utilizando la propiedad Math.PI debe realizarse el cálculo para determinar el área de la base del cilindro y el volumen del cilindro y mostrar estos resultados. Consulta en internet si no recuerdas las fórmulas a aplicar. .

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 3

Crea un script que dé lugar a lo siguiente:

a) En un div con id="volador” debes tener una palabra como <<JavaScript>>.

b) Utilizando las funciones trigonométricas, las propiedades de posicionamiento CSS y funciones para control de animaciones/tiempo debes dar lugar a que dicha palabra se desplace desde la izquierda hasta la derecha de la pantalla siguiendo un movimiento sinusoidal (es decir, subiendo y bajando suavemente describiendo ondas).

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01166E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EFECTOS JAVASCRIPT VISUALES, DE IMÁGENES, TEXTO... RECURSIÓN. SETTIMEOUT NO FUNCIONA EN BUCLES FOR, WHILE, ¿? (CU01166E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº66 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

EFECTOS JAVASCRIPT

Existen diferentes APIs y librerías JavaScript que permiten generar efectos visuales muy llamativos. En este curso no queremos entrar a estudiar estas posibilidades ya que nos estamos centrando en los fundamentos de JavaScript. Pero ya con estos podemos generar efectos variados.

Los efectos visuales muchas veces se basarán en el uso de recursión y funciones relacionadas con la repetición en el tiempo y ejecución de funciones con cierto retardo en el tiempo.

RECURSIÓN CON JAVASCRIPT

La recursión es una técnica de programación admitida por la mayor parte de los lenguajes modernos basada en la autollamada de un método o función a sí mismo. Estas llamadas van creando "una pila de llamadas" hasta que se llega a lo que se denomina caso base: una situación en que ya no se realiza una nueva llamada recursiva, sino que se devuelve un resultado concreto que da lugar a la salida de la recursión.

La recursión no es fácil de entender ni de explicar. No te preocupes si después de leer lo que vamos a exponer no terminas de comprenderla del todo. Ten en cuenta que muchos programadores no usan o no saben manejar la recursión y que aprender su uso y manejo requiere tiempo y práctica.

Uno de los ejemplos más habituales de recursión es el cálculo del factorial de un número. El factorial de un número n se calcula como n * (n-1) * (n-2) * (...) * 1. Por ejemplo el factorial de 3 es 3*2*1 = 6, mientras que el factorial de 5 es 5*4*3*2*1 = 120.

La forma más intutiva de calcular el factorial es no recursiva, sino usando un bucle tradicional como vamos a ver ahora. Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo(numero) { alert ('El factorial de 5 es '+factorialIterativo(numero)); }
function factorialIterativo(numero){ var factorial = 1;
//Factorial de numero
for (var i=numero; i>0; i--){ factorial = factorial * (i); }
return factorial;
}
</script>
</script></head>
<body onload="ejemplo(5)" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Ahora vamos a plantear el factorial de forma recursiva. El factorial de un número se puede ver como factorial de n = n * (factorial de n-1)

Así el factorial de 5 podríamos verlo como 5 * (factorial de 4), que a su vez es 5 * 4 * (factorial de 3), que a su vez es 5 * 4 * 3 * (factorial de 2) … ¿Cuándo termina este proceso recursivo? Cuando llegamos a un caso en el que el resultado es elemental, es decir, un caso donde no es necesario utilizar factorial de n = n * (factorial de n-1), sino simplemente factorial de n = resultado elemental. Este caso, que se denomina caso base, para el factorial lo tenemos con el 1: factorial de 1 es 1, caso elemental.

El código recursivo para el factorial sería:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo(numero) { alert ('El factorial de '+numero+' es '+factorialRecursivo(numero)); }

function factorialRecursivo(numero){
if (numero==0) {return 1;} //caso base
else {return numero*factorialRecursivo(numero-1);} //caso recursivo
}

</script></head>
<body onload="ejemplo(5)" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Los pasos que da el factorial recursivo son:

5 * factorialRecursivo(4) -- > 5 * 4 * factorialRecursivo(3) -- > 5 * 4 * 3 * factorialRecursivo(2) -- >

5 * 4 * 3 * 2 * factorialRecursivo(1) y llega al caso base, devolviendo 5*4*3*2*1 = 120.

A la hora de plantear un proceso recursivo siempre es necesario tener presente la necesidad de que exista un caso base: si no es así, la recursión puede no terminar nunca y se entra en procesos que se asemejan a un bucle infinito. Cuando esto ocurre, en la consola de depuración de JavaScript nos podemos encontrar mensajes como “Too much recursion”.

EL PROBLEMA DEL SETTIMEOUT DENTRO DE BUCLES

Supongamos que queremos ejecutar un proceso de forma que entre cada ejecución haya un intervalo de 5 segundos y escribimos un bucle con un setTimeOut llamando a la función oportuna dentro del bucle:

```
while (contador<numeroNodos){
contador = contador+1;
referenciaTimer[contador] = setTimeout (cambiarNodo(nodosTituloCurso[contador]), 5000);
}
```

"La idea" es realizar el proceso cada 5 segundos, sin embargo hay un fallo de planteamiento que hace que esto no funcione: lo que ocurre en realidad es que se ejecuta el bucle en todas sus iteraciones de forma prácticamente instantánea, lo que da lugar a que el setTimeout no se ejecute.

Una función para detenerse tiene que llamarse a sí misma o utilizar algún mecanismo que fuerce la detención.

Vamos a analizar esto con mayor detenimiento. Para ello escribe y ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
*{font-family: verdana, sans-serif;}
.nodoNuevo{background-color: black; color:white; width:100px;
text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">
var contador = 1;
function mostrarNumConRetardo() {
setTimeout(crearNodo, 1000);
}
function crearNodo() {
var nodoHijo = document.createElement("div");
nodoHijo.className="nodoNuevo";
nodoHijo.innerHTML = ''+contador;
document.body.appendChild(nodoHijo);
contador = contador+1;
}
</script>
</script></head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Con este código hacemos que se muestre el número 1 por pantalla después de 1 segundo. Hasta aquí todo correcto. Ahora supongamos que queremos que se cuente hasta 10 y lo hacemos con un bucle. Para ello modificamos la función mostrarNumConRetardo que queda así:

```
function mostrarNumConRetardo() {
for(var i=0; i<10; i++){
setTimeout(crearNodo, 1000);
}
}
```

El resultado no es que se vayan mostrando progresivamente 1, luego 2, luego 3, luego 4... sino que se nos muestran de golpe todos los números: 1, 2, 3,4, 5, 6, 7, 8, 9, 10. ¿Por qué? Porque el bucle se ejecuta de forma inmediata quedando 10 órdenes de que se muestren dentro de 1 segundo los números, cuando transcurre un segundo se ejecutan todas las órdenes.

Para solucionar este problema tenemos que forzar la detención y que la siguiente llamada se ejecute después de que el retardo previo haya tenido lugar. Hay varias maneras de solucionar esto. Aquí vamos a mostrar una de ellas, basada en llamadas recursivas.

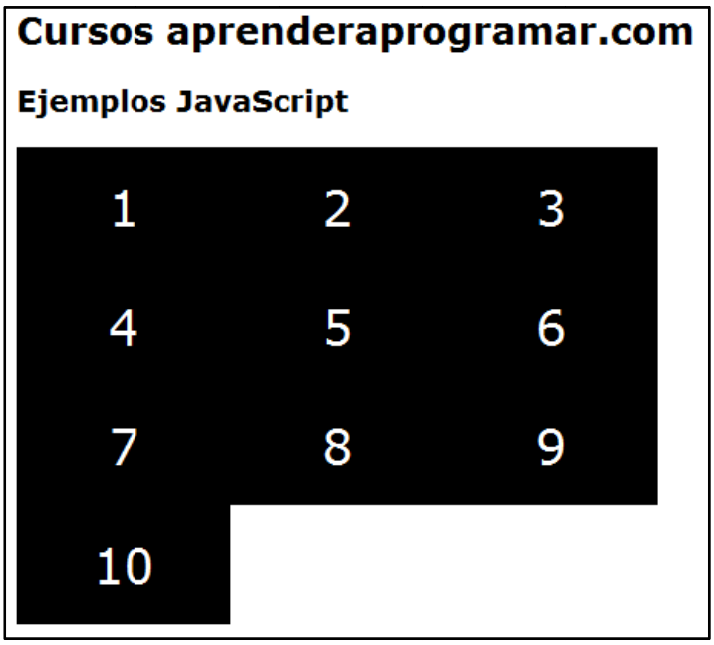
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
		*{font-family: verdana, sans-serif;}
		.nodoNuevo{background-color: black; color:white; width:100px;
		text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">
var contador = 1;
function mostrarNumConRetardo() { setTimeout(crearNodo, 1000);}
function crearNodo() {
		if (contador==11) {
		//Caso base: fin de la recursión (no hacemos nada)
		} else { //Caso recursivo
		var nodoHijo = document.createElement("div");
		nodoHijo.className="nodoNuevo";    nodoHijo.innerHTML = ''+contador;
		document.getElementById('cabecera').appendChild(nodoHijo);       contador = contador+1;
		setTimeout(crearNodo, 1000);
		}
}
</script>
</script>
</head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera" style="width:500px;">
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplos JavaScript</h3></div>
</body>
</html>
```

En este código observamos lo siguiente:

a) No existe un bucle como existía en el ejemplo puesto anteriormente. Lo que existen son llamadas recursivas que “simulan” un bucle (de hecho, llevamos un contador).

b) Cada llamada recursiva introduce un retardo. De este modo, la siguiente ejecución no comienza si no se ha producido el retardo. A diferencia del caso de un bucle, donde la ejecución es instantánea, en este caso la ejecución va siendo progresiva: no comienza la siguiente recursión hasta que pasa el tiempo de retardo especificado.

El resultado esperado es que vayan apareciendo en pantalla progresivamente los números 1, 2, 3, … hasta 10, con un pequeño retardo en la aparición de cada número.



CREAR EFECTOS JAVASCRIPT: HASTA DONDE LA IMAGINACIÓN ALCANCE...

Con las herramientas que hemos visto a lo largo del curso podemos crear potentes animaciones con JavaScript. Es cierto que existen frameworks JavaScript y APIs que nos facilitan crear animaciones, pero nosotros mismos podemos plantear animaciones usando puro código JavaScript hasta donde la imaginación nos alcance...

Escribe el siguiente código y comprueba los resultados de ejecución. Este código corresponde a una página web real que lanzamos en el dominio http://aprendeaprogramar.es. El código es sencillo, puede resultar un poco extenso para estudiar pero merece la pena, ya que aborda diferentes materias que hemos ido estudiando a lo largo del curso como acceso y manipulación del DOM, creación de efectos y retardo con setTimeOut, recursión, cambio de estilos CSS, uso de variables de distintos tipos, etc.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
<title>Cursos aprende a programar</title>
<meta charset="utf-8">
<meta name="description" content="Aprende a programar con cursos reconocidos por su calidad didáctica">
<meta name="keywords" content="HTML, CSS, JavaScript, Java, Visual Basic, Joomla, PHP">
<style type="text/css">
*{font-family: verdana, sans-serif;}
#principal{text-align:center;width:95%; margin: 0 auto;}
.tituloCurso {color: white; float:left; padding: 36px 44px; font-size: 2.65em; font-style:bold; text-decoration:none;}
a:hover{color:orange !important;}
</style>
<script type="text/javascript">
function ejemplo(){ var nodosTituloCurso = document.querySelectorAll(".tituloCurso");
var contador = 0; var nodosCambiados = new Array(); var tocaCambiar = true;
setTimeout(function() {ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar)}, 1500);
}
```

```
function ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar) {
	if (contador<nodosTituloCurso.length){
	var indice = Math.floor((Math.random() * (nodosTituloCurso.length)));
	if (nodosCambiados.length!=0) {
		for (var i=0; i<nodosCambiados.length; i++) {
			if(nodosCambiados[i]==indice) {tocaCambiar = false;}
		}
	}
	if (tocaCambiar==true) {
	cambiarNodo(nodosTituloCurso[indice]);
	nodosCambiados.push(indice);
	contador = contador+1;
	setTimeout(function() {ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar)}, 1500);
	} else {tocaCambiar=true; ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar);}
	}
	else { //Caso base fin de la recursión
		document.body.style.backgroundColor='black';
		document.getElementById('principal').style.color='white';
		for (var i=0; i<nodosTituloCurso.length; i++) {
			nodosTituloCurso[i].style.color='yellow';
		}
	}
}

function cambiarNodo(elNodo){ elNodo.style.backgroundColor = 'black'; }
</script>
</head>
<body onload="ejemplo()">
<div id="principal"><h1>Cursos de programación</h1>
<h3>Reconocidos por su calidad didáctica</h3>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=28&Itemid=59">
Fundamentos</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188">
Java</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=69&Itemid=192">
HTML</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=75&Itemid=203">
CSS</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206">
JavaScript</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=70&Itemid=193">
PHP</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=38&Itemid=152">
Joomla</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=37&Itemid=61"> Visual Basic</a></div>
<div ><a class="tituloCurso" href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=28&Itemid=59">
Pseudocódigo</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=section&layout=blog&id=7&Itemid=26">
Libros/ebooks</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=64&Itemid=87">Cursos tutorizados</a></div>
</div></body></html>
```

EJERCICIO 1

Realiza una descripción de cada una de las instrucciones del código del ejemplo anterior indicando cuál es su cometido. Si has seguido el curso desde el principio, debes ser capaz de interpretar todo el código.

Ejemplo: La primera línea <!DOCTYPE ... realiza la declaración de tipo de documento HTML a efectos de que los navegadores interpreten en qué versión de HTML está escrito el código.

La etiqueta html es la etiqueta de apertura del código HTML. La etiqueta head es ...

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea una animación JavaScript (decide tú el efecto que quieres implementar: pueden ser cosas moviéndose por la pantalla, texto cambiando de tamaño o cualquier otra cosa que se te ocurra) con las herramientas que hemos ido conociendo a lo largo del curso. Ejecútalo en distintos navegadores y comprueba que se ejecute tal y como esperas en todos ellos.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01167E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

TYPEOF JAVASCRIPT. SABER TIPO DE VARIABLE. GLOBAL Y LOCAL: AMBITO (SCOPE). VAR CAMBIA EL AMBITO. EJEMPLO CUENTA ATRÁS (CU01167E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº67 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

AMBITO (SCOPE) JAVASCRIPT

Cuando trabajamos con JavaScript podemos definir varios ámbitos: ámbito global (variables conocidas por todas las funciones y por el espacio global) y ámbitos locales (variables conocidas únicamente en determinadas partes del código).

OPERADOR TYPEOF

Existe un operador que nos va a resultar útil para determinar el tipo de una variable, o para determinar si esta variable existe en el ámbito en el que tratamos de utilizarla. La sintaxis para usar este operador es la siguiente:

```
typeof expresiónAEvaluar ó typeof (operandoAEvaluar)
```

El resultado de aplicar este operador a un identificador JavaScript es una cadena de texto que puede puede ser: string (si el tipo de expresiónAEvaluar es cadena de texto), number (si el tipo es numérico), boolean (si el tipo es booleano o se pasan las palabras clave true ó false) object (si el tipo es un objeto), function (si el tipo es una función o un método) o undefined si el identificador pasado no existe en el ámbito en que se trata de utilizar.

En este ejemplo vemos cómo obtener tipos conocidos de variables

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var miString = 'cosa';
var miNumeroEntero = 33;
var miNumeroDecimal = 23.65;
var miBooleano = true;
var miFuncion = new function(a, b) {return (a+b);}
msg = 'Tipo de miString: '+typeof miString + '\nTipo de miNumeroEntero: '+typeof miNumeroEntero;
msg = msg + '\nTipo de miNumeroDecimal: '+typeof miNumeroDecimal + '\nTipo de miBooleano: '+typeof miBooleano;
msg = msg + '\nTipo de miFuncion: '+typeof miFuncion + '\nTipo de Math: '+typeof Math;
msg = msg + '\nTipo de Date: '+typeof Date + '\nTipo de algoIndefinido: '+typeof algoIndefinido;
if (typeof miNumeroEntero == 'string') {msg = msg + 'miNumeroEntero es de tipo string';}
else {msg = msg + '\nmiNumeroEntero no es de tipo string';}
alert(msg);
}
</script>
</script></head>
<body onload="ejemplo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

AMBITO DE VARIABLES: GLOBAL Y LOCAL

La primera aproximación al concepto de ámbito (scope) va a consistir en distinguir entre las variables declaradas fuera de cualquier función, a las que se denomina variables globales y serán conocidas por todas las funciones, frente a variables declaradas dentro de una función concreta, que sólo serán conocidas dentro de esa función. Los parámetros pasados a funciones funcionan como si fueran variables locales a la función.

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

var idGlobal=33;
var msg = 'Variable global';

function ejemplo(numero) {
var miVarLocal = 'Soy una variable local';
var msg = 'idGlobal es conocido en la función ejemplo. idGlobal vale: '+idGlobal;
msg = msg + '\n\nEn ejemplo el valor de numero es recibido como parametro (local). numero es: '+numero;
msg = msg + '\n\n La variable local miVarLocal contiene: '+miVarLocal;
alert(msg); //msg es local a esta función, aunque exista otra variable con este nombre se reconoce sólo la más próxima
ejemplo2();
}

function ejemplo2(){
var msg = 'idGlobal es conocido en la función ejemplo2. idGlobal vale: '+idGlobal;
//msg = msg + '\n\nEjemplo2 desconoce qué es numero porque es local a otra función: aquí numero es '+numero;
//msg = msg + '\n\nEjemplo2 desconoce qué es miVarLocal porque es local a otra función: aquí miVarLocal es
'+MiVarLocal;
if (typeof miVarLocal != 'undefined') { msg=msg+'\n\nEjemplo2 desconoce qué es miVarLocal porque es local a otra
función: aquí miVarLocal es '+miVarLocal }
else {msg=msg+'\n\nmiVarLocal No existe en el ámbito de la función ejemplo2\n\n';}
alert(msg); //msg es local a esta función
}

</script>
</script></head>
<body onload="ejemplo(5)" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

El resultado esperado es:

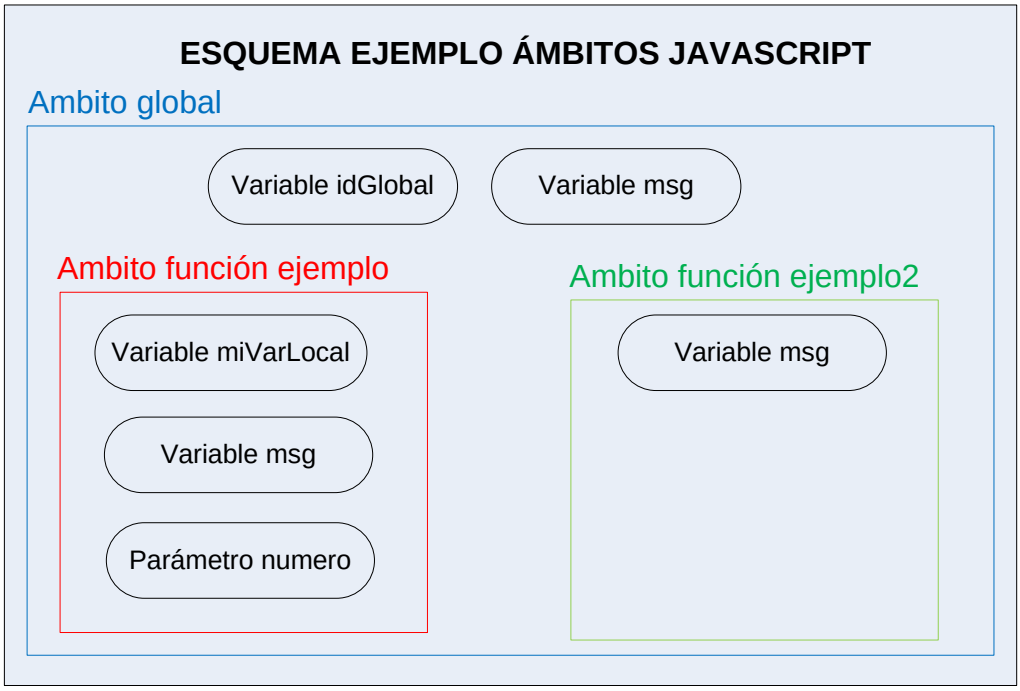
idGlobal es conocido en la función ejemplo. idGlobal vale: 33
En ejemplo el valor de numero es recibido como parametro (local). numero es: 5
La variable local miVarLocal contiene: Soy una variable local
idGlobal es conocido en la función ejemplo2. idGlobal vale: 33
miVarLocal No existe en el ámbito de la función ejemplo2

La variable global idGlobal es conocida por todas las funciones. De acuerdo con esto, la variable msg también es conocida por todas las funciones. Sin embargo, al declararse una variable local en las funciones con igual nombre, decimos que la variable global queda "tapada" por la variable local, es decir, al invocar dicha variable dentro de la función primeramente se busca si existe en el ámbito de dicha función (como local) y si es así, se devuelve dicha variable ignorando las que puedan existir en ámbitos externos. En resumen: siempre se busca si un identificador existe en el ámbito en que se invoca, si no es así se trata de localizar en el siguiente ámbito más próximo y así sucesivamente hasta llegar al ámbito global. Si no existe en el ámbito global y se trata de utilizar se obtiene un error de tipo << ReferenceError: MiVarLocal is not defined>>. Recordar que los errores no se muestran directamente en pantalla, sino que hemos de activar la consola de depuración del navegador para poder visualizarlos.

En el código aparecen dos líneas comentadas que si tratamos de ejecutar nos devuelven un error, ya que estamos tratando de invocar identificadores de variables o parámetros desconocidos en ese ámbito.

La sintaxis: if (typeof miVarLocal != 'undefined') nos permite determinar si el identificador miVarLocal es conocido en el ámbito. Podríamos también haber invertido la lógica escribiendo if (typeof miVarLocal == 'undefined') y cambiando el orden de las sentencias a ejecutar.

Gráficamente podríamos ver los ámbitos como espacios donde las variables son conocidas:



IMPLICACIONES DEL USO DE VAR EN EL ÁMBITO DE LAS VARIABLES

Nosotros estamos trabajando declarando una variable con la palabra clave var. Por ejemplo var miNumero = 33;

Sin embargo, el uso de var no es obligatorio, y el intérprete considera que una asignación de contenido a una variable que no ha sido declarada es una declaración "implícita". Por ejemplo podríamos escribir directamente miNumero=33; alert(miNumero) ; sin hacer uso de la palabra clave var.

Cuando no se escribe la palabra clave var dentro de una función tiene una implicación: la variable automáticamente es tratada como una variable global aún a pesar de estar dentro de una funcion. Ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function ejemplo() {miVarLocal = 'Soy una variable local'; ejemplo2();}

function ejemplo2(){
var msg= '';
if (typeof miVarLocal != 'undefined') { msg=msg+'\n\nEjemplo2 conoce qué es miVarLocal porque se'+
' declaró en otra función sin var: aquí miVarLocal es '+miVarLocal+ ' y es de tipo '+(typeof miVarLocal) }
else {msg=msg+'\n\nmiVarLocal No existe en el ámbito de la función ejemplo2\n\n';}
alert(msg); //msg es local a esta función
}
</script>
</script></head>
<body onload="ejemplo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

El resultado, que podemos ver como a priori poco lógico, es el siguiente:

<<Ejemplo2 conoce qué es miVarLocal porque se declaró en otra función sin var: aquí miVarLocal es Soy una variable local y es de tipo string>>

Esto nos da pie a que podamos plantear las siguientes situaciones y resultados:

Forma de declaración	Lugar de declaración	Efecto
Implícita sin uso de var	Fuera de cualquier función	La variable es global
Usando var	Fuera de cualquier función	La variable es global
Implícita sin uso de var	Dentro de una función	La variable es global
Usando var	Dentro de una función	La variable es local

En general recomendaremos hacer uso siempre de var, de forma que las variables declaradas dentro de funciones trabajen como variables locales. Esto hace la programación más segura y evita saturar el espacio global de nombres que en caso de repetición pueden crear conflictos.

Tener en cuenta que si una variable como mensaje es declarada como variable global y luego es utilizada dentro de una función sin hacer uso de var, su valor queda modificado al considerarse que se está haciendo referencia a la variable global (única) aunque nuestra intención no fuera esa. Esto refuerza la recomendación de hacer siempre uso de var para declarar variables.

No deben declararse indiscriminadamente variables sin uso de var dentro de funciones porque eso podría calificarse como "abuso de variables globales" que a la larga traerá complicaciones. Las variables globales deben ser usadas de forma controlada y en su justa medida.

EJERCICIO

Analiza este código JavaScript y responde a las preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var target_date = new Date("Jan 1, 2089").getTime();
var days, hours, minutes, seconds;
function ejemplo() {
var countdown = document.getElementById("countdown");
setInterval(function () {
   var current_date = new Date().getTime();
   var seconds_left = (target_date - current_date) / 1000;
   days = parseInt(seconds_left / 86400);
   seconds_left = seconds_left % 86400;
   hours = parseInt(seconds_left / 3600);
   seconds_left = seconds_left % 3600;
   minutes = parseInt(seconds_left / 60);
   seconds = parseInt(seconds_left % 60);
   countdown.innerHTML = 'Para el 1 de enero de 2089 faltan '  +days+ ' días, ' + hours + " horas, "
   + minutes + " minutos, " + seconds + " segundos";
            }, 1000);
}
</script>
</head>
<body onload="ejemplo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<span id="countdown"></span></div>
</body></html>
```

a) Haz una lista de las variables que intervienen indicando para cada variable cuál es su nombre, cuál es su tipo, cuál es su cometido y si está definida como variable global o variable local.

Ejemplo: la variable target_date es de tipo Date, su cometido es almacenar la fecha futura respecto de la cual el script va a mostrar los días, horas, minutos y segundos que faltan para alcanzar dicha fecha, y está definida como variable global.

b) ¿Qué ocurre si definimos la variable current_date en el ámbito global en vez de dentro de la función? ¿Por qué ocurre esto?

c) ¿Podríamos definir todas las variables como locales a la función y prescindir de las variables globales? Si se pudiera hacer, ¿crees que sería positivo para el diseño del código, su mantenimiento y ampliabilidad, o por el contrario, que sería negativo?

d) ¿Qué métodos de los empleados en el código devuelven valores en milisegundos?

e) Razona la lógica del código. Con los contenidos que hemos visto en el curso hasta el momento, debes ser capaz de entender todo el código que aparece en el script.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01168E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

THIS JAVASCRIPT: SIGNIFICADOS. AMBITOS (SCOPE). ANIDAMIENTO. NAMESPACES. EJEMPLO EJERCICIO RESUELTO. (CU01168E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº68 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MÁS SOBRE ÁMBITOS (SCOPES)

Hemos hablado básicamente de dos ámbitos: el global, cuyas variables son definidas por todas las funciones, y el local, correspondiente a una función. No obstante, los ámbitos son anidables en el sentido de que una función se puede definir dentro de otra.

AMBITOS INTERNOS

Una función se puede definir dentro de otra. En este caso, cada función define un ámbito local a ella misma.

La regla para saber si una variable es visible es: todo variable definida en un ámbito externo (que envuelve a otro) es conocida en un ámbito interno. Por el contrario, las variables definidas en un ámbito interno no son conocidas en ámbitos externos a dicho ámbito.

Ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() { //Ambito de ejemplo
var combustible=40;
var llenarDeposito = function() {
//Ambito de llenarDeposito (interno a ejemplo)
alert('Se han introducido en el depósito '+combustible+' litros');
var faltaParaLlenado = 200-combustible;
}
llenarDeposito();
if (typeof faltaParaLlenado == 'undefined') {alert('Aquí no se conoce la variable faltaParaLlenado');}
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es: <<Se han introducido en el depósito 40 litros>> y <<Aquí no se conoce la variable faltaParaLlenado.>>

En resumen, las funciones se pueden definir dentro de funciones, y cada función define un ámbito interno no conocido por las funciones externas.

En cambio, las variables definidas por funciones externas sí son conocidas por las funciones internas. Por ejemplo, la variable combustible definida en la función ejemplo sí es conocida dentro de la función interna referenciada por llenarDeposito.

ESPACIOS DE NOMBRES (NAMESPACES) JAVASCRIPT

Cuando se crea código donde se tienen decenas de archivos JavaScript y miles de líneas, es probable que se den colisiones de nombres: dos variables que toman el mismo nombre, ó dos funciones que toman el mismo nombre, etc. Una colisión de nombres no siempre es negativa en el sentido de que si está bien resuelta y planteada, no tiene por qué generar un conflicto. No obstante, con frecuencia se producen colisiones con efectos indeseados. Veámoslo con un ejemplo: partimos de este código.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var capacidadDeposito = 200;
function obtenerNecesidades(contenidoActual){
alert('La capacidad del depósito es ' + capacidadDeposito + ' litros y faltan ' + (capacidadDeposito-contenidoActual) +
' litros para llenarlo');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="obtenerNecesidades(20)"> Probar </div>
</body></html>
```

El resultado esperado al hacer click es: <<La capacidad del depósito es 200 litros y faltan 180 litros para llenarlo>>

Pero supongamos que es una aplicación web y que al cabo de un par de años, cuando esta aplicación tiene miles de líneas, otro programador introduce otra función que tiene el mismo nombre que la que nosotros habíamos definido antes. El código suponemos que hubiera quedado así:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var capacidadDeposito = 200;
//Aquí muchas líneas
function obtenerNecesidades(contenidoActual){
alert('La capacidad del depósito es ' + capacidadDeposito + ' litros y faltan ' + (capacidadDeposito-contenidoActual) +
' litros para llenarlo');
}
// Aquí muchas líneas
function obtenerNecesidades(contenidoActual) {
var capacidadDeposito = 300;
alert('La capacidad del depósito es ' + capacidadDeposito +
' litros y faltan ' + (capacidadDeposito-contenidoActual) + ' litros para llenarlo');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="obtenerNecesidades(20)"> Probar </div>
</body></html>
```

Ahora el resultado obtenido al hacer click es: << La capacidad del depósito es 300 litros y faltan 280 litros para llenarlo>>

Quizás la capacidad del depósito de un nuevo modelo de vehículo sea de 300 litros, pero nuestro código referido a otro modelo de vehículo ha dejado de funcionar debido a una colisión de nombres: la función que se invoca al hacer click es la última que el intérprete lee dentro del código, y al haber dos funciones con el mismo nombre una queda "tapada" por la otra.

Una forma de tratar de mitigar estos efectos indeseados es definir spacenames o espacios de nombres.

Los espacios de nombres son objetos que actúan a modo de contenedor para envolver todo un conjunto de propiedades, funciones, etc. de forma que su identificación sea más segura y la probabilidad de conflicto de nombres sea baja.

Para crear un espacio de nombres podemos usar esta sintaxis, que ya habíamos explicado como forma de crear objetos en JavaScript:

```
    var nombreObjetoCreado = {
        propiedad1: valorPropiedad1,
        propiedad2: valorPropiedad2,
        propiedadN: valorPropiedadN,
        ...
        método1: function () { ... código ... }
        método2: function (par1, par2, ..., parN) { ... código ... }
        métodoN: function () { ... código ... }
}
```

En nuestro caso, nombreObjetoCreado será el nombre del espacio de nombres. Este nombre será "un prefijo" que habrá que aplicar para invocar cualquier variable (propiedad) o método de ese espacio de nombres.

El prefijo puede ser relativo a la funcionalidad que tiene el código, o puede ser un prefijo que usemos nosotros para identificarnos como programadores, un prefijo creado para desarrollar una aplicación, etc.

Ejecuta este código y comprueba los resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

var modeloFordSpace = {
capacidadDeposito: 200,
obtenerNecesidades: function (contenidoActual){
alert('La capacidad del depósito es ' + modeloFordSpace.capacidadDeposito + ' litros y faltan ' +
 (modeloFordSpace.capacidadDeposito-contenidoActual) +
' litros para llenarlo');}
}
```

```
var modeloToyotaSpace = {
obtenerNecesidades: function(contenidoActual) {
var capacidadDeposito = 300;
alert('La capacidad del depósito es ' + capacidadDeposito +
' litros y faltan ' + (capacidadDeposito-contenidoActual) + ' litros para llenarlo');}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="modeloFordSpace.obtenerNecesidades(20)"> Probar (ford) </div>
<div style="color:blue;" id ="pulsador" onclick="modeloToyotaSpace.obtenerNecesidades(20)"> Probar (toyota) </div>
</body></html>
```

Con esta alternativa de diseño usamos a modo de variables propiedades de un objeto y a modo de funciones métodos de un objeto, sirviéndonos dicho objeto para crear el espacio de nombres. En este ejemplo tenemos dos funciones con el mismo nombre, pero al estar en distintos espacios de nombres podemos usar uno u otro sin problemas, simplemente empleando el prefijo adecuado.

Una sintaxis que puede resultar interesante es usar var espacioDeNombres = {}; para crear el espacio de nombres y posteriormente definir sus propiedades y métodos por separado. Ejecuta este ejemplo y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var modeloToyotaSpace = {};
modeloToyotaSpace.cocheToyota = function(modelo, capacidadDeposito) {
this.capacidadDeposito = capacidadDeposito;
this.modelo = modelo;
alert('Creado coche Toyota modelo ' + this.modelo + ' con capacidad '+this.capacidadDeposito);
this.obtenerVelocidad = function(){ if (capacidadDeposito<200) {return '150 km/h';} else {return '240 km/h';}}
this.fabrica = 'Ken-Zhuan';
}
function ejemplo() {
var coche1 = new modeloToyotaSpace.cocheToyota('Auris', 175);
alert('La velocidad máxima de este vehículo es ' + coche1.obtenerVelocidad());
alert('Vehiculo fabricado en la fábrica de ' + coche1.fabrica);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

LA PALABRA CLAVE THIS

Ya hemos hablado sobre el significado de la palabra clave this, que en JavaScript no siempre es el mismo. Vamos a tratar de resumir a qué puede hacer referencia la palabra clave this y repasar este concepto:

1. this para referirnos a un nodo del DOM (ya estudiado)

Un ejemplo lo veríamos aquí:

```
<h2 onclick="alert(this.nodeName)">Haz click aquí para ver el nodeName de this</h2>
<h2>Haz click <span onclick="alert(this.nodeName)" style="color:red;">aquí para ver el nodeName de this</span></h2>
```

En este caso al hacer click en el primer caso obtenemos H2 y en el segundo SPAN porque this alude al nodo dentro del cual estamos haciendo uso de esta palabra clave.

Aquí this (usada dentro de un manejador de evento onclick) nos devuelve el nodo de tipo Element definido por las etiquetas HTML donde se define el evento.

Generalizando, podemos decir que dentro de la función de respuesta a un evento, la palabra clave this hace referencia al elemento que es quien recibe (objetivo o target) el evento.

Este código refleja lo que hemos comentado:

```
window.onload = function () {
var elemsH = document.querySelectorAll("h1, h2, h3, h4, h5, h6");
for (var i=0;i<elemsH.length;i++) {
elemsH[i].addEventListener("mouseover", cambiarColor1);
elemsH[i].addEventListener("mouseout", cambiarColor2);}
function cambiarColor1() { this.style.color = 'orange';}
function cambiarColor2() { this.style.color = 'brown';}
}
```

Aquí las funciones de respuesta al evento son cambiarColor1 y cambiarColor2. Y dentro de estas funciones, this hace referencia al nodo del DOM (elemento) que recibe el evento.

2. this para referirnos a propiedades y métodos de un objeto (ya estudiado)

Lo aplicamos cuando construimos objetos de esta forma:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = function () { ... código .... }
this.método2 = function (param1, param2, ..., paramN) { ... código ...}
}
```

Aquí this sirve para aludir al objeto del cual estamos definiendo propiedades y métodos. This nos permite desambiguar los nombres y por ejemplo podemos tener como nombre de parámetro edad y establecer this.edad = edad; quedando así claro que this.edad alude a la propiedad del objeto y edad al parámetro recibido.

3. this para referirnos a un objeto especificado cuando usamos call y apply (ya estudiado)

La función call permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, así como los parámetros adicionales que sean necesarios.

Lo vemos en este ejemplo:

```
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar() { alert ('Hola, soy ' + this.nombre); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar.call(unProfesor);
}
```

Aquí al invocar call sobre la función saludar, se indica que el objeto que actuará como this será el objeto unProfesor. Pero podría haber sido otro objeto, el que nosotros hubiéramos querido.

Las funciones call y apply son muy similares, difieren tan solo en cómo pasan los parámetros a la función invocada (como una lista de items separados por comas con call ó como un array con apply).

4. this por defecto

Si no usamos this en ninguno de los contextos anteriores, ¿qué es this? This por defecto es el objeto dentro del cual se invoca. Si no se invoca dentro de ningún objeto, el objeto por defecto es el objeto global window.

Ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
alert ('Aquí this es ' + this);
var miObjeto = {};
miObjeto.habla = function() {alert('Aquí ahora this es ' + this);}
miObjeto.habla();
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es <<Aquí this es [object Window]>> y <<Aquí ahora this es[object Object]>>

Ten en cuenta que dentro de ejemplo this no es el nodo del DOM porque ejemplo no es la función de respuesta al evento. En realidad la función de respuesta al evento podemos decir que es una función cuyo contenido es ejemplo(), es decir, una función que da lugar a la ejecución de la función ejemplo, pero esta función no es la función de respuesta al evento.

PERDER EL THIS

Toda función define un ámbito y una función dentro de otra función puede hacer que this no se refiera a lo mismo según dónde lo usemos. Por ejemplo, si tenemos una función anónima dentro de otra, this en la función externa puede estar haciendo referencia al objeto envolvente mientras que this en la función anónima puede estar haciendo referencia al objeto global window.

La solución para mantener una referencia a this en una función anónima interna puede estar en crear un closure. Definiríamos como variable local a la función externa var that = this;, y luego en la función anónima interna haríamos referencia a that, variable auxiliar que nos sirve para mantener la referencia deseada.

En el ejercicio propuesto a continuación veremos un ejemplo de esta situación.

RESUMEN SOBRE THIS

Hemos visto distintas acepciones de this en JavaScript y posiblemente todavía nos falten otras acepciones y nos resulte difícil manejar este concepto y su importancia. No te preocupes ahora de entender todo lo relacionado con este concepto. Continúa avanzando con el curso, irás adquiriendo destreza en el manejo de estas ideas a medida que desarrolles más código y te enfrentes a situaciones variadas.

EJERCICIO

Analiza el siguiente código y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
var cabecera = document.querySelector('#cabecera');
var respuestaCabecera = function () {
  alert(this);
  setTimeout(function () { alert(this);}, 2000);
};
cabecera.addEventListener('click', respuestaCabecera, false);
}
</script>
```

```
</head>
<body onload="ejemplo()"><div id="cabecera"><h2>Cursos aprenderaprogramar.com HAZ CLICK
AQUÍ</h2><h3>Ejemplos JavaScript</h3></div>
</body>
</html>
```

a) ¿Por qué se muestran diferentes mensajes si en ambos alert estamos invocando this?

b) Modifica el código para que el mensaje que se muestre con retardo muestre lo mismo que el mensaje que se muestra sin retardo. Para ello, haz que en la función anónima sea conocida la referencia a this que existe en la funcion externa.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01169E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

CLOSURES JAVASCRIPT. EJEMPLOS. CONCEPTO: QUÉ SON Y PARA QUÉ SIRVEN. RETARDO DE EJECUCIÓN CON SETTIMEOUT Y CLOSURES. (CU01169E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº69 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CLOSURES JAVASCRIPT

Los closures JavaScript son funciones que llevan información asociada relativa al momento en que son invocadas. No es fácil explicar ni entender el concepto de closure porque éste es un tanto complejo. Recomendamos que se estudie viendo ejemplos y se vaya asimilando poco a poco a medida que se practique con el desarrollo de código JavaScript.

Los closures, en castellano denominados cierres, cerraduras o clausuras, son una característica de algunos lenguajes entre los que se encuentra JavaScript.

Un closure se genera cuando se produce la siguiente situación en el código:

```
    function funcionExterna(par1, par2, ..., parN){
        var miVariableLocal = un valor;
        var miFuncionInterna = function () {
                return par1 - miVariableLocal; // Situación que genera el closure
        }
        return miFuncionInterna ó miFuncionInterna(); // Veremos ejemplos para entenderlo
}
```

El closure se genera cuando una función interna a otra función usa una variable local (o parámetro recibido) de la función externa. Con un ejemplo lo veremos más claro:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeMsg(nombre){
        var msg = 'hola';
        var respuesta = function () { alert(msg+ ' ' + nombre); }
        respuesta()
}

function ejemplo(){ var habla1 = dimeMsg('Juan'); }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla <<Hola Juan>>

La función interna también la podemos definir así (siendo equivalente al código anterior):

```
function dimeMsg(nombre){
        var msg = 'hola';
        function respuesta () {  alert(msg+ ' ' + nombre); }
        respuesta()
}
```

A las funciones internas que hacen uso de variables locales de las funciones externas dentro de las cuales se encuentran las denominamos cerraduras o closures. Una cerradura tiene unas particularidades que trataremos de estudiar a continuación. Ten en cuenta que los closures a veces se generan "intencionadamente" y otras veces se generan "sin querer". Pero de una forma u otra, conviene entender qué implica que exista un closure para poder entender lo que ocurre en muchos scripts.

El código anterior parece que tiene poco interés, pero veamos cómo los closures tienen características interesantes.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeVuelta(entrega){
        var precio = 1000;
        var respuesta = function () { return precio-entrega; } //Aquí el closure
        return respuesta;
}

function ejemplo(){
        var calcula = dimeVuelta(600);
        alert('Su vuelta es ' +calcula());
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla: <<Su vuelta es: 400 >>

Analicemos el código. La referencia "respuesta" es una cerradura o closure, ya que es una función interna que utiliza variables locales de la función externa.

La función externa "dimeVuelta" devuelve como resultado una referencia a la función interna.

Al ejecutar var calcula = dimeVuelta(600); lo que se almacena en "calcula" es una referencia a la función interna. Es como si hiciéramos calcula = function () { return precio-entrega; } ¿Pero qué significado tendrían aquí precio y entrega, teniendo en cuenta que la función donde se definían ya se ejecutó y por tanto en principio están fuera de un ámbito válido? Teóricamente las variables globales a una función son destruidas cuando termina de ejecutarse la función, por tanto precio y entrega supuestamente deberían haber sido destruidas.

Sin embargo, cuando JavaScript encuentra un closure toda variable local que sea necesaria para el funcionamiento del closure queda encerrada en el propio closure. Es decir, dado que "respuesta" necesita de "precio" y "entrega", éstas se guardan dentro de la función cerradura.

Después de ejecutarse var calcula = dimeVuelta(600); en la variable "calcula" tenemos almacenados el precio (1000) y la entrega (600), aunque la función externa ya haya sido ejecutada.

Ahora calcula tiene una referencia a una función. Para ejecutar dicha función invocamos calcula(), y dado que esta función recuerda los valores de variables locales devuelve 400 (obtenidos de 1000-400, precio-entrega).

Ahora bien, ¿qué valor de variable local es el que almacena el closure? Tener en cuenta que una variable local puede cambiar a lo largo del código. Por ejemplo:

```
function dimeVuelta(entrega){
        var precio = 1000;
        var respuesta = function () { return precio-entrega; } //Aquí el closure
        precio = 700;
        return respuesta;
}
```

¿El closure quedará tomando como referencia 1000 ó 700? La realidad es que toma como referencia el valor que tenía la variable local cuando se produce la salida de la función externa (en este ejemplo justo antes del return), por tanto en este caso el closure queda almacenando como precio un valor de 700.

Podría darse la situación de que existan varias funciones internas a una función externa dada, y que varias de esas funciones internas usen variables locales de la función externa. En este caso decimos que se generan varias cerraduras (una por cada función interna que hace uso de variables locales), pero aquí sí es cierto que todas ellas quedan con una única referencia de variable local: la que exista cuando se produzca la salida de la función externa.

En el ejemplo anterior hemos usado la creación de una referencia intermedia para después invocar la función:

```
function ejemplo() {        var calcula = dimeVuelta(600);        alert('Su vuelta es ' +calcula()); }
```

Pero la invocación de la función podemos hacerla directamente si lo deseamos escribiendo esto:

```
function ejemplo() {  alert('Su vuelta es ' +dimeVuelta(600)()); }
```

Aquí dimeVuelta(600) nos devuelve la referencia a la función anónima, y al añadir () a continuación, damos pie a su ejecución directamente (sin necesidad de crear la referencia usando var).

Si escribes alert('Su vuelta es ' +dimeVuelta(600)); por pantalla obtendrás la función que devuelve la función invocada, por tanto se verá << Su vuelta es function () { return precio-entrega; }>> o un mensaje similar (puede variar según el navegador).

CADA CLOSURE LLEVA SUS DATOS

Cuando se invoca la función externa se genera un closure y cada closure que se genere guarda sus propias referencias, es decir, no se guarda una única referencia para todos los closures. Un closure es una combinación de función y de datos relativos al momento de su creación. En ese sentido podemos decir que recuerdan a los objetos de la programación orientada a objetos (objeto = datos + métodos closure = datos + función). Un closure sería un objeto con un solo método. Ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function dimeVuelta(entrega){            var precio = 1000;
            var respuesta = function () { return precio-entrega; } //Aquí el closure
            return respuesta;
}

function ejemplo(){ var calculo1 = dimeVuelta(600);          alert('Su vuelta es: '+calculo1());
                        var calculo2 = dimeVuelta(500);          alert('Su vuelta es: '+calculo2());
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado es <<Su vuelta es: 400>> y <<Su vuelta es: 500>> porque cada closure guarda la referencia al valor de las variables locales en el momento en que fueron creados.

El hecho de que cada closure guarde su información permite interesantes aplicaciones.

USAR CLOSURES PARA FUNCIONES RETARDADAS

Partimos del siguiente código con el que tratamos de hacer que mediante un bucle for se cuente de 1 a 10 con intervalos de 1 segundo entre que aparezca cada número en pantalla. Ejecuta el código y comprueba qué ocurre:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
*{font-family: verdana, sans-serif;}
.nodoNuevo{background-color: black; color:white; width:100px;
text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">

function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(crearNodo(i), 1000); }
}

function crearNodo(numero) {
var nodoHijo = document.createElement("div");
nodoHijo.className="nodoNuevo"; nodoHijo.innerHTML = ''+numero;
document.body.appendChild(nodoHijo);
}
</script>
</script></head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Lo que ocurre es que se muestra por pantalla de forma inmediata todos los números (1-2-3-4-5-6-7-8-9-10) sin retardo.

Podemos pensar en tratar de arreglarlo con la siguiente modificación:

```
function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(crearNodo(i), 1000*i); }
}
```

Pero esto no funciona. ¿Por qué? Porque el valor de i que se está pasando a setTimeout no es el valor de i en cada bucle, sino la referencia a la variable i cuando setTimeout se ha ejecutado y el bucle ha terminado, y esa referencia no tiene valor (ya que el bucle ha terminado).

Necesitamos que setTimeout “recuerde” el valor que tenía i en cada pasada del bucle. Esto lo podemos hacer creando un closure en cada pasada del bucle. Ejecuta este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">
*{font-family: verdana, sans-serif;}
.nodoNuevo{background-color: black; color:white; width:100px;
text-align:center; padding:20px; font-size:32px; float:left;}
</style>
<script type="text/javascript">
```

```
function mostrarNumConRetardo() {
for(var i=1; i<11; i++){ setTimeout(function(x) { return function() { crearNodo(x); }; }(i), 1000*i); }
}

function crearNodo(numero) {
var nodoHijo = document.createElement("div");
nodoHijo.className="nodoNuevo"; nodoHijo.innerHTML = ''+numero;
document.body.appendChild(nodoHijo);
}
</script>
</script></head>
<body onload="mostrarNumConRetardo()" >
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
</body></html>
```

Dentro del setTimeout invocamos la siguiente función:

function(x) { return function() { crearNodo(x); }; }(i)

Con esta sintaxis lo que hacemos es crear una función anónima que recibe un parámetro x y que devuelve otra función anónima interna que crea un closure ya que usa el parámetro que recibe la función externa. El closure recuerda el valor de la variable local (parámetro). Para ejecutar el closure, invocamos la función externa pasándole (i) como parámetro, siendo i el contador del bucle. Esto fuerza que setTimeout se ejecute con los valores que tenía i en cada pasada del bucle, y no con una única referencia a i.

RESUMEN SOBRE CLOSURES Y ÁMBITOS

Los closures son funciones que llevan datos asociados, relativos al momento en que fueron invocadas.

La existencia de closures aporta ventajas a la programación con JavaScript, ya que podemos usarlos para resolver necesidades que nos surjan. Pero también genera problemas: a veces se generan closures sin querer con efectos indeseados. O a veces se crea un excesivo número de closures innecesariamente, consumiendo recursos y haciendo más lenta la ejecución del código.

Los closures son una parte de la programación JavaScript que no es fácil de explicar ni de entender. Esto podemos extenderlo en general a “los ámbitos” y a la palabra clave this. No te preocupes si te has perdido en algunas partes de las explicaciones que hemos dado. Sigue avanzando con el curso y trata de ir adquiriendo destreza en la interpretación y uso de closures a medida que sigas programando JavaScript.

EJERCICIO

Analiza el siguiente código y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">

<style type="text/css">
body { font-family: Helvetica, Arial, sans-serif;}
h2 {  font-size: 1.5em;} h3 { font-size: 1.2em;}
div div {color:blue; margin:10px;}
</style>

<script type="text/javascript">
function cambiarDimensionFuente(size) {   return function() { document.body.style.fontSize = size + 'px';};
}

var size8 = cambiarDimensionFuente(8);
var size16 = cambiarDimensionFuente(16);
var size24 = cambiarDimensionFuente(24);

function setClicks(){
document.getElementById('fuente-8').onclick = size8;
document.getElementById('fuente-16').onclick = size16;
document.getElementById('fuente-24').onclick = size24;
}

</script></head>
<body onload="setClicks()">
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<div id ="fuente-8" > Poner texto a 8 </div> <div id ="fuente-16" > Poner texto a 16 </div>
<div id ="fuente-24" > Poner texto a 24 </div> </div>
<p>En las praderas de la estepa de la Tierra del Fuego suele hacer frío</p>
</body></html>
```

a) ¿En qué parte del código se genera un closure o cerradura? ¿Por qué?

b) ¿En qué parte del código se establece que al hacer click sobre el elemento con id fuente-8 se cambie el tamaño de las fuentes de la página?

c) Supón que eliminamos la función setClicks y dejamos su código “libre” dentro de las etiquetas <script> … </script>. ¿Qué mensaje de error te muestra la consola de depuración? (Activa la consola si no la tienes activada) ¿Por qué aparece ese mensaje de error?

d) ¿Debemos escribir document.getElementById('fuente-8').onclick = size8; ó document.getElementById('fuente-8').onclick = size8(); ?¿Por qué?

e) Supón que al cargar la página queremos que el tamaño inicial de fuente sea 8 y para ello nos valemos de la función setClicks. ¿Debemos escribir dentro de esta función size8; ó size8();? ¿Por qué?

f) Las closures no siempre son necesarias, incluso a veces se generan involuntariamente o innecesariamente consumiendo recursos del sistema que podrían ahorrarse. ¿Qué ventajas le ves al uso de closures en este código? ¿Y qué inconvenientes?

g) Reescribe el código (hazlo como mejor creas cambiando todo aquello que consideres necesario) de forma que obtengamos el mismo resultado pero sin hacer uso de closures.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01170E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JERARQUÍA DE OBJETOS JAVASCRIPT. FORMS, ELEMENTS, IMAGES, LINKS. NAVIGATOR: USERAGENT, GEOLOCATION, ONLINE. (CU01170E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº70 del Tutorial básico "JavaScript desde cero".

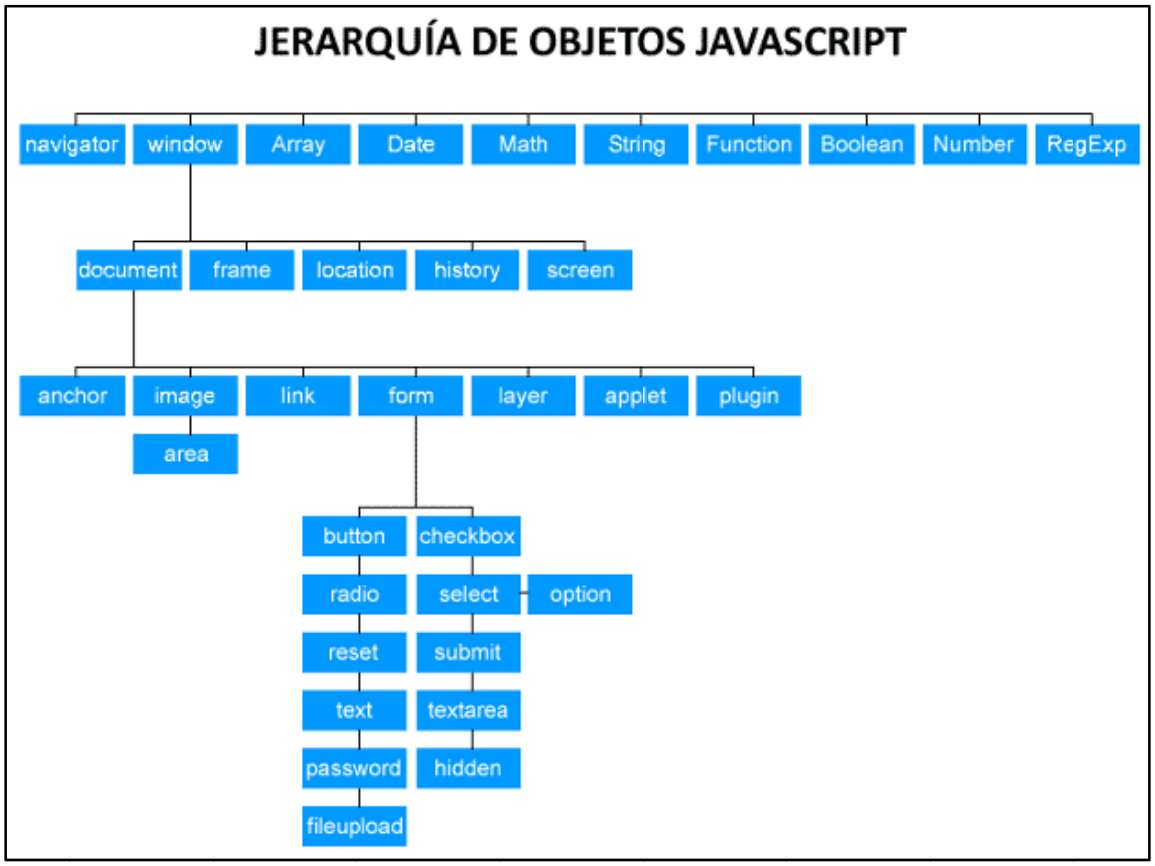
Autor: César Krall

JERARQUÍA DE OBJETOS JAVASCRIPT

Recordemos que window, el objeto global en JavaScript, tiene sus propiedades y métodos. La función que venimos usando para mostrar mensajes por pantalla, alert, es un método de window. Cuando escribimos alert('Hola'); en realidad estamos invocando window.alert('Hola');

Es decir, alert y window.alert son lo mismo para el navegador. El hecho de que no sea necesario escribir window cuando escribimos alert obedece a que si se invoca una función que no ha sido definida de otra manera, se considera que es un método del objeto global, es decir, un método de window.

JavaScript define una jerarquía de objetos que podemos reflejar de forma aproximada en un esquema similar al siguiente:



En estas jerarquía podemos pensar en objetos predefinidos de JavaScript (como Math, Date, String, RegExp, etc., incluido el objeto window).

También podemos pensar en el objeto document que contiene toda la información relativa a la estructura del documento HTML, lo que hemos llamado el DOM. No obstante, debemos diferenciar

entre la jerarquía del DOM, teniendo en cuenta que el DOM existe de forma independiente a JavaScript, y la jerarquía de objetos JavaScript, aunque guarden cierta similitud organizativa.

Podemos ver una página web como una colección de objetos. Por ejemplo, para JavaScript un formulario es un objeto, una imagen es un objeto, etc.

Los objetos tienen propiedades, métodos y eventos asociados.

Los objetos se organizan conforme a una jerarquía de forma que heredan métodos o propiedades de sus objetos padre, e incluso el nombre de un objeto se crea a partir de sus objetos padre.

Todo documento HTML dispone de los siguientes objetos en la jerarquía de objetos JavaScript:

navigator: tiene propiedades relacionadas con el nombre y la versión del navegador, protocolos de transferencia permitidos por el navegador (mime types) y sobre plugins instalados.

window: considerado habitualmente el objeto global o de máximo nivel. Tiene propiedades relacionadas con la ventana del navegador. En caso de uso de frames ("subventanas") hay un objeto window por cada "ventana hija" que exista.

document: tiene propiedades relacionadas con el documento como título, links, formularios, etc.

location: tiene propiedades relacionadas con la URL actual.

history: tiene propiedades relacionadas con URLs previamente visitadas.

Además de estos objetos en el documento HTML existirán más objetos según su contenido: objetos imágenes, objetos link, objetos formulario, etc.

CÓMO SE NOMBRAN LOS OBJETOS EN LA JERARQUÍA JAVASCRIPT

Para nombrar los objetos en la jerarquía de objetos JavaScript debemos tener en cuenta las siguientes reglas:

1) El nombre de un objeto que desciende de otro en la jerarquía JavaScript incluye el nombre de los objetos padre de la jerarquía. Por ejemplo el objeto document podemos nombrarlo como window.document. Un objeto hijo es a su vez un objeto y una propiedad del objeto padre. document es a su vez un objeto y una propiedad de window.

2) Dado que todos los objetos que podemos usar en el código descienden de window, normalmente omitiremos el uso de window a la hora de nombrar un objeto. Por eso escribiremos por ejemplo document.body en lugar de window.document.body, aunque ambas formas son válidas.

3) JavaScript organiza de forma automática ciertos objetos de naturaleza "múltiple" en arrays. Por ejemplo, una página web puede contener varios formularios. JavaScript, de forma automática, crea un array de objetos cuyo nombre es forms, siendo el primer formulario el de índice 0 y sucesivamente el 1, 2, etc. representan los siguientes formularios que aparezcan en el documento HTML por orden de aparición. Nos podemos referir a un formulario como window.document.forms[0], o más frecuentemente: document.forms[0]

Entre los arrays de objetos que crea JavaScript automáticamente tenemos:

forms: array con todos los formularios existentes en el documento HTML.

elements: array para cada formulario con los objetos que conforman dicho formulario (esto comprende objetos text, button, checkbox, hidden, radio, textarea, etc.). Si el primer formulario de una web tiene 3 input de tipo texto nos podemos referir al último de ellos como forms[0].elements[2]

images: array con todas las imágenes existentes en el documento HTML.

links: array con todos los links (tag HTML a) existentes en el documento HTML.

Para comprobar cómo podemos acceder a los objetos en la jerarquía de JavaScript escribe este código y comprueba los resultados (hemos incluido la ruta de dos imágenes, cámbiala si es necesario):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
window.document.images[0].style.border = 'solid blue 10px';
document.images[1].style.border = 'solid red 10px';
document.links[0].style.color = 'grey';
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="pulsador" onclick="ejemplo()"><a href="#"> Probar</a> </div>
<img src="http://i.imgur.com/afC0L.jpg" alt="Notepad++" title="Notepad++, un útil editor de texto" />
<img src="http://www.crimsoneditor.com/images/logo.jpg" alt="Crimson" title="Crimson, un editor de texto alternativo"
/>
</body>
</html>
```

El resultado esperado es que al pulsar sobre el texto “Probar”, la primera imagen adquiere un borde azul de 10 píxeles de grosor, la segunda imagen un borde rojo del mismo grosor y el texto del link queda de color gris. La sintaxis window.document.images[1] o simplemente document.images[1] nos da acceso a la segunda imagen existente en el documento HTML.

A través de la jerarquía de objetos de JavaScript podemos acceder a las propiedades de cada objeto por separado. En el ejemplo anterior hemos visto cómo cambiar propiedades de estilo CSS, pero también podríamos acceder y modificar otras propiedades. Por ejemplo, introduce estas líneas y comprueba el resultado:

```
var rut = 'http://aprenderaprogramar.com/images/stories/Libros/portada_libro_aprender_programar_java_lowres.jpg';
document.images[1].src = rut;
```

El resultado esperado es que al hacer click en el texto <<Probar>> una de las imágenes existentes deje de mostrarse y pase a mostrarse otra, debido a que hemos accedido al objeto que representa la imagen y hemos modificado su propiedad src (que define la ruta de la imagen).

El acceso a formularios y elementos de formularios suele ser de gran importancia con JavaScript, por lo que lo estudiaremos por separado.

EL OBJETO NAVIGATOR

Podemos recuperar un objeto de tipo Navigator haciendo la invocación objetoNav = window.navigator; o dado que window no es necesario, simplemente escribiendo navigator.

Ese objeto Navigator dispondrá de propiedades y métodos que nos pueden ser útiles, aunque de momento muchas de las propiedades y métodos son experimentales o no responden de la misma manera en los distintos navegadores. Vamos a citar aquí algunas de las propiedades:

PROPIEDAD	UTILIDAD	EJEMPLOS aprenderaprogramar.com
userAgent	Devuelve una cadena de texto representando el agente que se está empleando en la navegación. Puede identificar el navegador empleado, pero no es seguro porque puede configurarse para falsearlo.	alert(window.navigator.userAgent); // Por ejemplo Mozilla/5.0 (Windows NT 6.0; rv:31.0) Gecko/20100101 Firefox/31.0
battery	Devuelve un objeto BatteryManager que puede usarse para obtener información sobre el estado de la batería. No disponible para todos los navegadores.	alert('¿Cargando?: '+window.navigator.battery.charging); // ¿Cargando?: true si está cargando
geolocation	Devuelve un objeto Geolocation que puede usarse para obtener información sobre la ubicación (latitud, longitud) desde donde el usuario navega. Algunos navegadores no responden bien. Otros restringen o piden permiso al usuario por considerar que puede atentar contra su privacidad.	navigator.geolocation.getCurrentPosition (funcionSiExito, funcionSiFallo, arrayDeOpciones);

language	Devuelve un string representativo del lenguaje del navegador. Algunos navegadores no disponen de language pero en su lugar tienen disponible userLanguage. Otros navegadores no reconocen ni una ni otra forma.	var lenguaje = navigator.language || navigator.userLanguage;
		alert ('Código lenguaje navegador: '+lenguaje);
		//Código lenguaje navegador: es-ES por ejemplo

online	Devuelve un valor booleano (true o falso) que indica si se está o no con conexión a internet. Combinado con eventos (window.ononline y window.onoffline), puede servir para mostrar un mensaje de alerta si se pierde o recupera la conexión.	alert ('¿Estamos online?: '+window.navigator.onLine); // ¿Estamos online?: true si estamos conectados a internet

Entre los métodos de los objetos Navigator únicamente citaremos a Navigator.vibrate(), que genera la vibración de aquellos dispositivos (como smartphones) que admiten la vibración.

EJERCICIO

Usando la propiedad userAgent de los objetos Navigator, determina el navegador que está usando el usuario y muestra un mensaje por pantalla informando de ello. El resultado debe ser del tipo: <<Estás usando: nombreNavegador>>, donde nombreNavegador será Google Chrome, Apple Safari, Opera, Mozilla Firefox, Microsoft Internet Explorer ó Desconocido. Resuélvelo de dos maneras distintas:

a) Usando expresiones regulares.

b) Usando el método indexOf de los objetos tipo String de JavaScript.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01171E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

REDIRECCIONAR Y RECARGAR WEBS CON JAVASCRIPT. WINDOW.LOCATION. HREF, HOSTNAME, ASSIGN, RELOAD, REPLACE. (CU01171E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

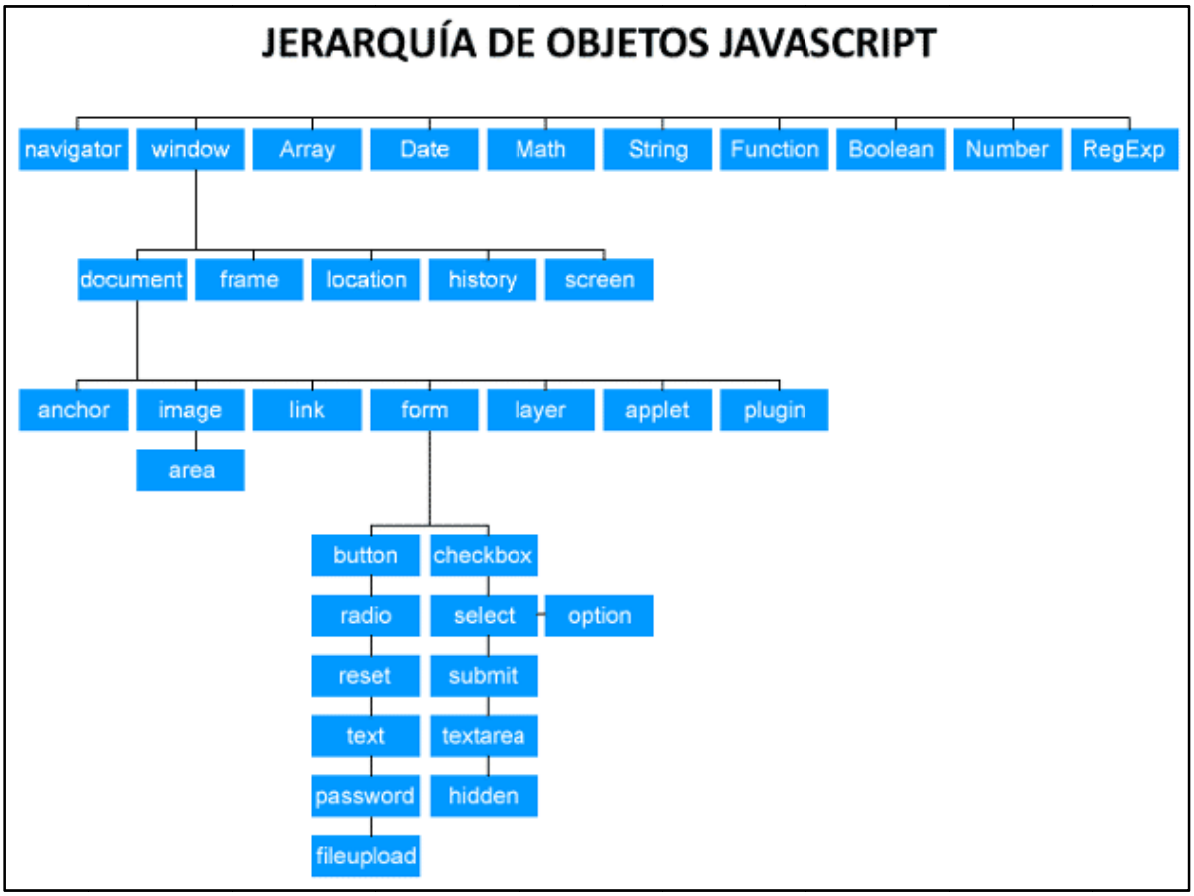
Resumen: Entrega nº71 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

WINDOW.LOCATION

Dentro de la jerarquía de objetos JavaScript existe un objeto denominado location, que es una propiedad de window. A su vez, se dice que existen objetos de tipo Location, siendo el objeto window.location un ejemplo de este tipo de objetos. Con este objeto podemos extraer información de la url (por ejemplo parámetros recibidos), redireccionar a otra página web, ó refrescar la web actual.

Recordemos que de forma aproximada puede representarse la jerarquía de objetos JavaScript de la siguiente manera.



PROPIEDAD WINDOW.LOCATION

La propiedad window.location devuelve un objeto de tipo Location con información relacionada con la URL del documento actual.

Vamos a citar aquí algunas de las **propiedades** de los objetos Location. Tener en cuenta que algunas de ellas no son soportadas por algunos navegadores:

PROPIEDAD	UTILIDAD	EJEMPLOS aprenderaprogramar.com
href	Devuelve un string con la url completa. window.location es equivalente a document.location pero recomendamos usar window.location.	alert ('url actual: '+window.location.href); alert ('url actual: '+location.href); // Devuelve la url actual completa
protocol	Devuelve el protocolo de la URL, normalmente http ó https. Si trabajamos en local será file:	alert ('protocolo actual: '+window.location.protocol); //Devuelve por ejemplo http
host	Devuelve el nombre del servidor, normalmente como nombre de dominio, y si existe un puerto especificado también, separado con :.	alert ('host actual: '+window.location.host); //Devuelve por ejemplo http://aprenderaprogramar.com
hostname	Devuelve el dominio o servidor.	alert ('el hostname actual: '+window.location.hostname); //Devuelve por ejemplo http://aprenderaprogramar.com
pathname	Devuelve la ruta excluido el nombre de dominio o servidor.	alert ('El pathname actual: '+window.location.pathname); //Por ejemplo devuelve /ejemplo1.html
search	Devuelve la cadena que representa los parámetros que existen en la url.	alert ('Los parámetros en url: '+window.location.search); // Por ejemplo: Los parámetros en url: ?name=pepe&apellido1=perez&pais=ecuador
hash	Devuelve # seguido de un texto si en la url existe un #seguido de un texto (ancla de localización del fragmento html).	alert ('Valor localizador #: '+ window.location.hash); // Por ejemplo: #final
origin	Devuelve una cadena que representa la raíz del sitio web	alert ('Valor origin: '+ window.location.origin); //Devuelve por ejemplo http://aprenderaprogramar.com

Vamos a citar aquí algunos de los **métodos** de los objetos Location. Tener en cuenta que algunas de ellas no son soportadas por algunos navegadores:

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com

assign (urlDestino)	Asigna una url a la ventana actual, lo que genera que se cargue la urlDestino. El usuario puede volver atrás con el botón back del navegador.	alert ('Transfiriendo a aprenderaprogramar: ');
		window.location.assign('http://aprenderaprogramar.com');
		//Da lugar a que se cargue la url indicada con posibilidad de volver atrás con el botón back

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com

reload (opcSiDesdeServidor)	Da lugar a la recarga de la url actual. Si se indica como parámetro true, la recarga se hará desde el servidor (refresco completo). Si no se especifica o es false la recarga puede ser desde caché.	alert ('Recargando la página... ');
		window.location.reload(true);
		//Se recarga la página desde el servidor (refresco completo)

replace (urlDestino)	Asigna una url a la ventana actual, lo que genera que se cargue la urlDestino. El usuario no puede volver atrás con el botón back del navegador.	alert ('Reemplazando la página... ');
		window.location.replace('http://aprenderaprogramar.com');
		//Da lugar a que se cargue la url indicada sin posibilidad de volver atrás con el botón back

toString()	Devuelve una cadena representativa de la url (window.location.toString() devuelve lo mismo que window.location.href)	alert ('Url actual: '+window.location.toString()); //Devuelve la url actual

EJEMPLO DE USO DE WINDOW.LOCATION

Supongamos una url que se construye según este patrón: http://aprenderaprogramar.com/index.php?option=compra&moneda=euro&producto=libro1, donde el parámetro option puede ser compra, venta o intercambio. El parámetro moneda puede ser pesoMexicano, euro, pesoArgentino, pesoColombiano, bolivar, sol, pesoChileno. Y el parámetro producto puede ser cualquier valor.

Y supongamos que queremos rescatar el valor que tenga el parámetro moneda usando JavaScript. Podemos hacerlo de esta manera. Escribe este código y guárdalo con nombre de archivo ejemplo1.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
        msg = '';
        var moneda = extraerParametroUrl('moneda');
        if (moneda) {alert ('El valor del parámetro moneda en la url es: '+moneda)}
        else {alert ('No se recibe moneda en la url'); }
}
function extraerParametroUrl( nombreParametro ){
        var regexS = "[\\?&]"+nombreParametro+"=([^&#]*)";
        var regex = new RegExp ( regexS );
        var tmpURL = window.location.href;
        var results = regex.exec( tmpURL );
        if( results == null ) { return "";} else {return results[1];}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="pulsador" onclick="ejemplo()"><a href="#"> Probar</a> </div>
</body></html>
```

A continuación visualiza el archivo en tu navegador. El resultado esperado es que al pulsar el enlace "Probar" se muestre: << No se recibe moneda en la url >>

Si añadimos el parámetro al final de la ruta, por ejemplo: .../ ejemplo1.html?moneda=pesoMexicano

El resultado esperado es que al pulsar el enlace "Probar" se muestre: <<El valor del parámetro moneda en la url es: pesoMexicano>>

Este código hace uso de expresiones regulares JavaScript. Si no recuerdas cómo funcionan éstas, lee los anteriores apartados del curso que hemos dedicado a expresiones regulares.

Hemos definido var regexS = "[\\?&]"+nombreParametro+"=([^&#]*)";

Esto representa una expresión regular JavaScript que comienza con ? ó &, viene seguido de nombreParametro, seguido de un igual, y de cualesquiera caracteres hasta alcanzar el final de la cadena ó un simbolo & ó un símbolo #, que serían las situaciones que nos indicarían que el valor del parámetro ya ha sido extraido completamente. Recordar que dentro de corchetes el símbolo ^ funciona como negación. Al ejecutar exec sobre una cadena (en este caso la URL) el índice [0] del resultado contiene el valor completo, por ejemplo <<?moneda= pesoMexicano >>, mientras que el siguiente índice contiene el match correspondiente al primer paréntesis, en este caso sería << pesoMexicano >>. El índice [2] contendría el match correspondiente al segundo paréntesis, pero aquí sólo tenemos un paréntesis con lo cual el índice [2] no tiene valor definido.

Como conclusión de este ejemplo, vemos cómo podemos rescatar parámetros de una url enviados por el método GET haciendo uso de JavaScript de varias maneras: hemos visto métodos específicos y esta otra manera mostrada en este ejemplo, que al ser más generalista asegura un buen funcionamiento en todos los navegadores.

EJERCICIO

Crea una página web que al ser invocada muestre un mensaje "Hemos cambiado de ubicación esta página. En breves momentos será redireccionado..." junto a una cuenta atrás que muestre 5, 4, 3, 2, 1 (correspondiente a 5 segundos). Tras transcurrir 5 segundos y mostrarse la cuenta atrás, el usuario debe ser redireccionada a la url http://aprenderaprogramar.com. Para realizar este ejercicio debes usar la propiedad window.location junto con funciones que permitan el retardo en la ejecución vistas en entregas anteriores del curso.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01172E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

RETROCEDER PÁGINA ANTERIOR JAVASCRIPT. WINDOW.HISTORY. OBTENER TAMAÑO PANTALLA. WINDOW.SCREEN. FRAMES. (CU01172E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

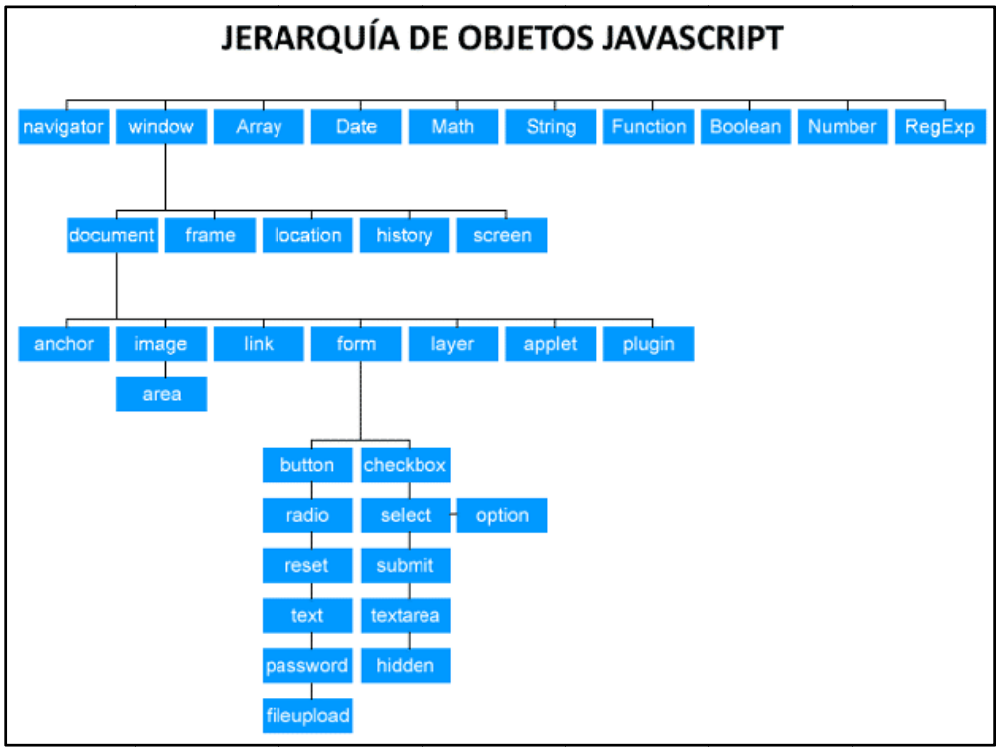
Resumen: Entrega nº72 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

WINDOW.HISTORY

Dentro de la jerarquía de objetos JavaScript existe un objeto denominado history, que es una propiedad de window. A su vez, se dice que existen objetos de tipo History, siendo el objeto window.history un ejemplo de este tipo de objetos. Con este objeto podemos trabajar con el historial de navegación del usuario y retroceder a una página anterior o avanzar páginas visitadas.

Recordemos que de forma aproximada puede representarse la jerarquía de objetos JavaScript de la siguiente manera.



PROPIEDAD - OBJETO WINDOW.HISTORY

La propiedad window.history devuelve un objeto de tipo History con información relacionada con las URLs visitadas por el usuario en la pestaña actual.

Por motivos de seguridad, window.history no almacena información sobre las url´s concretas accedidas por el usuario, aunque permite moverse hacia delante y hacia detrás en las urls visitadas usando métodos específicos para ello.

Vamos a citar aquí algunas de las **propiedades** de los objetos History. Tener en cuenta que algunas de ellas no son soportadas por algunos navegadores:

PROPIEDAD	UTILIDAD	EJEMPLOS aprenderaprogramar.com
length	Devuelve un entero que representa el número de elementos (urls) visitados para esa pestaña y sesión por el usuario. Si sólo se ha visitado la url actual devuelve 1.	alert ('Valor window.history.length es: '+ window.history.length); //Devuelve un entero, por ejemplo 4 si se han visitado 4 direcciones web (3 más la actual).
state	Un objeto asociado a la url actual, que almacena propiedades informativas de situaciones que existían durante la navegación y que deben establecerse a través de código de programación. Su fin es poder "recordar" situaciones que existían durante la navegación.	alert ('Valor window.history.state es: '+ window.history.state); // Por defecto null

La propiedad state nos permitiría por ejemplo almacenar un valor que hubiera elegido un usuario de un formulario y que no se encuentra en la url pero podemos considerar como asociado a la historia de navegación.

Vamos a citar aquí algunos de los **métodos** de los objetos History. Tener en cuenta que algunas de ellas no son soportadas por algunos navegadores:

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
back()	Carga la anterior url visitada en esa pestaña de navegación por el usuario. Equivale a que el usuario pulsara el botón back del navegador.	window.history.back(); //Carga la anterior url en la historia de navegación
forward()	Carga la siguiente url almacenada en el historial de navegación del usuario para esa pestaña.	window.history.forward(); //Carga la siguiente url en la historia de navegación
go (numeroUrls)	Permite avanzar numeroUrls hacia delante o hacia detrás (con números negativos). Así history.go(-1) equivale a history.back()	window.history.go(-2); //Salta atrás dos urls en la historia de navegación
Otros métodos	Otros métodos como history.pushState y history.replaceState() permiten añadir o modificar entradas de la historia de navegación del usuario.	No vamos a presentar ejemplos de estos métodos

Recordar que en el contexto global window.history.back() es equivalente a history.back(). No obstante, si queremos dejar patente nuestra intención de referirnos a window podemos escribirlo si así lo preferimos.

Escribe este código y comprueba su funcionamiento:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
 window.onload = function () {
var adelante = document.getElementById('irAdelante');
adelante.addEventListener("click", irAdelante);
var atras = document.getElementById('irAtras');
atras.addEventListener("click", irAtras);
function irAdelante() { this.style.color = 'orange'; window.history.forward();}
function irAtras() { this.style.color = 'red'; window.history.back();}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="irAdelante">Ir adelante</div>
<div style="color:blue; margin:20px;" id ="irAtras">Ir atras </div>
<a href="http://aprenderaprogramar.com">Pulsa para generar un avance y poder volver atrás con back</a>
</body>
</html>
```

Para comprobar el funcionamiento, accede a una página web, por ejemplo clarin.com, luego carga la url correspondiente al archivo html y pulsa “Ir atrás”. Al pulsar, el navegador volverá a clarin.com. Si pulsas en el enlace “Pulsa para generar un avance…”, luego pulsas el botón back del navegador y luego pulsas sobre “Ir adelante”, el navegador cargará la web aprenderaprogramar.com porque es la que se encuentra delante en el historial de navegación.

EVENTO ONPOPSTATE

Existe un evento denominado onpopstate que se dispara cuando se produce un cambio de url (navegación, avance o retroceso) dentro del proceso de navegación del usuario.

PROPIEDAD - OBJETO WINDOW.SCREEN

La propiedad window.screen devuelve un objeto de tipo Screen con información relacionada con la pantalla donde el usuario está visualizando el documento HTML.

Vamos a citar aquí algunas de las **propiedades** de los objetos Screen. Tener en cuenta que algunas de ellas no son soportadas por algunos navegadores. Recordar que para ver los mensajes con la instrucción console.log hemos de tener abierta la consola del navegador.

PROPIEDAD	UTILIDAD	EJEMPLOS aprenderaprogramar.com
width	Devuelve el ancho de pantalla, en píxeles	console.log('Ancho de pantalla en px: '+window.screen.width); // Por ejemplo 1280 px
height	Devuelve el alto de pantalla, en píxeles	console.log('Alto de pantalla en px: '+window.screen.height); // Por ejemplo 800 px
availWidth	Devuelve el ancho de pantalla disponible para visualización, descontados elementos fijos como barras que resten espacio de visualización.	console.log('Ancho de pantalla efectivo en px: '+window.screen.availWidth); // Por ejemplo 1280 px
availHeight	Devuelve el ancho de pantalla disponible para visualización, descontados elementos fijos como barras que resten espacio de visualización.	console.log('Alto de pantalla efectivo en px: '+window.screen.availHeight); // Por ejemplo 770 px debido a la barra de tareas
Otras	colorDepth (calidad de color en bits), pixelDepth (cantidad de colores que puede representar un píxel), y otras propiedades experiementales como orientation.	No veremos ejemplos sobre otras propiedades

Los objetos Screen tienen algunos métodos pero su uso está vinculado a prefijos específicos de navegador o son tecnologías experimentales más propias de smartphones y de tablets que de desarrollos web habituales, por lo que no vamos a comentarlos aquí.

PROPIEDAD - OBJETO WINDOW.FRAMES

Si un documento HTML contiene frames ó iframes, el navegador crea un objeto window asociado al documento HTML principal, y tantos objetos window adicionales como frames o iframes existan.

La propiedad window.frames devuelve un objeto de tipo array-like con los objetos tipo window (frames ó iframes), que son subventanas de la ventana principal.

Para entender qué devuelve window.frames ejecuta este código donde se accede de forma tradicional:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
window.onload = function() {
iframe = document.getElementsByTagName( "iframe" );
  alert( "Tenemos: " + iframe[0].name + ', ' + iframe[1].name );
  }
function ejemplo() {
for (var i = 0; i < iframe.length; i++) {iframe[i].style.background = "red";}
}
</script>
```

```
</head>
<body>
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="pulsador" onclick="ejemplo()"> Probar </div>
<iframe name="aprende1" style="margin:20px; padding:20px;" src="http://aprendeaprogramar.es" width="350"
height="600">
  <p>Tu navegador no admite iframes.</p>
</iframe>
<iframe name="aprende2" style="margin:20px; padding:20px;" src="http://aprenderaprogramar.com/foros" width="350"
height="600">
  <p>Tu navegador no admite iframes.</p>
</iframe>
</body>
</html>
```

En teoría window.frames nos devolvería lo mismo que document.getElementsByTagName("iframe");, pero en la práctica el resultado de usar esta propiedad es irregular y está influido por parámetros (server side headers) que envía el servidor que sirve la página web dentro del frame o iframe que pueden dar lugar a restricciones.

Si pruebas a hacer el cambio en el código anterior: iframe = window.frames;

Al intentar ejecutarlo obtendrás un mensaje de error por consola (por ejemplo Error: Permission denied to access property 'name', Error: Permission denied to access property 'style'). Sin embargo si en lugar de intentar modificar el estilo intentas modificar la propiedad location, por ejemplo iframe[i].location = "http://aprenderaprogramar.com"; no obtendrás este error.

EJERCICIO

Crea una página web que muestre en el lado izquierdo tres opciones donde podrá pulsar el usuario: clarin.com, elmercurio.cl y elespectador.com. En la parte inferior izquierda, debajo de estas opciones, habrá un frame donde deberá cargarse la página web según elija el usuario (por ejemplo si el usuario pulsa en elmercurio.cl en el frame que se encuentra debajo deberá mostrarse la web de elmercurio.cl). En el lado derecho estarán las opciones eluniversal.com.mx, elcomercio.pe y elmundo.es. En la parte inferior derecha, debajo de estas opciones, habrá un frame donde deberá cargarse la página web según elija el usuario.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01173E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

OBJETO WINDOW JAVASCRIPT. PROPIEDADES: NAME, INNERWIDTH, LENGTH, , OPENER, OUTERHEIGHT, PAGEXOFFSET, SCREENX (CU01173E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

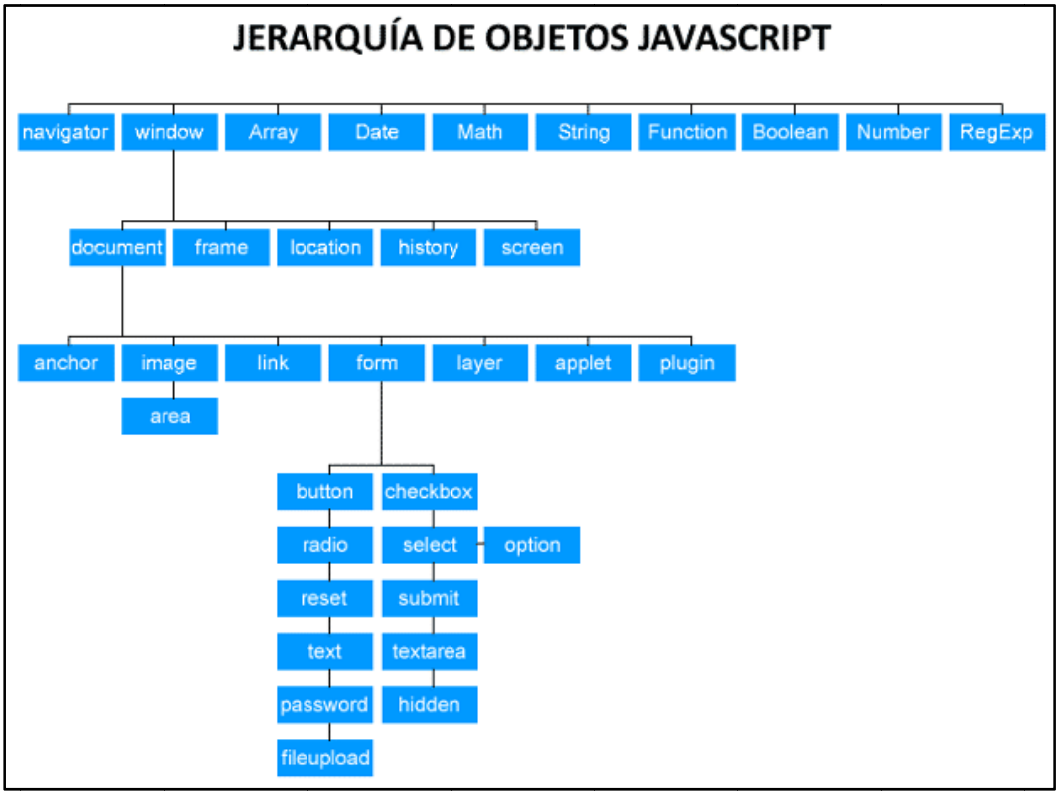
Resumen: Entrega nº73 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

WINDOW JAVASCRIPT

Hemos visto distintas propiedades del objeto window, que a su vez se constituyen en objetos que disponen de sus propias propiedades y métodos. Ya conocemos muchas de las propiedades y métodos del objeto window, pero vamos ahora a repasar algunas y estudiar otras nuevas.

Recordemos que de forma aproximada puede representarse la jerarquía de objetos JavaScript de la siguiente manera.



Un objeto window representa una ventana cuya organización interna es conforme al DOM (document object model). En un navegador donde tengamos abiertas varias pestañas, cada pestaña equivale a un objeto window que tiene su propia propiedad document y con el que podemos trabajar por separado. Desde esta perspectiva, cada pestaña es un objeto window. No obstante, hay algunos métodos que sólo se pueden aplicar sobre ventanas abiertas mediante JavaScript y no a la ventana principal de navegación del usuario.

No tiene interés conocer de memoria cuáles son todas las propiedades de los objetos window, pero sí al menos tener una idea o referencia que nos permita buscar información cuando nos resulte necesario.

Vamos a citar aquí de forma breve **propiedades** de los objetos window. Tener en cuenta que usando JavaScript podemos crear ventanas auxiliares para mostrar al usuario distinto contenido, y que dichas ventanas auxiliares también son objetos window. Cuando hablamos de dimensiones, posicionamiento y coordenadas, pueden obtenerse resultados extraños si se trabaja con más de un monitor.

PROPIEDAD	UTILIDAD	EJEMPLOS aprenderaprogramar.com
closed	Devuelve un valor booleano que indica si un objeto window creado previamente está cerrado o no	if (myWindow.closed) {alert('La ventana fue cerrada');} //Se muestra el mensaje si la ventana fue cerrada
document	Devuelve el objeto document asociado a la ventana.	Ya vistos durante el curso
frames	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
history	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
innerHeight	Devuelve la altura en píxeles del área de contenidos (viewport) del navegador incluyendo scrollbar si existe.	alert ('Altura en px del viewport es: '+window.innerHeight); // Por ejemplo 820
innerWidth	Devuelve la anchura en píxeles del área de contenidos (viewport) del navegador incluyendo scrollbar si existe.	alert ('Anchura en px del viewport es: '+window.innerWidth); // Por ejemplo 1130
length	Devuelve el número de elementos frame ó iframe presentes en la ventana.	alert ('Elementos frame o iframe: '+window.length); // Por ejemplo 0
location	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
name	Devuelve o establece el nombre de una ventana. Una ventana no tiene que tener nombre obligatoriamente aunque puede ser útil para diferenciar entre ventanas si hay varias.	window.name='Aprender a programar'; alert ('Valor name de ventana es: '+window.name); // Valor name de ventana es Aprender a programar
navigator	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
opener	Si desde una ventana se ha abierto otra, opener devuelve una referencia a la ventana "generadora" o ventana desde la que se invocó la apertura de la ventana actual. Si es la ventana inicial de navegación, opener devuelve null.	if (window.opener){alert ('Nombre de la ventana que abrió esta: '+window.opener.name); } //Devuelve el nombre si existe una ventana padre
outerHeight	Devuelve la altura en píxeles de la ventana completa hasta el límite exterior de sus bordes (quedando dentro de esta medida barras de menú, barras de estado, etc.).	alert ('Dimension vertical total: '+window.outerHeight); // Por ejemplo 1040
outerWidth	Devuelve el ancho en píxeles de la ventana completa hasta el límite exterior de sus bordes (quedando dentro de esta medida barras de menú, barras de estado, etc.).	alert ('Dimension horizontal total: '+window.outerWidth); // Por ejemplo 1296

pageXOffset	Valor en píxeles indicativo de cuánto scroll se ha realizado en sentido horizontal.	alert ('Scroll horizontal realizado: '+window.pageXOffset); // 0 si no hay barra de scroll horizontal o no se ha hecho scroll
pageYOffset	Valor en píxeles indicativo de cuánto scroll se ha realizado en sentido vertical.	alert ('Scroll vertical realizado: '+window.pageYOffset); // 0 si no hay barra de scroll vertical o no se ha hecho scroll
scrollX	Igual que pageXOffset. Se recomienda usar pageXOffset.	Igual que pageXOffset. Se recomienda usar pageXOffset.
scrollY	Igual que pageYOffset. Se recomienda usar pageYOffset.	Igual que pageYOffset. Se recomienda usar pageYOffset.
parent	Referencia a la ventana que aloja una subventana (es decir, ventana que aloja un frame o iframe típicamente). Si no existe parent, devuelve una referencia a sí misma (la propia ventana).	window.name = 'Aprender programación'; alert ('Mi propio nombre es: '+window.parent.name); // Mi propio nombre es: Aprender programación
screen	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
screenLeft	Igual que screenX. Se recomienda usar screenX.	Igual que screenX. Se recomienda usar screenX.
screenTop	Igual que screenY. Se recomienda usar screenY.	Igual que screenY. Se recomienda usar screenY.
screenX	Devuelve la coordenada X relativa a la pantalla (screen). Origen lateral izquierdo de la pantalla. Valor en pixels.	alert ('Ventana a: '+window.screenX + ' pixels del borde izquierdo de pantalla'); // Por ejemplo Ventana a: 0 pixels del borde izquierdo de pantalla Nota: dependiendo del monitor y del navegador se pueden obtener valores negativos (por ejemplo: -1024x0 a 0x768).
screenY	Devuelve la coordenada Y relativa a la pantalla (screen). Origen borde superior de la pantalla. Valor en pixels.	alert ('Ventana a: '+window.screenY + ' pixels del borde superior de pantalla'); // Por ejemplo Ventana a: 0 pixels del borde superior de pantalla Nota: dependiendo del monitor y del navegador se pueden obtener valores negativos como origen.
self	Devuelve una referencia a la propia ventana	if (window.self.name){ alert ('Mi nombre es: ' + window.self.name);} else {alert('No tengo nombre');}
top	Devuelve una referencia a la ventana principal (ventana más alta en la jerarquía de ventanas)	if (window.top){ alert ('Soy la ventana principal');} // Soy la ventana principal si es la principal
Otras	Existen otras propiedades, algunas no estandarizadas.	No veremos ejemplos

EJERCICIO

El siguiente código crea una ventana cuando se pulsa en el texto “Pulsa aquí”. Esta ventana es asignada a una variable u objeto denominado nuevaVentana. Queremos conocer las siguientes propiedades del objeto nuevaVentana y mostrarlas por pantalla: si está cerrada o no, dimensiones del viewport, número de frames presentes en ella, url a la que apunta, valor de name, dimensiones de la ventana completa hasta los bordes exteriores y cuántos píxeles está desplazada en horizontal y vertical respecto al punto origen de la pantalla

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
 window.onload = function () {
		var ejemplo = document.getElementById('ejemplo');
		ejemplo.addEventListener("click", ejecutarEjemplo);
}
function ejecutarEjemplo () {
		var nuevaVentana = window.open ('http://aprendeaprogramar.es', 'miNombre', 'width=300, height=300,
resizable=true, menubar=yes');
		nuevaVentana.focus();
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
</body>
</html>
```

Añade el código JavaScript necesario para mostrar esta información por pantalla.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01174E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

MÉTODOS DE WINDOW JAVASCRIPT. EFECTOS. CONFIRM, PROMPT, OPEN, MOVEBY, MOVETO, FOCUS, RESIZETO, CLOSE (CU01174E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº74 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MÉTODOS DE WINDOW

Hemos visto distintas propiedades del objeto window, que a su vez se constituyen en objetos que disponen de sus propias propiedades y métodos. Ya conocemos muchas de las propiedades y métodos del objeto window, pero vamos ahora a repasar algunas y estudiar otras nuevas.

Un objeto window representa una ventana cuya organización interna es conforme al DOM (document object model). En un navegador donde tengamos abiertas varias pestañas, cada pestaña equivale a un objeto window que tiene su propia propiedad document y con el que podemos trabajar por separado. Desde esta perspectiva, cada pestaña es un objeto window. No obstante, hay algunos métodos que sólo se pueden aplicar sobre ventanas abiertas mediante JavaScript y no a la ventana principal de navegación del usuario.

No tiene interés conocer de memoria cuáles son todos los métodos de los objetos window, pero sí al menos tener una idea o referencia que nos permita buscar información cuando nos resulte necesario.

Vamos a citar aquí algunos de los **métodos** de los objetos window. Cuando hablamos de dimensiones, posicionamiento y coordenadas, pueden obtenerse resultados extraños si se trabaja con más de una pantalla (por ejemplo un portátil y una pantalla auxiliar).

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
alert (mensaje)	Muestra una ventana de diálogo con la cadena especificada o con la conversión a string del objeto pasado como parámetro y un botón “Aceptar”	alert('hola usuario');
confirm (mensaje)	Abre un cuadro de diálogo con un mensaje y dos botones, Aceptar y Cancelar. Devuelve un valor booleano: true si se pulsa aceptar o false si se pulsa cancelar.	if (window.confirm("Va a salir de la aplicación ¿Desea salir realmente?")) { window.alert('Gracias por visitarnos'); } // Si usuario pulsa aceptar se cumple la condición
clearInterval (referenciaTimer)	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
clearTimeout (referenciaTimer)	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
open (opURL, opName)	Uso habitual var nuevaVentana = window.open (...) donde nuevaVentana es una referencia al objeto window que se crea. opURL es un parámetro opcional, url a cargar en una nueva pestaña. opName nombre opcional que puede usarse para fijar atributo target en un link.	var nuevaVentana = window.open ('http://aprendeaprogramar.es', 'miNombre'); //Si no se especifica url de destino, se abre una pestaña en blanco

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
open (opURL, opName, opParametros)	Uso habitual var nuevaVentana = window.open (...) donde nuevaVentana es una referencia al objeto window que se crea. opURL es un parámetro opcional, url a cargar en la nueva ventana. opName: nombre, opcional, puede usarse para fijar atributo target en un link. opParametros es una lista de parámetros como left ó top para posición, height o width para dimensiones, outerHeight, outerWidth, innerHeight, innerWidth, menubar, toolbar, location, resizable, scrollbars, status, titlebar...	var nuevaVentana = window.open ('http://aprendeaprogramar.es', 'miNombre', 'width=300, height=300, resizable=true, menubar=yes'); // Puede haber diferencias entre navegadores en la respuesta a algunos de los parámetros
close()	Cierra una ventana. Sólo se pueden cerrar ventanas que hayan sido abiertas por un script. No es posible cerrar la ventana principal del navegador por no abrirse por un script.	nuevaVentana.close(); //Cierra la ventana

moveBy (valorX, valorY)	Desplaza el objeto window una distancia valorX en horizontal y valorY en vertical, en píxeles, respecto de su posición inicial (movimiento relativo). No se pueden desplazar ventanas no creadas con scripts.	nuevaVentana.moveBy(200, 200);
		nuevaVentana.focus();
		// Si no usamos focus() la ventana puede perder el foco y no ser visible.
		Si la ventana contiene una url que no pertenece al mismo directorio-dominio que la ventana madre podemos obtener un error de tipo Error: Permission denied to access property 'moveBy'

focus()	Pone el foco en una ventana que lo ha perdido	nuevaVentana.focus();

moveTo(coordX, coordY)	Coloca el objeto window en una posición valorX en horizontal y valorY en vertical, en píxeles, respecto de la esquina superior izquierda de la pantalla. No se pueden desplazar ventanas no creadas con scripts.	nuevaVentana.moveTo(0, 0);
		nuevaVentana.focus();
		// Si no usamos focus() la ventana puede perder el foco y no ser visible.

print()	Abre el cuadro de diálogo para imprimir por impresora el contenido de la ventana	window.print();

prompt(msg, opValorDefecto)	Muestra un cuadro de diálogo pidiendo una entrada al usuario y, opcionalmente, un valor de defecto. Devuelve un objeto String con el contenido introducido por el usuario, o null si se pulsa la tecla cancelar. Si se pulsa aceptar sin introducir nada retorna una cadena vacía.	var persona = prompt("¿Cuál es tu nombre?", "Barack Obama");
		if (persona != null) {alert("Hola " + persona);}
		//Por ejemplo <<Hola Pedro>>

resizeBy (varX, varY)	Modifica el tamaño de la ventana agrandándola o empequeñeciéndola en la cantidad de pixels indicada por varX para la dimensión horizontal o varY para la dimensión vertical. No se pueden redimensionar ventanas no creadas con scripts.	nuevaVentana.resizeBy(250, 250);
		nuevaVentana.focus();
		// Si no usamos focus() la ventana puede perder el foco y no ser visible.

resizeTo(ancho, alto)	Modifica el tamaño de la ventana y lo establece a los valores de ancho y alto especificados. No se pueden redimensionar ventanas no creadas con scripts.	nuevaVentana.resizeTo(100, 100);
		nuevaVentana.focus();
		// Si no usamos focus() la ventana puede perder el foco y no ser visible.

MÉTODO	UTILIDAD	EJEMPLOS aprenderaprogramar.com
scrollBy(varX, varY)	Hace scroll en la ventana en la cantidad de pixels indicada por varX para la dimensión horizontal o varY para la dimensión vertical.	window.scrollBy(0, 400); //Desplaza el scroll 400 píxeles hacia abajo
scrollTo(posX, posY)	Hace scroll hasta dejar como la esquina superior izquierda a posX pixeles del origen en horizontal y posY píxeles en vertical.	window.scrollTo(0, 400); //Coloca el scroll 400 píxeles por debajo del origen
scrollByLines(numLin)	Hace scroll hacia arriba o abajo en el número de líneas especificado. Una línea se corresponde con un párrafo <p>.	window.scrollByLines(20);
setInterval(...)	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
setTimeout	Ya estudiado. Ver entregas anteriores.	Ya estudiado. Ver entregas anteriores.
stop()	Detiene la carga de los contenidos de la ventana de la misma manera que si el usuario pulsara el botón para detener la carga en el navegador.	window.stop();
Otros	blur() quita el foco, btoa('cadena') codifica en base-64 una cadena y atob(referencia) decodifica una referencia codificada en base-64, fullScreen() indica si la ventana está a pantalla completa o no, find('cadena') para buscar una cadena en la ventana, minimize() para minimizar la ventana	Algunos de estos métodos no son soportados por todos los navegadores. No vamos a ver ejemplos de ellos.

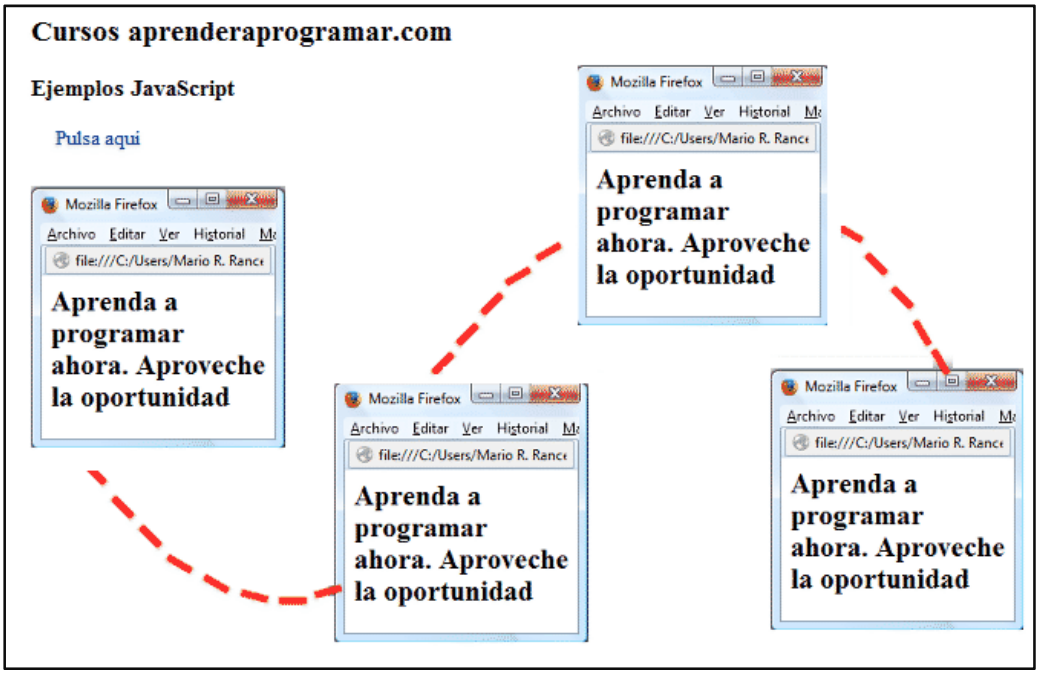
EJEMPLO Y EJERCICIO

El siguiente código nos sirve al mismo tiempo de ejemplo y de ejercicio. Ejecuta el código y comprueba sus resultados. Luego responde las preguntas y cuestiones que se plantean más abajo.

Nota: si estás trabajando con una sola pantalla, el resultado esperado es que aparezca una nueva ventana que se va moviendo de izquierda a derecha y de arriba abajo con un movimiento sinusoidal (como una onda que sube y baja). Si estás trabajando con varias pantallas, es posible que el único movimiento que veas sea en horizontal. ¿Por qué? Porque habrá un monitor principal y un monitor secundario. Con moveTo el navegador puede tratar de realizar el posicionamiento en un monitor que no es el que tú estés usando y en consecuencia no visualizarse lo esperado.

Nota: para comprender los detalles de este ejercicio hace falta tener conocimientos básicos de trigonometría (seno de un ángulo, cálculos usando radianes). Si no tienes estos conocimientos no te preocupes: sigue avanzando.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var start = 0;
var positionY = window.innerHeight/2-125;
var nuevaVentana;
var controlSinusoidal=0;
 window.onload = function () {
            var ejemplo = document.getElementById('ejemplo');
            ejemplo.addEventListener("click", ejecutarEjemplo);
}

function ejecutarEjemplo () {
            nuevaVentana = window.open ('', 'miNombre', 'width=250, height=250, resizable=true, menubar=yes');
            nuevaVentana.document.write('<h1>Aprenda a programar ahora. Aproveche la oportunidad</h1>');
            nuevaVentana.focus();
            nuevaVentana.moveTo(0,positionY);
            nuevaVentana.setInterval( sine, 1000/30 );
}

function sine(){
  controlSinusoidal = controlSinusoidal + Math.PI/256;
  positionY = 100*Math.sin(controlSinusoidal);
  start += 2;
  nuevaVentana.moveTo(start, window.innerHeight/2-100+positionY);
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
</body>
</html>
```



Preguntas y cuestiones a desarrollar:

a) ¿Qué representa la variable start? ¿Qué significado tiene que su valor inicial sea cero?

b) ¿Qué representa la variable positionY? ¿Qué significado tiene que su valor inicial sea window.innerHeight/2-125 ?

c) ¿Con qué frecuencia (cada cuánto tiempo) se produce el refresco o redibujado de la pantalla?

d) ¿Cuántos píxeles se desplaza hacia la derecha la ventana en cada refresco de pantalla que realiza el navegador?

e) La base matemática del movimiento que describe la ventana es el movimiento ondulatorio. ¿Por qué la ventana sube y baja?

f) Modifica el código para que cuando la ventana llegue al lateral derecho de la pantalla se detenga su movimiento y aparezca un mensaje (alert) con el texto ¡Se terminó!

g) Modifica el código para que cuando la ventana llegue al lateral derecho de la pantalla haga un efecto de "rebote" y empiece a moverse en sentido opuesto al que venía (es decir, subiendo y bajando pero ahora de derecha a izquierda), hasta llegar al lateral izquierdo y volver a rebotar, y así indefinidamente....

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01175E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FORMULARIOS JAVASCRIPT: EJEMPLOS DE DOCUMENT.FORMS Y ELEMENTS. ACCESO DIRECTO A FORMULARIOS Y CAMPOS CON ID Y NAME (CU01175E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº75 del Tutorial básico "JavaScript desde cero".

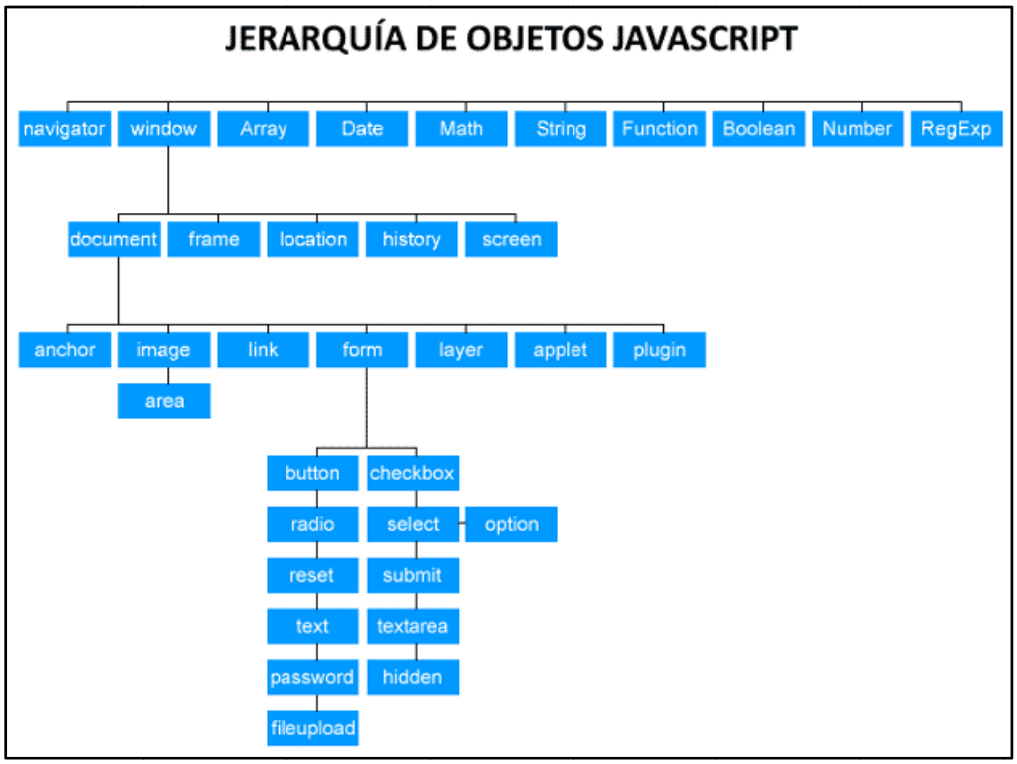
Autor: César Krall

TRABAJO CON FORMULARIOS JAVASCRIPT

En la jerarquía de objetos JavaScript que hemos visto podríamos distinguir entre tres tipos de objetos: objetos del navegador (window, document, etc.), objetos del lenguaje (Math, RegExp, etc.) y por su importancia tradicional en la programación JavaScript, los objetos campos de formularios. Vamos a hablar sobre el trabajo con formularios en JavaScript.

Cuando comenzó el desarrollo de internet y de la programación web existía el problema de que el trabajo con formularios era muy pesado. Supón que un usuario deja un campo de un formulario que es obligatorio vacío: si la información tiene que ser enviada al servidor, en el servidor procesarse la información, comprobar que hay un campo vacío, y enviar una respuesta al usuario informándole de que hay un campo vacío, se consume tiempo, puede haber interrupciones, el usuario se desespera... Por ello JavaScript se convirtió en una potente herramienta para solucionar este problema: realizando la validación del formulario del lado del cliente (en el propio pc del usuario) la detección de un error es instantánea y no requiere comunicación con el servidor. Sólo cuando el formulario está correcto se permite su envío. Por esto el uso en formularios ha sido una de las aplicaciones principales de JavaScript históricamente.

Si recordamos la jerarquía de objetos JavaScript podemos ver gráficamente la importancia de los formularios y campos de formularios.



Hoy en día el trabajo con formularios no tiene tanta importancia como la tuvo en el pasado, pero sigue siendo un aspecto importante de la programación web y de JavaScript.

TODOS LOS FORMULARIOS EN FORMS

La propiedad forms del objeto document devuelve una colección de objetos tipo array conteniendo todos los formularios que existan en el documento HTML, con índice 0 para el primer formulario por orden de aparición, índice 1 para el siguiente y así sucesivamente.

En este ejemplo vemos cómo podemos acceder a document.forms y obtener el número de formularios existentes en el documento HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
 window.onload = function () {          var ejemplo = document.getElementById('ejemplo');
                                        ejemplo.addEventListener("click", ejecutarEjemplo);       }
function ejecutarEjemplo () {
var formularios = document.forms; alert('El número de formularios en el documento es: '+formularios.length);       }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body></html>
```

TODOS LOS CAMPOS DE FORMULARIOS EN ELEMENTS

Si prestamos atención a los campos de un formulario, cada uno de estos campos puede tener hasta dos identificadores importantes. Por ejemplo:

```
<input id="nombreFormContacto" type="text" name="nombre" />
```

Estos identificadores son el id, que identifica de forma única a un elemento dentro de un documento html (es decir, el id es único en toda la página web), y el atributo name, que identifica de forma única a un elemento dentro de un formulario (es decir, no habrá dos elementos dentro de un formulario con el mismo atributo name, pero sí puede haber elementos en distintos formularios que tengan el mismo atributo name.

Es frecuente que id y name tengan un mismo valor, por ejemplo <input id="nombre" type="text" name="nombre" />, pero no siempre ocurre así.

El atributo id, si existe, nos permitiría acceder a este elemento del formulario usando el método getElementById('valorDeId').

La propiedad name posiblemente no nos sea útil para acceder con getElementsByName debido a que pueden existir varios elementos con el mismo name, sin embargo sí nos va a ser útil con métodos para trabajar con formularios de los que nos provee JavaScript de los que vamos a ir hablando.

La propiedad elements de cada objeto form obtenido mediante document.forms devuelve una colección de objetos tipo array conteniendo todos los campos de formulario que existan en el formulario concreto al que hagamos alusión, con índice 0 para el primer elemento por orden de aparición, índice 1 para el siguiente y así sucesivamente.

Cada campo de formulario es para JavaScript un "HTML Object" y como tal tiene propiedades a las que podemos acceder. Por ejemplo name es una propiedad, value otra propiedad, id otra propiedad, etc.

Analiza este código y comprueba sus resultados en tu navegador para ver cómo elements nos permite acceder a los campos de formulario.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
 window.onload = function () {
            var ejemplo = document.getElementById('ejemplo');
            ejemplo.addEventListener("click", ejecutarEjemplo);
}

function ejecutarEjemplo () {
var msg = '';
var formularios = document.forms;
for (var i=0; i<formularios.length;i++){
            for (var j=0; j<formularios[i].elements.length; j++){
                        msg = msg + '\n\nElemento '+j+ ' del formulario '+(i+1)+ ' tiene id: '+ formularios[i].elements[j].id;
                        msg = msg + ' y name: ' + formularios[i].elements[j].name;
            }
}
alert (msg);
}

</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>

<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
```

```
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body></html>
```

El resultado esperado es que cuando pulsemos en el texto “Pulsa aquí” se muestre por pantalla lo siguiente:

```
Elemento 0 del formulario 1 tiene id: nombreFormContacto y name: nombre
Elemento 1 del formulario 1 tiene id: apellidosFormContacto y name: apellidos
Elemento 2 del formulario 1 tiene id: botonEnvio1 y name:
Elemento 0 del formulario 2 tiene id: motivoFormReclama y name: motivo
Elemento 1 del formulario 2 tiene id: fechaFormReclama y name: fecha
Elemento 2 del formulario 2 tiene id: botonEnvio2 y name:
```

Vemos cómo usando elements hemos accedido a cada uno de los elementos HTML dentro del formulario y mostrado sus propiedades. Vemos cómo los campos name correspondientes a los botones de envío quedan vacíos debido a que estos elementos input no tienen establecido atributo name.

ACCESO DIRECTO A FORMULARIOS MEDIANTE EL ATRIBUTO NAME

Aunque podamos acceder a cualquier elemento mediante el uso de document.forms[indice1].elements[indice2], el acceso mediante índices numéricos posiblemente no resulta cómodo. Además, si introducimos nuevos elementos en los formularios o nuevos formularios, los índices numéricos cambiarán y esto afectaría al diseño de nuestro código.

JavaScript provee una forma cómoda de acceder a los formularios a la que denominamos “acceso directo al formulario a través de su atributo name”. Para ello nos basamos en que cada formulario se mantiene como una propiedad de document a la que se puede acceder escribiendo simplemente:

```
document.valorAtributoNameDelFormulario
```

De esta forma podemos obtener una referencia a un formulario simplemente con una expresión del tipo: var formulario1 = document.formularioContacto; donde formularioContacto es el atributo name de un formulario dentro del documento HTML.

ACCESO DIRECTO A ELEMENTOS DE FORMULARIOS MEDIANTE EL ATRIBUTO NAME

Del mismo modo que podemos acceder a un formulario usando el valor de su atributo name, podemos acceder a elementos de un formulario usando el atributo name de estos elementos con una sintaxis de este tipo:

```
document.valorAtributoNameDelFormulario.valorAtributoNameDelElementoHTML
```

De esta forma podemos obtener una referencia a un elemento simplemente con una expresión del tipo: var elemento1 = document.formularioContacto.apellidos; donde formularioContacto es el atributo name de un formulario dentro del documento HTML y apellidos es el atributo name de un elemento input dentro del formulario. Aunque en una primera lectura puede parecer un trabalenguas, leyéndolo un par de veces y viendo un ejemplo se comprenderá con facilidad.

Veamos el ejemplo. Escribe este código, guárdalo con un nombre como ejemplo.html y visualiza los resultados que se obtienen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
 window.onload = function () {            var ejemplo = document.getElementById('ejemplo');
            ejemplo.addEventListener("click", ejecutarEjemplo); }
function ejecutarEjemplo () {
var formulario1 = document.formularioContacto; dimeDatos(formulario1);
var formulario2 = document.formularioReclamacion; dimeDatos(formulario2);
}
function dimeDatos(formulario){
var msg=''; msg = msg + '\n\nElemento input del formulario '+formulario.name+ ' tiene id: '+ formulario.nombre.id;
msg = msg + '\n\nOtro elemento input del formulario '+formulario.name+ ' tiene id: '+ formulario.apellidos.id;
alert (msg+'\n\n');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Nombre:<input id="nombreFormReclamacion" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormReclamacion" type="text" name="apellidos" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body></html>
```

En este ejemplo var formulario2 = document.formularioReclamacion; nos devuelve el objeto HTML que es el form cuyo atributo name es formularioReclamacion. A su vez document.formularioReclamacion.apellidos nos devuelve el elemento HTML que es el input cuyo atributo name es apellidos.

Estos mismos resultados podrían conseguirse combinando los distintos métodos de acceso a nodos del DOM que ofrece JavaScript (document.getElementById, getElementsByName, childnodes, etc.) o los arrays document.forms y document.elements, pero el acceso basado en nombres resultará cómodo si hemos creado una buena definición de nombres en nuestro documento HTML.

ACCESO CON SINTAXIS MIXTA FORMS – ELEMENTS – NAME – ID

Hay más forma de acceder a los elementos de un formulario, e incluso al formulario en sí.

Podemos acceder a formularios o a los elementos de un formulario con una sintaxis como:

```
var formulario1 = document.forms['valorAtributoIdDelFormulario'];

var elemento1 = formulario1.elements['valorAtributoIdDelElemento']
```

Por ejemplo var formulario1 = document.forms['form1']; nos permite acceder a un formulario cuyo atributo id es form1.

Igualmente podemos acceder a formularios o a los elementos de un formulario con una sintaxis como:

```
var formulario1 = document.forms['valorAtributoNameDelFormulario'];

var elemento1 = formulario1.elements['valorAtributoNameDelElemento'];
```

Por ejemplo var formulario1 = document.forms['formularioContacto']; nos permite acceder a un formulario cuyo atributo name es formularioContacto.

Finalmente, un elemento de un formulario tiene una propiedad que es una referencia al formulario en el cual está inserto. La sintaxis sería del tipo:

```
var formulario1 = document.getElementById['valorIdDelCampo'].form;
```

Suponiendo que un formulario tiene un campo como: <label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>, podemos acceder al formulario usando document.getElementById('apellidosFormContacto').form

EVENTOS RELACIONADOS CON FORMULARIOS

Ya hemos mencionado los eventos relacionados con formularios en apartados anteriores del curso, cuando tratamos los eventos. Recordar que los principales eventos son onfocus (un elemento toma el foco), onblur (pérdida del foco), onchange (cambio), onselect (selección de texto de un elemento input o textarea), onsubmit (pulsar botón de envío, antes del envío), y onreset (pulsar botón de cancelación).

EJERCICIO

Un uso habitual de JavaScript con formularios es usar JavaScript para validar que el contenido introducido por el usuarios sea válido. Crea un formulario que conste de cinco campos: nombre, apellidos, email, ciudad y país. Usando el evento onsubmit, realiza la validación para:

a) Comprobar que en el momento del envío ninguno de los campos tiene menos de dos caracteres (es decir, si está vacío, contiene una letra o dos letras se considerará no válido) accediendo a los campos mediante document.forms y elements.

b) Igual que el apartado a) pero accediendo a los campos directamente usando el atributo name (por ejemplo formularioContacto.apellidos haría alusión a un elemento input cuyo atributo name es apellidos en un formulario cuyo atributo name es formularioContacto)

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01176E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

SETATTRIBUTE Y GETATTRIBUTE JAVASCRIPT. DIFERENCIAS DE SINTAXIS PROPIEDADES CON HTML. EJEMPLO MAXLENGTH. (CU01176E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº76 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

FORMULARIOS. SETATTRIBUTE Y GETATTRIBUTE

JavaScript facilita el trabajo con formularios proveiéndonos de formas de acceso simplificadas a los formularios y sus elementos internos. Estas formas de acceso directo son ligeramente distintas a lo que es el acceso al resto de nodos del DOM, aunque al fin y al cabo lo que logramos es lo mismo: acceder a un nodo.

En su momento describimos el DOM, y ahora creemos oportuno recordar algunas cosas que ya habíamos estudiado para refrescarlas.

Los principales tipos de nodos en el DOM son:

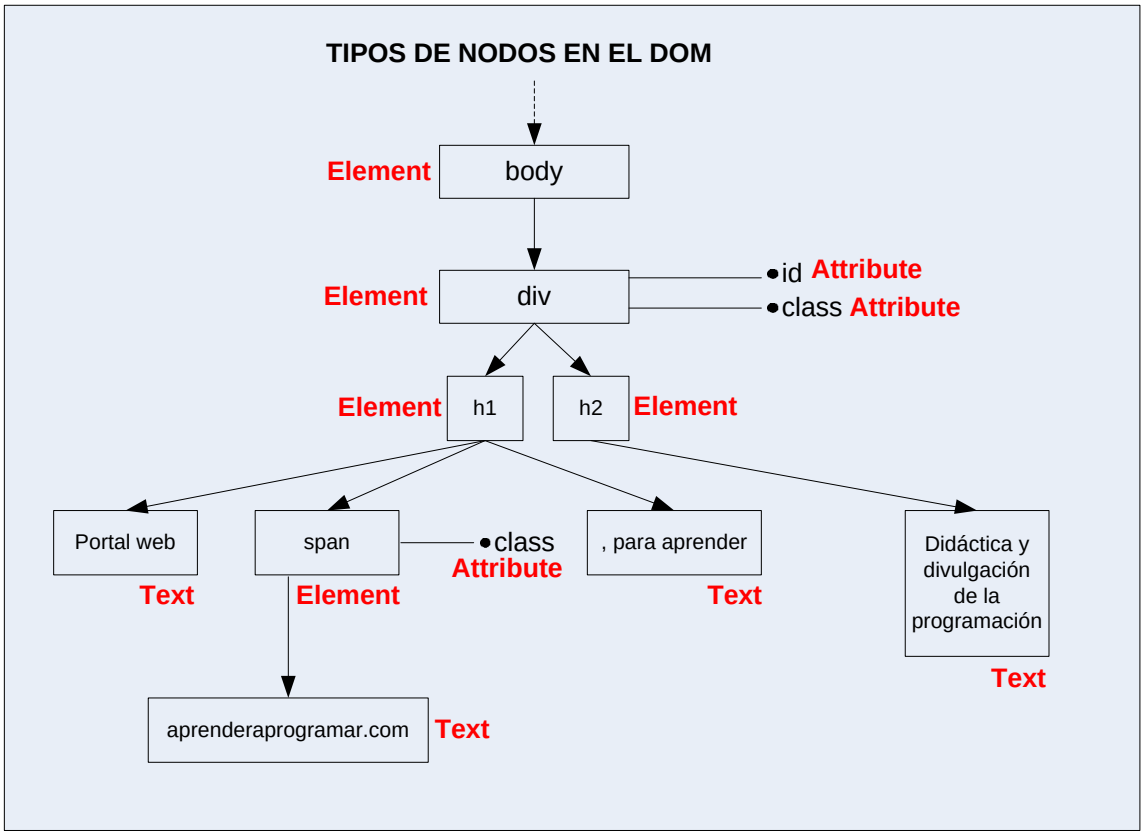
- **Document**: el nodo document es el nodo raíz, a partir del cual derivan el resto de nodos.
- **Element**: son los nodos definidos por etiquetas html. Por ejemplo una etiqueta div genera un nodo. Si dentro de ese div tenemos tres etiquetas p, dichas etiquetas definen nodos hijos de la etiqueta div. Un formulario y los elementos de un formulario son nodos element.
- **Text**: el texto dentro de un nodo element se considera un nuevo nodo hijo de tipo text (texto). Los navegadores también crean nodos tipo texto sin contenido para representar elementos como saltos de línea o espacios vacíos.
- **Attribute**: los atributos de las etiquetas definen nodos, aunque trabajando con JavaScript no los veremos como nodos, sino que lo consideramos información asociada al nodo de tipo element. Para trabajar con formularios nos va a interesar conocer los atributos de los formularios y de los elementos de los formularios. De esta forma podremos recuperarlos para realizar verificaciones o ejecutar condicionales, o modificarlos.
- **Comentarios y otros**: los comentarios y otros elementos como las declaraciones doctype en cabecera de los documentos HTML generan nodos.

Existen más tipos de nodos en el DOM, pero de uso más infrecuente.

Un fragmento de código como este:

```
<body>
<div id="cabecera" class="brillante">
<h1>Portal web <span class="destacado">aprenderaprogramar.com</span>, para aprender</h1>
<h2>Didáctica y divulgación de la programación</h2>
</div>
</body>
```

Se puede ver como un árbol de nodos del DOM:



Una etiqueta como la etiqueta input de un formulario genera un nodo. Sus atributos id, name, size, etc. diremos que son propiedades del nodo. Por ejemplo supongamos un elemento input de tipo texto. Si escribimos algo como var nodo = document.getElementById('nombreFormularioContacto') nos devuelve un nodo, y escribiendo nodo.maxlenght podemos recuperar o establecer un valor para esta propiedad del nodo.

No todos los nodos de tipo element tienen las mismas propiedades disponibles. Por ejemplo un nodo de tipo input de un formulario que es un campo de texto dispondrá de un atributo maxlength, pero un nodo span no dispondrá de este atributo. Las propiedades de los nodos están por tanto ligadas a la definición del estándar HTML, que para cada elemento HTML define una serie de atributos permitidos.

Por tanto, para trabajar con estas propiedades en JavaScript, tendremos que conocer o remitirnos al estándar HTML para saber a qué propiedades podemos acceder.

En general las propiedades son accesibles y modificables mediante JavaScript.

HTML FRENTE A DOM Y maxlength FRENTE A maxLength

Vamos ahora a pensar en los elementos input de los formularios. Estos elementos si son de tipo text tienen como atributo opcional una longitud máxima en número de caracteres. Por ejemplo <input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/> indica que para este input el número máximo de caracteres que podrá introducir el usuario por teclado es 4.

Supongamos que usamos nodo = document.getElementById('nombreFormContacto') para recuperar el nodo del DOM correspondiente a este elemento. Este nodo tiene atributos como type que podemos recuperar como nodo.type. Igualmente podríamos pensar que nodo.maxlength nos debería devolver el valor del atributo maxlength del elemento. Sin embargo no ocurre así, y tenemos que escribir nodo.maxLength con la L mayúscula intermedia para poder recuperar el valor del atributo con JavaScript. ¿Por qué?

Pues básicamente por el mismo motivo por el que no podemos acceder a las propiedades CSS de los elementos con exactamente los mismos nombres en JavaScript y en CSS. Por ejemplo, podemos recordar que en CSS escribimos background-color: red; mientras que en JavaScript usamos una sintaxis como nodo.style.backgroundColor = 'red';

JavaScript nos da acceso al DOM y a la manipulación de elementos del DOM, pero no se ciñe siempre a la sintaxis especificada por estándares como CSS ó HTML.

Muchas propiedades se escriben igual en HTML que con JavaScript. Por ejemplo la propiedad type de HTML encuentra su equivalente en una sintaxis de tipo nodo.type. Pero no siempre ocurre así.

En el caso de maxlength, para acceder a esta propiedad a través de JavaScript hemos de escribir maxLength. En general podemos decir que se sigue la sintaxis “camelCase”, pero siempre debemos hacer comprobaciones porque puede haber casos especiales.

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
 window.onload = function () {
            var ejemplo = document.getElementById('ejemplo');
            ejemplo.addEventListener("click", ejecutarEjemplo);
}

function ejecutarEjemplo () {
var msg = '';
var formularios = document.forms;
for (var i=0; i<formularios.length;i++){
            for (var j=0; j<formularios[i].elements.length; j++){
                        if (formularios[i].elements[j].type=='text' ) {
                        msg = msg + '\n\nValor inicial maxlength: '+formularios[i].elements[j].maxLength;
                        msg = msg + '\n\nFijada longitud máxima 5 para elemento con id: '+formularios[i].elements[j].id;
                        formularios[i].elements[j].maxLength=5;
                        }
            }
}
alert (msg);
}

</script>
</head>
```

```
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body></html>
```

El resultado que hemos obtenido con este código ha sido que después de pulsar sobre "Pulsa aquí" las casillas de texto quedan con un máximo de 5 caracteres admitidos para entrada y que por pantalla se muestra los siguiente:

Navegadores 1 y 2	Navegador 3
Valor inicial maxlength: 4 maxlength ahora 5 para: nombreFormContacto Valor inicial maxlength: -1 maxlength ahora 5 para: apellidosFormContacto Valor inicial maxlength: -1 maxlength ahora 5 para: motivoFormReclama Valor inicial maxlength: -1 maxlength ahora 5 para: fechaFormReclama	Valor inicial maxlength: 4 maxlength ahora 5 para: nombreFormContacto Valor inicial maxlength: 2147483647 maxlength ahora 5 para: apellidosFormContacto Valor inicial maxlength: 2147483647 maxlength ahora 5 para: motivoFormReclama Valor inicial maxlength: 2147483647 maxlength ahora 5 para: fechaFormReclama

Lo que nos llama la atención de este resultado es: cómo distintos navegadores devuelven un resultado distinto cuando maxlength no está establecida (en unos casos obtenemos -1, que interpretamos como una señal de valor no establecido, y en otros 2147483647 que interpretamos como el valor más grande para un entero, que interpretamos como que no existe límite en el número de caracteres que se pueden introducir en el input.

Prueba ahora a sustituir en el código anterior maxLength por maxlength. Abre la consola de depuración de tu navegador y comprueba cómo no aparece ningún error. Sin embargo, después de hacer click en "Pulsa aquí" los input siguen sin cambiar el límite de caracteres admitido. ¿Por qué? En este caso, lo que estaríamos haciendo al invocar formularios[i].elements[j].maxlength sería invocar una propiedad que no existe sobre un objeto, y esto nos devuelve undefined, que es precisamente lo que se muestra por pantalla. Al escribir formularios[i].elements[j].maxlength=5; no estamos modificando el atributo que limita el máximo número de caracteres admisible, sino que estamos creando un nuevo atributo al que hemos denominado maxlength con minúsculas, y que hemos establecido con valor 5. Pero esto es lo mismo que si hubiéramos escrito formularios[i].elements[j].lechugas = 5; ya que podemos escribir una nueva propiedad de un objeto y asignarle un valor en el momento en que queramos.

ENCONTRAR LAS EQUIVALENCIAS ENTRE HTML Y JAVASCRIPT

El HTML y el propio JavaScript son realidades relativamente cambiantes: a medida que surgen nuevos estándares, nuevas versiones de navegadores, van apareciendo nuevas formas sintácticas y propiedades, por lo que es casi imposible encontrar un lugar donde tengamos todas las propiedades HTML y sus equivalentes para JavaScript.

Nos encontramos ante un pequeño problema: ¿cómo encontrar las equivalencias entre la sintaxis HTML y la sintaxis JavaScript?

De forma orientativa podemos seguir las siguientes reglas:

1) Escribe la propiedad en JavaScript de la misma forma que lo harías en HTML. Esto es válido para la gran mayoría de propiedades como value, src, type, etc.

2) Si no funciona lo anterior usa sintaxis camelcase, por ejemplo maxLength para JavaScript frente a maxlength para HTML.

3) Si tienes dudas sobre la sintaxis o existencia de una propiedad en HTML, busca la referencia HTML correspondiente. Por ejemplo, si quieres utilizar un elemento input y conocer sus propiedades válidas en HTML puedes recurrir a un libro o a una búsqueda en internet donde buscaríamos <<input HTML>> ó <<input HTML mozilla developer>>. En este caso hemos optado por incluir <<mozilla developer>> en la búsqueda para acceder a la documentación que facilita la fundación Mozilla al respecto, que suele ser de buena calidad en líneas generales. Por ejemplo, encontraríamos que nos aparece maxlength y nos indica “If the value of the type attribute is text, email, search, password, tel, or url, this attribute specifies the maximum number of characters (in Unicode code points) that the user can enter”

4) Para conocer el equivalente para JavaScript nos hemos de remitir a la especificación del DOM. Por tanto haríamos una búsqueda en internet del texto <<html input element dom>> ó <<html input element dom mozilla developer>>. En los resultados de búsqueda comprobaremos que existe la propiedad maxLength, de la que se indica “Reflects the maxlength HTML attribute, containing the maximum length of text…”

SETATTRIBUTE Y GETATTRIBUTE

JavaScript provee más formas de acceder y modificar los atributos HTML relacionados con un nodo. Vamos a ver una de ellas. Podemos usar esta sintaxis para establecer el valor de un atributo:

```
nodo.setAttribute('nombreAtributoHMTL', 'valorAtributoHTML');
```

O esta otra sintaxis para recuperar el valor del atributo:

```
nodo.getAttribute('nombreAtributoHMTL');
```

En general se recomienda el acceso directo a las propiedades, por ejemplo nodo.type = 'text'; en lugar de usar nodo.setAttribute('type', 'text'); pero la alternativa está ahí y debe conocerse.

En este caso, donde se indica nombreAtributoHTML tendremos que usar la forma de nombrar el atributo con HTML. Por ejemplo si nos referimos a maxlength, habremos de escribirlo todo en minúsculas.

EJERCICIO

Dado el siguiente código HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="ejemplo">Pulsa aquí</div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body>
</html>
```

Crea un script que recorra todos los elementos input dentro de cada uno de los formularios presentes y si son de tipo text, modifique su atributo asociado maxlength usando el método setAttribute de los objetos tipo Element para limitar a 8 el número máximo de caracteres que pueda introducir el usuario. Una vez modificado el atributo, muestra por pantalla el valor que tiene dicho atributo para todos los elementos de tipo input usando el método getAttribute.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01177E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FUNCIONES MANEJADORAS DE EVENTOS Y ADDEVENTLISTENER CON PARÁMETROS. THIS.STYLE IS UNDEFINED (CU01177E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº77 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

ADDEVENTLISTENER CON PARÁMETROS

Vamos a seguir estudiando formularios, pero vamos a detenernos en ver cómo podemos utilizar el método addEventListener con parámetros. Con lo que hemos estudiado anteriormente sobre ámbito de variables, closures, etc. vamos a ser capaces de resolverlo, aunque no es sencillo y merece la pena estudiarlo paso a paso.

Nos planteamos querer añadir respuesta a eventos cuando el usuario hace click sobre un inputbox de un formulario.

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo1 aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
window.onload = function () {
        var formularios = document.forms;
        formularios['formularioContacto'].elements['nombreFormContacto'].addEventListener('click', cambiaColor);
        formularios['formularioContacto'].elements['apellidosFormContacto'].addEventListener('click', cambiaColor);
}
function cambiaColor () {this.style.backgroundColor='yellow'; }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form></body></html>
```

El resultado esperado es que cuando hagamos click sobre un inputbox del primer formulario, éste pase a tener color de fondo amarillo.

Es interesante ver cómo dentro de la función cambiaColor, la referencia this nos devuelve el objeto HTML que dispara el evento. ¿Por qué? Se ha explicado en anteriores entregas del curso, pero lo recordamos aquí: this usada dentro de una función manejadora de evento nos devuelve el nodo de tipo

Element definido por las etiquetas HTML que reciben el evento. Y en este caso cambiaColor es una función manejadora del evento click para los input de los formularios.

Pero el código planteado no es muy eficiente: si quisiéramos hacer esto mismo para todos los inputbox de todos los formularios y tuviéramos muchos, tendríamos que escribir mucho código a mano.

Además expresiones como formularios['formularioContacto'].elements['apellidosFormContacto'] no nos generan mucha seguridad porque ¿qué ocurre si se cambian los nombres asociados a los formularios o los campos? El código dejaría de funcionar.

Por ello veremos vamos a intentar buscar una solución más genérica (más abstracta). Pero primero repasaremos algunos conceptos.

FUNCIONES MANEJADORAS DE EVENTOS CON NINGÚN O UN PARÁMETRO

Recordamos ahora cosas ya explicadas anteriormente en el curso pero que conviene repasar. Una función manejadora de un evento recibe "de forma automática" un objeto de tipo Event. Este objeto es un argumento que se pasa automáticamente cuando se invoca la función al dispararse el evento. El nombre de este argumento es el que nosotros decidamos, es decir, podemos llamarlo evobject, eventoDatos, miEvento u objetoRepresentaEvento, etc. y para poder usarlo hemos de declararlo como parámetro de la función.

Ejemplo:

```
function cambiaColor (elEvento) {
alert('Detectado evento de tipo: '+elEvento.type);
this.style.backgroundColor='yellow';
}
```

En el ejemplo anterior, el parámetro el evento se recibe siempre que la función se invoque como respuesta a un evento. Con el código que hemos visto previamente y esta función manejadora del evento, cuando se hiciera click en el inputbox se mostraría por pantalla << Detectado evento de tipo: click>> y seguidamente el color del cuadro del inputbox pasaría a ser amarillo.

¿Podría una función manejadora de evento invocarse sin ser como respuesta a un evento? Sí, aunque hay que tener en cuenta cómo funciona JavaScript a este respecto.

Si estamos usando la función <<function cambiaColor ()>> que carece de parámetros significa que el objeto tipo Event que se envía automáticamente cuando sucede el evento no es recuperable ¿Por qué? Porque para poder recuperarlo necesitamos especificar un parámetro en la definición de la función que represente al objeto Event que se envía automáticamente. En un caso así podemos invocar la función de un modo como este: <<cambiaColor.call(document.body);>> dando como resultado que la función se ejecutará tomando como this al objeto document.body.

En cambio si estamos usando la función <<function cambiaColor (elEvento)>> y hacemos una invocación como <<cambiaColor.call(document.body);>> podemos obtener un error del tipo 'elEvento is undefined'. Este error aparece si tratamos de usar el parámetro (que es el Evento que se envía

automáticamente en segundo plano) cuando en este caso no ha existido evento, sino simplemente una invocación directa para la ejecución de la función.

Podríamos usar un código como este para diferenciar si la función se ejecuta como respuesta al evento o por invocación directa desde el código:

```
function cambiaColor (elEvento) {
if(elEvento){ alert('Detectado evento de tipo: '+elEvento.type); }
this.style.backgroundColor='yellow';
}
```

De esta manera, si no se recibe el parámetro, no se intenta hacer uso de él.

FUNCIONES MANEJADORAS DE EVENTOS CON MÁS DE UN PARÁMETRO

Supongamos que queremos enviar a la función manejadora del evento un parámetro que represente el color de fondo que debe adquirir el elemento.

Podríamos pensar en algo como esto (mala idea):

```
function cambiaColor (elColor) {
alert(elColor);
if(elColor) {this.style.backgroundColor=elColor;}
else {this.style.backgroundColor='yellow'; alert(this);}
}
```

Esta idea falla porque no tiene en cuenta una cosa: si la función se ejecuta como respuesta a un evento, el primer argumento (no explícito, pero existente) es el objeto Event asociado al evento. Con un código como el anterior tendríamos este resultado:

Si la función se invoca desde código con una sintaxis como cambiaColor.call(document.body, 'pink'); el resultado es: << Argumento recibido: pink>>

Si la función se invoca como respuesta a un evento el resultado es: <<Argumento recibido: [object MouseEvent]>>

Si pretendemos que esta sea una función manejadora de eventos a la que le podamos pasar un parámetro como el color deseado, deberemos escribirla siguiendo esta idea:

```
function cambiaColor (elEvento, elColor) {
if(elEvento){ alert('Detectado evento de tipo: '+elEvento.type); }
if(elColor){ alert('Argumento recibido: '+elColor); }
if(elColor) {this.style.backgroundColor=elColor;}
else {this.style.backgroundColor='yellow';}
}
```

Si quisiéramos invocar de forma directa esta función, tendríamos que hacerlo incluyendo dos parámetros porque la definición de la función así lo requiere. Pero si no hay evento, ¿qué ponemos como primer parámetro? Tendremos que decidirlo, pero por ejemplo podemos simplemente pasar la cadena vacía. La función la podríamos invocar directamente desde código con una sintaxis como esta:

cambiaColor.call(document.body, '', 'pink');

Aquí vemos que el argumento correspondiente al objeto Event, dado que no hay evento, lo enviamos como cadena vacía. La función, si es que va a usarse de esta manera, tendría que tener previsto el tratamiento oportuno.

Posiblemente en nuestro código las funciones manejadoras de eventos no van a ser invocadas directamente, pero el ejemplo anterior nos ha servido para comprender mejor la lógica a aplicar con este tipo de funciones.

BUSCANDO UNA SOLUCIÓN MÁS GENÉRICA PARA MEJORAR NUESTRO CÓDIGO

Volvemos a nuestro código con dos formularios, sobre el que estamos tratando de hacer que el color de fondo de las cajas de los inputbox cambie cuando hagamos click sobre ellos.

Vamos a tratar de buscar una solución más genérica (más abstracta) que la vista antes.

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo2 aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
			if (formularios[i].elements[j].type=='text' ) {
				formularios[i].elements[j].addEventListener('click', cambiaColor);
			}
		}
	}
}

function cambiaColor () {
this.style.backgroundColor='yellow';
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
```

```
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form></body></html>
```

El resultado esperado es que ahora todas las cajas de texto de los inputbox adquieren color de fondo amarillo cuando pulsamos sobre ellos. Además, hemos definido que se recorran todos los elementos de todos los formularios y se añada el escuchador de evento correspondiente, y el código seguirá funcionando aunque cambien los atributos id ó name de los elementos input.

Ahora nos vamos a plantear lo siguiente: queremos que el color de fondo de un inputbox se convierta en amarillo si es el primer, tercer, quinto, séptimo, etc. elemento dentro del formulario, o que se convierta en verde si es el segundo, cuarto, sexto, octavo, etc. elemento dentro del formulario.

Podemos plantearlo de dos maneras:

a) Desde la función cambiaColor, recuperar de alguna manera la posición en que se encuentra el inputbox del formulario. No vamos a estudiar esta opción.

b) Pasarle a la función de manejo del evento un parámetro que le diga si el elemento es impar o par dentro del formulario. Vamos a tratar de resolver esta cuestión porque es un buen ejemplo de paso de parámetros a una función manejadora de eventos.

Inicialmente podría pensarse en una solución de este tipo:

```
if (formularios[i].elements[j].type=='text' && j%2==0 ) {
                                        formularios[i].elements[j].addEventListener('click', cambiaColorPonAmarillo);
                                }
                                else {
                                formularios[i].elements[j].addEventListener('click', cambiaColorPonNaranja);
                                }
```

Pero esto no es interesante: ¿qué ocurriría si quisiéramos poner 10 colores distintos? ¿Tendríamos que crear 10 funciones con 10 nombres distintos? ¿Y si en un momento dado queremos cambiar los colores?

Esto sería poco útil. Nuestra idea es pasar un parámetro a la función manejadora del evento con el color.

Podríamos pensar en redefinir la función cambiaColor como function cambiaColor (elEvento, elColor) { ... } y escribir algo como esto. Pruébalo y comprueba los resultados:

```
                if (formularios[i].elements[j].type=='text' && j%2==0 ) {
                        formularios[i].elements[j].addEventListener('click', cambiaColor('', 'yellow'));
                }
                else {      formularios[i].elements[j].addEventListener('click', cambiaColor('', 'orange')); }
```

Pero esto tiene varios problemas (activa la consola del navegador para ver los mensajes de error si no la tienes activada): pasamos un argumento evento vacío, ¿por qué si aquí estamos con una función manejadora de eventos? No tiene sentido. Otro problema es que addEventListener espera una referencia a una función, mientras que el código anterior ejecuta la función, lo que es distinto.

Para definir correctamente una referencia a una función tendríamos que definir una función anónima.

Nos vamos aproximando a una solución si planteamos esto, aunque todavía no funciona. Prueba este código y comprueba qué sucede:

```
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
		if (formularios[i].elements[j].type=='text' && j%2==0 ) {
			formularios[i].elements[j].addEventListener('click', function() {cambiaColor('', 'yellow');});
		}
		else {	formularios[i].elements[j].addEventListener('click', function(){cambiaColor('', 'orange');});
	}
		}
	}
}

function cambiaColor (elEvento, elColor) { alert(elEvento + '***'+elColor + ' - ' + this);
this.style.backgroundColor=elColor;
}
```

¿Qué ocurre con este código? En primer lugar que dentro de la función anónima estamos creando un ámbito nuevo, y this ya no es el elememento html que recibe el evento. ¿Qué es entonces this? Como siempre que this no está definido en un ámbito, es el objeto window. Al invocar this.style.backgroundColor=elColor; obtenemos un error del tipo this.style is undefined porque al objeto window no le podemos aplicar estilos (ya que no es un elemento HTML). Además estamos perdiendo el objeto Event al pasar un argumento vacío. Activa la consola si no la tienes activada para poder ver los mensajes de error.

STOP FOR A MINUTE

Lo que estamos explicando es un tanto engorroso, pero llegar a comprender todo lo que estamos discutiendo es interesante de cara a la comprensión de JavaScript. Si vienes siguiendo el curso desde el principio, deberías ser capaz de seguir estas explicaciones. Si estás siguiendo el curso y te has perdido, vuelve atrás y relee las explicaciones y haz pruebas con el código con calma. Incluso déjalo para mañana, siempre es útil descansar y retomarlo al día siguiente. Si después de releer esta entrega se te sigue atragantando, te recomendamos escribir una consulta en los foros aprenderaprogramar.com.

CONTINUAMOS

Repasamos conceptos: this es el elemento HTML que nos interesa dentro de la función anónima que es ahora la manejadora del evento, pero ya no lo es en la función cambiaColor porque ésta ya no es la manejadora del evento, sino una función invocada por la manejadora del evento.

Ahora la función manejadora del evento es la función anónima, por lo que si quisiéramos pasar el evento tendríamos que hacer algo así:

```
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
		if (formularios[i].elements[j].type=='text' && j%2==0 ) {
			formularios[i].elements[j].addEventListener('click', function(elEvento)
{cambiaColor(elEvento, 'yellow');});
		}
		else {	formularios[i].elements[j].addEventListener('click',
function(elEvento){cambiaColor(elEvento, 'orange');});	}

		}
	}
}

function cambiaColor (elEvento, elColor) {
this.style.backgroundColor=elColor;
}
</script>
```

Ahora estamos pasando el evento, pero todavía no hemos resuelto el problema de haber perdido la referencia al this.

Podríamos pensar en intentar algo como esto:

```
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
		if (formularios[i].elements[j].type=='text' && j%2==0 ) {
			formularios[i].elements[j].addEventListener('click', function(elEvento) {
			cambiaColor(elEvento, 'yellow', formularios[i].elements[j]);});
		}
		else {	formularios[i].elements[j].addEventListener('click', function(elEvento){
		cambiaColor(elEvento, 'orange',formularios[i].elements[j]);});	}
		}
	}
}
function cambiaColor (elEvento, elColor, quienEsThis) {
quienEsThis.style.backgroundColor=elColor;
}
</script>
```

Ahora nos enfrentamos a un error de tipo <<TypeError: formularios[i] is undefined>> porque la función anónima define un nuevo ámbito, independiente puesto que es un argumento de addEventListener y no una función anidada dentro de la función de respuesta a window.onload.

Finalmente vamos a llegar a una solución válida que consiste en pasarle el this a la función cambiaColor. Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
			if (formularios[i].elements[j].type=='text' && j%2==0 ) {
				formularios[i].elements[j].addEventListener('click', function(elEvento) {
				cambiaColor(elEvento, 'yellow', this);});

			}
			else {	formularios[i].elements[j].addEventListener('click', function(elEvento){
			cambiaColor(elEvento, 'orange', this);});		}
		}
	}
}

function cambiaColor (elEvento, elColor, quienEsThis) {
quienEsThis.style.backgroundColor=elColor;
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>

<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" maxlength="4"/></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>

<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body>
</html>
```

¿PODEMOS MEJORAR?

En el código que se ha propuesto como solución, cada vez que se añade un manejador de eventos a un elemento HTML estamos creando una función anónima. Esto puede resultar ineficiente. Más adelante estudiaremos cómo podemos mejorar este código para hacerlo más eficiente.

EJERCICIO

Crea un código HTML donde tengas un formulario con seis inputbox de texto que servirán para pedir al usuario Nombre, Apellidos, Correo electrónico, Teléfono, Domicilio y País. Crea el código JavaScript para que los elementos 1, 4, 7, 10 etc. del formulario tomen color de fondo amarillo cuando el usuario pulse sobre ellos. Los elementos 2, 5, 8, 11, etc. del formulario deberán tomar color de fondo azul claro cuando el usuario pulse sobre ellos. Los elementos 3, 6, 9, 12, 15, etc. del formularios deberán tomar color de fondo verde claro cuando el usuario pulse sobre ellos.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01178E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

BIND JAVASCRIPT . FUNCIÓN ENTRE PARÉNTESIS QUE LA ENVUELVEN. FUNCTION STATEMENT REQUIRES A NAME (CU01178E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº78 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

BIND JAVASCRIPT

El método bind de los objetos Function en JavaScript tiene distintas aplicaciones que nos van a hacer más cómoda la programación. Tiene similitudes (y diferencias) con los métodos call y apply. Hemos pospuesto el estudio de bind hasta este momento precisamente para tener conocimientos suficientes que nos permitan valorar lo que hace este método.

Nos planteamos querer obtener los índices de los elementos dentro de los formularios que sean de tipo input y type text, es decir, cajas de texto para entrada de datos en un formulario. Los formularios existentes en el código html los obtenemos con var formularios = document.forms; donde formularios[0] será el primer formulario, formularios[1] el segundo formulario y así sucesivamente

Con formularios[0].elements[0] accedemos al primer elemento dentro del formulario, con formularios[0].elements[1] accedemos al segundo elemento dentro del formulario y así sucesivamente.

Escribe este código y comprueba sus resultados (activa primero la consola del navegador):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
			if (formularios[i].elements[j].type=='text' ) {
				formularios[i].elements[j].addEventListener('click', dimeLosIndices(i,j));	}
		}
	}
}
function dimeLosIndices (indiceI, indiceJ) {console.log('Has pulsado en el inputbox cuyos índices son i='+indiceI+' ,
j='+indiceJ);}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label></form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form></body></html>
```

El resultado esperado es que se muestre por consola lo siguiente:

Has pulsado en el inputbox cuyos índices son i=0 , j=0
Has pulsado en el inputbox cuyos índices son i=0 , j=1
Has pulsado en el inputbox cuyos índices son i=1 , j=0
Has pulsado en el inputbox cuyos índices son i=1 , j=1

Ya sabemos que esto ocurre porque en lugar de pasar una referencia de función en el addEventListener estamos invocando la ejecución de la función.

Si modificamos la línea del addEventListener y escribimos esto:

```
formularios[i].elements[j].addEventListener('click', function(){dimeLosIndices(i,j);});
```

El resultado es que pulsemos sobre el inputbox que pulsemos siempre se muestra: "Has pulsado en el inputbox cuyos índices son i=2 , j=3" ¿Por qué?

Porque los índices i=2 y j=3 son los últimos índices que toman las variables en el bucle. En todos los inputbox el comportamiento es: cuando se pulsa, se dispara el evento click, y con el evento click se ejecuta la función dimeLosIndices pasándole los parámetros i, j actuales, que son una vez finalizada la ejecución del bucle, i=2, j=3.

No hemos resuelto lo que queríamos, ya que queremos “ligar” a cada inputbox una función manejadora de evento que se dispare con sus propios índices, no con los índices existentes al final del bucle.

CLOSURES Y ¿UNA FUNCIÓN ENVUELTA EN PARÉNTESIS?

Podemos resolverlo con closures. Si no recuerdas los closures y su significado lee las anteriores entregas del curso. Escribe este código y comprueba los resultados (activa primero la consola del navegador):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">

<style type="text/css">
label{display:block;margin:5px;}
</style>

<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
			if (formularios[i].elements[j].type=='text' ) {
```

```
                                        (function(){
                                        var closureI =i;
                                        var closureJ =j;
                                        formularios[i].elements[j].addEventListener('click', function(){
                                        dimeLosIndices(closureI,closureJ);});
                                    })(); //Creamos un ámbito con la función anónima y la ejecutamos con ()
                                    }

                            }
                    }
}

function dimeLosIndices (indiceI, indiceJ) {
console.log('Has pulsado en el inputbox cuyos índices son i='+indiceI+' , j='+indiceJ);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>

<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>
<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form></body></html>
```

El resultado es que ahora sí se mostrarán correctamente los índices. Por ejemplo si pulsamos sobre el primer inputbox del primer formulario se mostrará: "Has pulsado en el inputbox cuyos índices son i=0 , j=0"

En el código anterior tenemos algo interesante además de los closures, que ya hemos explicado en apartados anteriores del curso: una función envuelta en paréntesis con una sintaxis como esta:

(function () { //Contenido de la función

} () **)**

Bien, con function () { ... } () lo que estamos haciendo es invocar la ejecución de la función inmediatamente después de haberla creado. ¿Pero qué hacen los paréntesis exteriores?

Los paréntesis exteriores están ahí indicándole al intérprete JavaScript que ese fragmento de código es una expresión y no una declaración de función. Una expresión se ejecuta dando lugar a una

equivalencia en código, mientras que una declaración de función implica definir un nombre para una función que puede ser posteriormente invocada.

En el contexto en el que estamos no podemos declarar una función (daría lugar a un error del tipo SyntaxError: function statement requires a name), pero sí podemos usar una expresión utilizando una función para lograr nuestros objetivos.

BIND JAVASCRIPT

El uso de funciones anónimas y closures en el código anterior deriva de la imposibilidad de pasar parámetros a referencias de funciones: si pasamos parámetros, invocamos la función. Sin embargo las últimas versiones de JavaScript incorporaron un método para los objetos de tipo Function que permite ligar parámetros a referencias de funciones sin necesidad de ejecutar la función: el método bind. Este método puede que no sea reconocido por algunos navegadores antiguos.

El método bind recuerda en cierta manera a los métodos call y apply que ya hemos estudiado. La sintaxis que emplearemos será habitualmente:

```
var copiaFunc = nombreDeLaFuncion.bind(objetoQueSeráThis, opcArg1, opcArg2, ...);
```

El método bind crea una copia de la función original que será ejecutada siempre que se invoque con el objeto this que se haya especificado y con los argumentos que se hayan especificado. Es importante reseñar que bind, a diferencia de call y apply, no da lugar a la ejecución de la función.

Si en un caso dado no tiene relevancia qué objeto actuará como this, se puede pasar entonces undefined en el lugar que ocupa este parámetro.

Escribe este código y comprueba su resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
function suma3(x, y, z) {return x+y+z;}
var suma3Primos = suma3.bind(undefined, 2, 3, 5);
alert (suma3Primos());
alert(suma3Primos()+15);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue; margin:20px;" id ="pulsador" onclick="ejemplo()">Pulsa aquí</div>
</body>
</html>
```

El resultado esperado es que se muestre por pantalla: 10, 25.

Fíjate que escribimos suma3Primos() con los paréntesis finales porque suma3Primos es una función. Decimos que es una función ligada (binded) a la función suma3 a través del método bind.

Dado que bind no ejecuta la función, sino que crea una copia en la cual "fijamos" qué objeto actuará como this y cuáles serán los parámetros con los que será invocada la copia de la función, podemos escribir el código para mostrar los índices de los inputs de formularios en un documento HTML de una forma más simple que la que vimos anteriormente (basada en closures y funciones anónimas, un tanto engorrosa en su definición).

Escribe este código y comprueba sus resultados (activa la consola del navagador si no la tienes activada):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css"> label{display:block;margin:5px;}</style>
<script type="text/javascript">
window.onload = function () {
	var formularios = document.forms;
	for (var i=0; i<formularios.length;i++){
		for (var j=0; j<formularios[i].elements.length; j++){
			if (formularios[i].elements[j].type=='text' ) {
				formularios[i].elements[j].addEventListener('click', dimeLosIndices.bind(this,i,j));

			}

		}
	}
}

function dimeLosIndices (indiceI, indiceJ) {
console.log('Has pulsado en el inputbox cuyos índices son i='+indiceI+' , j='+indiceJ);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>

<form name ="formularioContacto" method="get" action="accion1.html">
<h2>Formulario de contacto</h2>
<label>Nombre:<input id="nombreFormContacto" type="text" name="nombre" /></label>
<label>Apellidos:<input id="apellidosFormContacto" type="text" name="apellidos" /></label>
<label><input id ="botonEnvio1" type="submit" value="Enviar"></label>
</form>

<form name ="formularioReclamacion" method="get" action="accion2.html">
<h2>Formulario de reclamación</h2>
<label>Motivo reclamación:<input id="motivoFormReclama" type="text" name="motivo" /></label>
<label>Fecha del hecho:<input id="fechaFormReclama" type="text" name="fecha" /></label>
<label><input id="botonEnvio2" type="submit" value="Enviar"></label>
</form>
</body></html>
```

El resultado es el mismo que obtuvimos antes: se muestran correctamente los índices. Por ejemplo si pulsamos sobre el primer inputbox del primer formulario se mostrará: "Has pulsado en el inputbox cuyos índices son i=0 , j=0"

Al escribir dimeLosIndices.bind(this,i,j) como argumento de addEventListener estamos creando una copia de la función (función ligada) con su objeto this y parámetros específicos para cada elemento input en el que introducimos un manejador de eventos.

Este código es más claro que el basado en closures y uso de funciones anónimas, por lo tanto es preferible.

RESUMEN BIND JAVASCRIPT

El uso de bind tiene distintas aplicaciones en JavaScript y en general nos facilita la programación. Normalmente lo usaremos para mantener una referencia al objeto que queremos que actúe como this, para mantener referencia a parámetros asociados a una función, o para ambas cosas.

En muchos casos nos puede ahorrar el escribir funciones anónimas y usar closures, lo que a la larga se traduce en un código más limpio y legible. Sin embargo, bind no es una varita mágica que haya que usar para resolverlo todo. Hay que buscar un equilibrio: usar, no abusar, de este método.

EJERCICIO

Escribe este código, ejecútalo y responde a las preguntas que aparecen a continuación.

```
function conversor(toUnit, factor, offset, input) {
    offset = offset || 0;
    return [((offset+input)*factor).toFixed(2), toUnit].join(" ");
}

var milesToKm = conversor.bind(undefined, 'km', 1.60936, 0);
var poundsToKg = conversor.bind(undefined, 'kg', 0.45460, 0);
var farenheitToCelsius = conversor.bind(undefined, 'gradosC',0.5556, -32);

alert(milesToKm(10));
alert(poundsToKg(2.5));
alert(farenheitToCelsius(98));
```

a) ¿Cuántas funciones ligadas (copias parametrizadas) de la función conversor se crean en este código?

b) ¿Qué es lo que devuelve la función conversor?

c) ¿Qué objeto actúa como this en la función milesToKm?

d) ¿Qué tarea cumple y con qué fórmula trabaja la función milesToKm?

e) ¿Qué tarea cumple y con qué fórmula trabaja la función poundsToKg?

f) ¿Qué resultado devuelve milesToKm(10) y qué significa este resultado?

g) ¿Qué resultado devuelve poundsToKg(2.5) y qué significa este resultado?

h) ¿Qué resultado devuelve farenheitToCelsius(98) y qué significa este resultado?

i) ¿Qué significado tiene la expresión offset = offset || 0;?

j) ¿Cuál es la finalidad del uso de toFixed(2)?

k) ¿Por qué crees que se usa el parámetro offset en la función conversor?

l) Modifica el código para ampliar la información que se nos muestra: introduce un parámetro fromUnit en la función conversor de modo que el resultado nos informe del dato de origen, sus unidades, y el dato convertido y sus unidades.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01179E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

EVENTOS DE FORMULARIOS: CHANGE. OPTIONS, SELECTEDINDEX OBTENER VALUE DE INPUT, CHECKED Y TEXT DE SELECT (CU01179E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº79 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MÁS JAVASCRIPT EN FORMULARIOS

Cuando trabajamos con formularios es habitual que queramos introducir efectos dinámicos a medida que el usuario realiza selección de opciones. Por ejemplo, si se trata de una compra y el usuario selecciona para con tarjeta Visa, en un lateral podemos mostrar la comisión aplicable. Si cambia la elección a tarjeta Mastercard, podemos hacer que automáticamente cambie la comisión aplicable utilizando JavaScript.

RECUPERAR VALUE PARA TEXT Y TEXTAREA

Para recuperar el contenido de un cuadro de texto correspondiente a un input de tipo text ó textarea usamos el atributo value del elemento HTML. La sintaxis será similar a:

```
document.getElementById('apellidos').value
```

El evento para detectar cambios en los cuadros de texto y textarea normalmente será “onchange”.

Veremos el ejemplo de código después de explicar otros elementos.

RECUPERAR CHECKED PARA UN CONJUNTO DE RADIO BUTTONS

Cuando tenemos un conjunto de radio buttons, cada uno de ellos tiene un value, pero lo que nos interesa saber es cuál de los radio buttons de los posibles ha sido elegido. En un conjunto de radio buttons, sólo uno puede estar marcado (o no estar marcado ninguno), y ese elemento es el que nos interesará. Para determinar el elemento marcado usamos la propiedad checked. La sintaxis será similar a:

```
var radioButTrat = document.getElementsByName("nameUtilizado");
for (var i=0; i<radioButTrat.length; i++) {
if (radioButTrat[i].checked == true) { ... código a ejecutar }
}
```

El evento para detectar cambios en los conjuntos radio button normalmente será “onclick”.

Veremos el ejemplo de código después de explicar otros elementos.

RECUPERAR CHECKED PARA UN CONJUNTO DE CHECKBOX

Cuando tenemos un conjunto de checkbox, cada uno de ellos tiene un value, pero lo que nos interesa saber es qué casillas han sido marcadas (a diferencia de los radio buttons, en el caso de los checkbox podemos tener marcada ninguna, una o varias casillas). Para ello usamos la propiedad checked. La sintaxis será similar a:

```
var radioButTrat = document.getElementsByName("tratamiento");
for (var i=0; i<radioButTrat.length; i++) {
if (radioButTrat[i].checked == true) { ... código a ejecutar }
}
```

El evento para detectar cambios en los conjuntos checkbox normalmente será “onchange”.

Veremos el ejemplo de código después de explicar otros elementos.

RECUPERAR LA OPCIÓN SELECCIONADA EN UN SELECT

Cuando tenemos una lista desplegable select con diferentes option, en general nos interesará saber cuál de los elementos option es el que se encuentra seleccionado y el contenido de su atributo value. Para ello hemos de tener en cuenta que:

a) Cada elemento HTML de tipo select tiene una propiedad denominada options, de tipo array, que contiene los elementos options con índice 0, 1, 2, ... n. Por ejemplo aquí vemos cómo extraer los elementos select, sus atributos value y el texto que contienen (mediante la propiedad text).

```
var elementosSelect = document.getElementsByTagName('select');
alert('Contenido de value es: '+ elementosSelect[0].value;
alert('Contenido texto es: '+ elementosSelect[0].text;
```

La propiedad selectedIndex del elemento select nos devuelve el índice numérico de la opción seleccionada (la primera opción tiene índice 0, la segunda 1, y así sucesivamente).

Aquí vemos un ejemplo de cómo se puede utilizar selectedIndex:

```
var elementoCiudad = document.getElementById('ciudad');
var indiceSeleccionado = elementoCiudad.selectedIndex;
alert (elementoCiudad.options[indiceSeleccionado].text);
```

También podemos escribir elementoCiudad.options[elementoCiudad.selectedIndex].text prescindiendo del uso de la variable intermedia indiceSeleccionado.

El evento para detectar cambios en los select normalmente será “onchange”.

EJERCICIO Y EJEMPLO

El código que vamos a ver a continuación nos va a servir al mismo tiempo como ejemplo de las cuestiones que hemos explicado anteriormente y como ejercicio.

Para probar el funcionamiento de este código debes poner dos imágenes de 100 por 100 píxeles con el nombre caraHombre.jpg y caraMujer.jpg, en el mismo directorio donde tengas el código HTML. Nosotros vamos a usar estas dos imágenes (usa estas u otras según prefieras):

Escribe el siguiente código, guárdalo como archivo con extensión HTML, visualízalo en tu navegador y comprueba su funcionamiento. No prestes atención al código CSS, ya que lo que nos interesa es centrarnos en JavaScript. Estudia el código y responde a las preguntas que se plantean más abajo. Este código usa conocimientos variados que hemos ido adquiriendo a lo largo del curso, por lo que nos viene bien como repaso. Si has venido siguiendo el curso, deberías ser capaz de entender todo el código.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
 <style type="text/css"> body {margin-left:30px; font-family: sans-serif;} h4 {margin:0;} div {float:left;}
.estiloForm, #encuadraImagen, #datos {background-color: #f3f3f3; border: solid 2px black; margin-left:10px; width:300px; }
  .estiloForm{ width: 330px; padding:10px;} #datos {margin-top: 20px; height:226px; padding:10px;}
#encuadraImagen {width:100px; height:100px;   text-align:center;}
 .estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
 br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
 </style>
<script type="text/javascript">
window.onload = function () {
            /* Variables globales (por estar declaradas sin var) */
            casillaDatos = document.getElementById('datos'); //Nodo donde vamos a mostrar los datos
            radioButTrat = document.getElementsByName("tratamiento"); //Nodos radio buttons
            checkboxElements = new Array();

            var elementosDelForm = document.getElementsByTagName('input'); //Elementos input
            var elementosSelect = document.getElementsByTagName('select');
            for(var i=0; i<elementosDelForm.length;i++) {
            if (elementosDelForm[i].type == 'radio') {elementosDelForm[i].addEventListener("click", actualizarDatos);}
            else {elementosDelForm[i].addEventListener("change", actualizarDatos);}
            if (elementosDelForm[i].type == 'checkbox') {checkboxElements.push(elementosDelForm[i]);}
            }
            for (var i=0; i<elementosSelect.length;i++) {elementosSelect[i].addEventListener("change", actualizarDatos);}
}
```

```
function actualizarDatos() {
var rutaImagen = '';
var radioButSelValue = '';
for (var i=0; i<radioButTrat.length; i++) {if (radioButTrat[i].checked == true) { radioButSelValue= radioButTrat[i].value;} }
if (radioButSelValue != ''){
            if (radioButSelValue =='Sr.') {rutaImagen='caraHombre.jpg';} else {rutaImagen='caraMujer.jpg';}
            document.getElementById('encuadraImagen').innerHTML = '';
            document.getElementById('encuadraImagen').style.background='url("'+rutaImagen+'") no-repeat';
}
var checkBoxSel = new Array();
for (var i=0; i<checkboxElements.length;i++) {
if (checkboxElements[i].checked ==true) {checkBoxSel.push(checkboxElements[i].name);}
}

var elementoCiudad = document.getElementById('ciudad');
casillaDatos.innerHTML='<h4> Datos introducidos: </h4><p>Tratamiento: '+(radioButSelValue||' --- ')+'</p>'+
'<p>Nombre: '+document.getElementById('nombre').value+'</p>'+
'<p>Apellidos: ' + (document.getElementById('apellidos').value||' --- ')+'</p>'+
'<p>Dirección: ' + (document.getElementById('direccion1').value||' --- ')+'</p>'+
'<p>Ciudad: ' + (elementoCiudad.options[elementoCiudad.selectedIndex].text||' --- ')+'</p>'+
'<p>Preferencias: ' + (checkBoxSel||' --- ')+'</p>';
}
</script></head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
   <div class="estiloForm">
      <form name ="formularioContacto" method="get" action="#">
      <label>Tratamiento</label>
      <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
      <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
      <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
      <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
      <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
      <label>Ciudad</label><select id="ciudad" name="ciudad">
         <option value="">Elija opción</option>
         <option value="Mexico">México D.F. (MX)</option>
         <option value="Madrid">Madrid (ES)</option>
         <option value="Santiago">Santiago (CL)</option>
      </select><br/>
      <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
     <input name="Peliculas" type="checkbox"  />Películas
     <input type="submit" value="Enviar"/> <input type="reset" value="Cancelar"/>
    </form>
   </div>
   <div style="float:left;">
   <div id="encuadraImagen"><h1> ? </h1></div>            <br/>
   <div id="datos" style="float:left;"><h4> Datos introducidos: </h4></div>
   </div>
</body></html>
```

El resultado esperado es que cuando se escriba o seleccionen opciones en el formulario, se muestren las imágenes y texto correspondientes en el lado derecho (en la imagen vemos un ejemplo).

Responde estas preguntas (mantén la consola activada para ver posibles mensajes de error):

a) ¿Qué código es el que hace posible que cuando se selecciona el radio button Sr aparezca la imagen caraHombre.jpg y que cuando se selecciona el radio button Sra aparezca la imagen caraMujer.jpg?

b) ¿ radioButTrat es una variable global o local a una función? ¿Por qué?

c) ¿radioButTrat es una array o un objeto NodeList de tipo array-like? ¿El método length es aplicable a objetos de tipo array? ¿El método length es aplicable a objetos de tipo array-like?

d) ¿Qué ocurre si tratas de aplicar el método push a radioButTrat? Escribe el código para ello. ¿Qué mensaje visualizas en la consola? Busca información y trata de explicar por qué ocurre esto.

e) ¿Las imágenes son el fondo de un elemento HTML distinto de img o es el atributo source de un elemento img?

f) ¿checkBoxSel es un array o un NodeList de tipo array-like?

g) ¿Qué significa y para qué sirve este código: checkBoxSel||' --- ' ?

h) ¿Y este otro: elementoCiudad.options[elementoCiudad.selectedIndex].text ?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01180E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ARRAY.PROTOTYPE JAVASCRIPT. DIFERENCIA ENTRE ARRAY ARRAY-LIKE, NODELIST. TYPEERROR: NOT A FUNCTION NO METHOD 'FOREACH' (CU01180E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº80 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

ARRAY.PROTOTYPE, NODELIST, ARRAYS Y ARRAY-LIKE OBJECTS

En apartados anteriores del curso estudiamos que la herencia en JavaScript es una herencia basada en prototipos. En esta entrega vamos a repasar algunos conceptos sobre herencia y acceso a los objetos en la raíz de la cadena de herencia, y a estudiar algunos conceptos sobre qué son NodeList, Arrays y array-like objects.

Vamos a llamar la atención sobre el hecho de que los métodos de tipo get que actúan sobre el DOM devuelven una colección de objetos a la que se denomina NodeList o lista de nodos, que se dice son objetos array-like (similares a un array, pero no exactamente arrays). El hecho de que un NodeList no es un array se comprueba con este código. Ejecútalo y comprueba los resultados (activa la consola para ver los mensajes de error):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
  <style type="text/css"> body {margin-left:30px; font-family: sans-serif;}
          .estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
  .estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
           br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
  </style>
<script type="text/javascript">
window.onload = function () {
          var elementosInput = document.getElementsByTagName('input'); //Elementos input
          var elementosSelect = document.getElementsByTagName('select');
          for (var i=0; i< elementosSelect.length; i++) {elementosInput.push(elementosSelect[i])};
          alert ('Tenemos un número de elementos input ó select en el formulario: '+ elementosInput.length);
}
</script></head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
  <div class="estiloForm"><form name ="formularioContacto" method="get" action="#">
    <label>Tratamiento</label>
    <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
                    <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
                    <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
    <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
    <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
    <label>Ciudad</label><select id="ciudad" name="ciudad">
                                <option value="">Elija opción</option>
      <option value="Mexico">México D.F. (MX)</option>
      <option value="Madrid">Madrid (ES)</option>
      <option value="Santiago">Santiago (CL)</option>
    </select><br/>
                    <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
                    <input name="Peliculas" type="checkbox"  />Películas
                    <input type="submit" value="Enviar"/> <input type="reset" value="Cancelar"/>
  </form></div>
</body></html>
```

El resultado es un error similar a este: <<TypeError: elementosInput.push is not a function>>. El motivo de esto es que un objeto array-like no dispone de todos los métodos propios de los arrays. Sin embargo, los nodeList sí tienen algunas características comunes con el array, como disponer de una propiedad length que informa del número de elementos que contiene la colección de elementos.

El interés de los NodeList está en que son entidades dinámicas: si durante la ejecución de código se elimina un elemento que pertenecía a un NodeList, el NodeList queda actualizado automáticamente porque está referenciando dinámicamente. En cambio si tuviéramos los nodos en un array y se elimina un nodo, el array seguiría tal y como estaba porque el array no está ligado dinámicamente a lo que ocurra en el DOM en tiempo de ejecución.

En ocasiones nos puede interesar convertir un objeto array-like en un array verdadero. Esto se puede hacer de varias maneras:

a) Manualmente introduciendo cada elemento en un nuevo objeto array. Ejemplo modificando el código anterior (ejecútalo y comprueba los resultados):

```
<script type="text/javascript">
window.onload = function () {
	var elementosInput = document.getElementsByTagName('input'); //Elementos input
	var elementosSelect = document.getElementsByTagName('select');
	var arrayNodos = [];
	addNodeListToArray (elementosInput, arrayNodos);
	addNodeListToArray (elementosSelect, arrayNodos);
	alert ('Tenemos un número de elementos input ó select en el formulario: '+ arrayNodos.length);
}

function addNodeListToArray (elNodeList, elArray) {
	for (var i=0; i< elNodeList.length; i++) {
	var elementoActual = elNodeList[i];
	elArray.push(elementoActual);
	}
	alert ('Añadidos ' + elNodeList.length +' elementos');
}
</script>
```

El resultado esperado es que se muestre: Añadidos 9 elementos, Añadidos 1 elementos, Tenemos un número de elementos input ó select en el formulario: 10

b) Accediendo al objeto Array para aplicar un método que nos permite obtener un array a partir de un objeto array-like. La sintaxis sería cualquiera de estas (ten en cuenta que en algunos casos, especialmente con navegadores antiguos, estas sintaxis pueden no ser reconocidas):

```
var divsEnArray = Array.prototype.slice.call(document.querySelectorAll('div'));
```

```
var divs = [].slice.call(document.querySelectorAll('div'));
```

Esta sintaxis merece ser comentada. La invocación Array.prototype nos permite acceder al objeto en la cima de la jerarquía de herencia u objeto "padre" de todos los objetos de su tipo. Cuando escribimos algo como miArray = new Array(); estamos creando una instancia de un objeto tipo Array, que hereda todas sus características del "objeto padre" Array.prototype.

Normalmente invocamos métodos sobre instancias de los objetos en la cima jerárquica, pero en ocasiones (como en este ejemplo), puede que nos resulte de interés invocar directamente un método sobre el objeto padre o prototype. En este caso al invocar el método slice sin argumentos, se nos devuelve un array sin más. Para indicarle a partir de qué colección se debe devolver el array, usamos el método call indicando que el objeto que actuará como this y a partir del cual se debe devolver el array es un objeto NodeList (un array-like object).

La sintaxis [].slice.call(document.querySelectorAll('div')); es similar, pero en lugar de operar sobre el prototype utilizamos un objeto array anónimo representado por []. Aunque esta sintaxis es posible, preferimos la otra basada en el uso del objeto prototype porque en general será más segura y eficiente.

Aplicado al ejemplo de código que venimos viendo podríamos utilizar este código. Escríbelo en tu editor y comprueba los resultados:

```
window.onload = function () {
	var elementosInput = document.getElementsByTagName('input'); //Elementos input
	var elementosSelect = document.getElementsByTagName('select');
	var arrayNodos = Array.prototype.slice.call(elementosInput);
	arrayNodos.push([].slice.call(elementosSelect));
	alert ('Tenemos un número de elementos input ó select en el formulario: '+ arrayNodos.length);
}
```

El resultado esperado es que se muestre por pantalla: <<Tenemos un número de elementos input ó select en el formulario: 10>>

Hemos usado en un caso Array.prototype y en otro [] para poner un ejemplo de uso, aunque en realidad preferiremos usar Array.prototype:

```
window.onload = function () {
	var elementosInput = document.getElementsByTagName('input'); //Elementos input
	var elementosSelect = document.getElementsByTagName('select');
	var arrayNodos = Array.prototype.slice.call(elementosInput);
	arrayNodos.push(Array.prototype.slice.call(elementosSelect));
	alert ('Tenemos un número de elementos input ó select en el formulario: '+ arrayNodos.length);
}
```

ARRAYS.FROM

Aún nos queda una alternativa para resolver la conversión de objetos tipo array-like en arrays. El método Array.from. Este método crea un objeto de tipo array a partir de un objeto iterable u objeto de tipo array-like. Su uso habitual será con sintaxis similar a esta:

```
var divs = document.querySelectorAll('div');
var arrayDivs = Array.from(divs);
```

Este método parece el más sencillo, sin embargo puede tener importantes problemas y no ser soportado por diferentes navegadores donde puede generar mensajes de error (tipo Uncaught TypeError: undefined is not a function). Por ello recomendamos usar alguno de los métodos vistos anteriormente, que nos evitan problemas de compatibilidad entre navegadores.

RESUMEN

Vamos a tratar de hacer un resumen que aclare ideas. Los aspectos clave que hemos visto son:

a) Existen algunos objetos que son similares a los arrays pero que no son exactamente arrays. Estos objetos se suelen denominar objetos tipo array-like. Entre estos objetos tenemos los NodeList o listas de nodos devueltos por los métodos populares de acceso al DOM como document.getElementById y similares.

b) Los objetos array-like tienen algunas propiedades y métodos de los arrays, pero no todos ellos (por ejemplo pueden carecer de los métodos push, no pueden ser recorridos con foreach, etc.). Si tratamos de aplicar ciertos métodos de los arrays a objetos array-like podemos obtener errores del tipo: Object #<NodeList> has no method. No podemos usar un objeto array-like como si fuera un array, pero en circunstancias concretas sí podemos transformar los objetos array-like en arrays verdaderos para realizar ciertas manipulaciones u operaciones.

c) Los NodeList son objetos de tipo array-like que contienen lo que se denomina live Nodes: colecciones de elementos que automáticamente se actualizan si se modifica el DOM. Podemos decir que esta ventaja de actualización automática es importante, y que por ello en algunas ocasiones nos interesa trabajar con NodeList en lugar de con arrays, a pesar de que por otro lado tengan la desventaja de tener menos funcionalidad.

EJERCICIO

Examina el siguiente código y responde a las cuestiones que se muestran a continuación:

```
var arr = [];
var divs = document.querySelectorAll('div');
for(var i = divs.length; i--; arr.unshift(divs[i]));
```

a) ¿Qué cometido cumple este código?

b) Aplicando la idea que podemos extraer de este código, modifica el código que hemos visto como ejemplo en esta entrega para crear un array con los elementos input y select del formulario y mostrar por pantalla el número de elementos input y select existentes en el formulario.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01181E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

HTMLELEMENT JAVASCRIPT. MÉTODOS FOCUS, BLUR. SUBMIT FORMULARIOS: DETENER ENVÍO. ONSUBMIT RETURN. EJEMPLOS. (CU01181E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº81 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

SIGNIFICADOS DE FOCUS Y BLUR

Hemos de distinguir varios posibles usos de focus y blur. Como sabemos, focus hace alusión a que un elemento HTML toma el foco en la página web (bien porque el usuario haga click sobre él, bien porque al pulsar la tecla de tabulación el foco llegue hasta él). Tomar el foco indica que se puede escribir sobre el elemento (si es un campo de texto de un formulario) o que el elemento está seleccionado (si es otro tipo de elemento como una imagen).

Vamos a llamar la atención sobre el hecho de que focus y blur pueden tener distintos significados o usos:

a) Eventos que podemos capturar para disparar un código de respuesta (esto ya lo hemos estudiado en apartados anteriores del curso).

b) Un método aplicable a objetos de tipo Window (esto ya lo hemos estudiado en apartados anteriores del curso).

c) Un método aplicable a objetos de tipo HTMLElement (nodos del DOM) para hacerles obtener o perder el foco, siempre que dicho objeto sea susceptible de tener el foco. Hay elementos como un input de un formulario que pueden tener el foco y otros como un span que no pueden tener el foco.

La sintaxis para establecer el foco en un elemento que admite el foco es:

```
nombreDelElemento.focus();
```

La sintaxis para hacer que un elemento que tenía el foco lo pierda es:

```
nombreDelElemento.blur();
```

Escribe el siguiente código y comprueba sus resultados. Comprueba cómo en él se hace uso por un lado de los eventos focus y blur, y por otro lado del método focus para establecer el foco cuando carga la página en el primer elemento input de tipo text que existe en el formulario.

Fíjate también en cómo se manejan los eventos blur y focus para hacer que cuando un input de tipo text recibe el foco, su color de fondo cambie a amarillo, mientras que cuando pierde el foco, su color de fondo cambia a blanco.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
   <style type="text/css"> body {margin-left:30px; font-family: sans-serif;}
            .estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
   .estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
             br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
   </style>
<script type="text/javascript">
window.onload = function () {
            var elegido = false;
            var elementosInput = document.getElementsByTagName('input'); //Elementos input
            for (var i=0; i< elementosInput.length; i++) {
                        if (elementosInput[i].type == 'text' && elegido==false) {var elegidoParaFoco = elementosInput[i];
elegido=true;}
                        if (elementosInput[i].type == 'text') {
                        elementosInput[i].addEventListener('blur', ponerFondoBlanco);
                        elementosInput[i].addEventListener('focus', ponerFondoAmarillo);
                        }
            }
            elegidoParaFoco.focus(); elegidoParaFoco.style.backgroundColor='yellow';
}

function ponerFondoBlanco() {this.style.backgroundColor = 'white';}
function ponerFondoAmarillo() {this.style.backgroundColor = 'yellow';}
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
   <div class="estiloForm">
     <form name ="formularioContacto" method="get" action="#">
      <label>Tratamiento</label>
      <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
                        <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
                        <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
      <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
      <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
      <label>Ciudad</label><select id="ciudad" name="ciudad">
                                    <option value="">Elija opción</option>
         <option value="Mexico">México D.F. (MX)</option>
         <option value="Madrid">Madrid (ES)</option>
         <option value="Santiago">Santiago (CL)</option>
      </select><br/>
                        <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
                        <input name="Peliculas" type="checkbox"  />Películas
                        <input type="submit" value="Enviar"/> <input type="reset" value="Cancelar"/>
   </form>
   </div>
</body></html>
```

Cuando en una página web el formulario es el elemento principal, se considera una buena práctica (para facilitar que el usuario introduzca los datos) establecer el foco sobre el primer elemento del formulario una vez la página haya cargado.

SIGNIFICADOS DE SUBMIT

Al igual que focus y blur pueden ser eventos o métodos de objetos, submit puede ser un evento pero también un método de un formulario. Como sabemos, submit hace alusión al envío de un formulario.

Podemos utilizar el evento submit para retrasar el envío del formulario hasta después de haber procesado el evento (de esta forma podemos hacer operaciones previas al envío, por ejemplo una validación). Una vez procesado el evento, procederíamos al envío del formulario. Para que el formulario se envíe o no según los resultados de las operaciones previas al envío añadiremos el parámetro para capturar el evento en la función manejadora y aplicaremos el método preventDefault() que hemos estudiado en anteriores entregas del curso.

Este sería un ejemplo de sintaxis:

```
document.forms['nombreFormulario'].addEventListener('submit', funcionManejadora);
...
...
    function funcionManejadora(evObject) {
    evObject.preventDefault(); //Anulamos la acción de defecto
    if ( ... ) { //Tareas a realizar si se cumple la condición
    document.forms['formularioContacto'].submit(); } //Se envía el formulario
    else { //Tareas a realizar si no se cumple la condición
    ... } //No se envía el formulario
    }
```

Escribe este código y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
   <style type="text/css"> body {margin-left:30px; font-family: sans-serif;}
            .estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
   .estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
            br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
   </style>
<script type="text/javascript">
window.onload = function () {
document.forms['formularioContacto'].addEventListener('submit', avisarUsuario);
}

function avisarUsuario(evObject) {
evObject.preventDefault();
var nuevoNodo = document.createElement('h2');
nuevoNodo.innerHTML = '<h2 style="color:orange;">Enviando el formulario...</h2>';
document.body.appendChild(nuevoNodo);
var retrasar = setTimeout(procesaDentroDe2Segundos, 1000);
}
```

```
function procesaDentroDe2Segundos() {
document.forms['formularioContacto'].submit();
}

</script>
</head>

<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
  <div class="estiloForm">
    <form name ="formularioContacto" method="get" action="http://aprenderaprogramar.com">
     <label>Tratamiento</label>
     <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
                    <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
                    <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
     <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
     <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
     <label>Ciudad</label><select id="ciudad" name="ciudad">
                              <option value="">Elija opción</option>
       <option value="Mexico">México D.F. (MX)</option>
       <option value="Madrid">Madrid (ES)</option>
       <option value="Santiago">Santiago (CL)</option>
     </select><br/>
                    <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
                    <input name="Peliculas" type="checkbox"  />Películas
                    <input type="submit" value="Enviar"/> <input type="reset" value="Cancelar"/>
  </form>
  </div>
</body>
</html>
```

El resultado esperado es que cuando se pulsa el botón enviar del formulario, debajo de este aparezca el mensaje "Enviando el formulario…". Para hacer aparecer este mensaje hemos añadido un nodo al DOM y establecido su contenido con innerHTML.

Nota: para que el efecto sea visible hemos introducido un pequeño retraso forzado en el envío del formulario usando un setTimeout. Hemos hecho esto para que nos dé tiempo a ver el texto (si la conexión a internet que usemos es muy rápida y no establecemos una pequeña demora no nos daría tiempo a ver el mensaje, ya que este desaparece en cuanto comienza a cargarse la página de destino del formulario). Este retraso no tiene una utilidad "práctica", únicamente lo hemos introducido para poder ver cómo podemos dar lugar a la ejecución del evento submit usando el método submit() en el momento en que nosotros deseemos.

EJERCICIO Y EJEMPLO

Otra forma de detener el envío de un formulario se basa en que si el código de respuesta a un evento submit es el valor booleano false, no se producirá el envío del formulario. En otro caso, sí se producirá el envío. Examina el siguiente código y responde a las cuestiones que se muestran a continuación:

```
<!DOCTYPE html>
<html><head><script>
function validarForm() {
   var x = document.forms["elForm1"]["nombre"].value;
   if (x==null || x=="") { alert("El formulario no puede enviarse sin rellenar el nombre");
      return false; }
}
</script></head>
<body><form style ="margin:30px;" name="elForm1" action="http://aprenderaprogramar.com" onsubmit="return
validarForm()" method="get">
Nombre: <input type="text" name="nombre">
<input type="submit" value="Enviar">
</form>
</body></html>
```

a) ¿Qué cometido cumple este código?

b) ¿Qué ocurre si en lugar de onsubmit="return validarForm()" escribes onsubmit="validarForm()"? ¿Por qué?

c) Si x no es null ni es vacío, ¿qué devuelve la función validarForm()? ¿Qué implicaciones tiene esto en relación al envío del formulario?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01182E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT VALIDAR CAMPO TEXTO NO VACÍO. QUE VALOR SEA NUMÉRICO. EMAIL O CORREO ELECTRÓNICO EN FORMULARIOS. EJEMPLOS (CU01182E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº82 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

VALIDACIÓN DE FORMULARIOS CON JAVASCRIPT

Un uso habitual e importante de JavaScript es servir para realizar la validación de formularios. Cuando un usuario rellena un formulario y pulsa en el botón enviar, comprobaremos con JavaScript si ha introducido los datos mínimos requeridos y con el formato requerido. Si no es así, abortaremos el envío del formulario y mostraremos los avisos correspondientes evitando enviar la información al servidor.

El envío de la información al servidor supondría una demora en la respuesta (enviar la información, realizar la validación en el servidor, enviar la respuesta…) y daría la impresión de que la página web "está lenta". Con JavaScript podemos lograr una respuesta inmediata, haciendo que la navegación para el usuario de nuestra página web sea satisfactoria.

La validación normalmente consiste en detectar el evento submit del formulario y derivar la gestión del mismo a una función manejadora que se encarga de revisar el contenido del formulario. Si en esa revisión se detectan errores en los formatos o falta de datos, el envío se aborta y se muestra un mensaje de aviso al usuario. Si en la revisión se considera que está todo conforme, se procede al envío del formulario.

La validación no supone más que una aplicación práctica de los conocimientos que hemos adquirido durante el curso (acceso a elementos del DOM, manejo de objetos, expresiones regulares, etc.).

La invocación a la validación se puede enfocar de distintas maneras. La forma quizás preferible es usar addEventListener y preventDefault:

```
document.nombreDelFormulario.addEventListener('submit', validarFormulario);

function validarFormulario(evObject) {
evObject.preventDefault(); //Evita el envío del formulario hasta comprobar
}
```

Otra forma que ha venido siendo bastante utilizada es capturar el evento en el código HTML para que devuelva true (en cuyo caso se enviará el formulario) o false (en cuyo caso no se enviará):

```
function validarFormulario() {
if (… ) { return true; // Si se verifican las condiciones enviar formulario
} else { return false; // Si no se verifican las condiciones no se enviará el formulario
}
---
}
---
<form action="" method="" id="" name="" onsubmit="return validarFormulario ()">
```

VALIDAR QUE UN CAMPO NO ESTÁ VACÍO

En muchos formularios se requerirá que uno o varios campos no se encuentren vacíos. Por ejemplo si se trata de una reserva de hotel, será necesario que datos como el nombre de la persona que realiza la reserva no se encuentren vacíos.

También podemos comprobar que no se hayan introducido sólo espacios en blanco (que son caracteres pero no aportan información), u otro tipo de comprobaciones más elaboradas usando expresiones regulares.

Escribe este código que es un ejemplo de comprobación de que un campo no esté vacío o contenga sólo espacios en blanco y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css">body {background-color:yellow; font-family: sans-serif;} label{color: maroon; display:block;
padding:5px;}
</style>
<script type="text/javascript">
window.onload = function () {
document.formularioContacto.nombre.focus();
document.formularioContacto.addEventListener('submit', validarFormulario);
}

function validarFormulario(evObject) {
evObject.preventDefault();
var todoCorrecto = true;
var formulario = document.formularioContacto;
for (var i=0; i<formulario.length; i++) {
        if(formulario[i].type =='text') {
                if (formulario[i].value == null || formulario[i].value.length == 0 || /^\s*$/.test(formulario[i].value)){
                alert (formulario[i].name+ ' no puede estar vacío o contener sólo espacios en blanco');
                todoCorrecto=false;
                }
        }
        }
if (todoCorrecto ==true) {formulario.submit();}
}

</script></head>
<body><div id="cabecera"><h1>Portal web aprenderaprogramar.com</h1><h2>Didáctica y divulgación de la
programación</h2>
</div>
<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="http://aprenderaprogramar.com">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<label><input type="submit" value="Enviar"><input type="reset" value="Cancelar"></label>
</form></body></html>
```

Recordar lo que explicamos en los apartados del curso dedicados a expresiones regulares. /^\s*$/ define una expresión regular donde \s es el espacio en blanco, ^ indica comienzo de cadena con ese carácter, * indica repetición cualquier número de veces y $ señala el carácter de terminación de cadena. Por tanto la expresión regular indica “las cadenas que empiezan con espacio en blanco, siguen con cualquier número de espacios en blanco y terminan con un espacio en blanco”.

El método test devuelve true si la cadena hace match con la expresión regular o false si la cadena no hace match.

VALIDAR QUE UN CAMPO CONTIENE UN VALOR NUMÉRICO

En ocasiones se puede requerir que un campo contenga un valor numérico. Las comprobaciones se pueden hacer basándonos en expresiones regulares, basándonos en la función isNaN, o basándonos en que el valor numérico debe ser mayor, menor, mayor o igual, menor o igual que otro valor numérico de referencia.

Recordar que isNaN(x) devuelve true si x es un valor no asimilable a un número o false si el valor es asimilable a un número. La función isNaN es “laxa” en el sentido de que tratará de asimilar a un número todo lo que sea posible. Por ejemplo alert(isNaN(-3.23)); devuelve false (se considera -3.23 un número). Esto puede ser interesante en algunas ocasiones pero no interesante en otras.

Ejemplo: pedimos la altura respecto al nivel del mar, siendo 0 el nivel del mar, un valor positivo una altura sobre el nivel del mar y un valor negativo una altura por debajo del nivel del mar (por ejemplo la altura de un submario). En este caso nos interesa usar isNaN.

Otro ejemplo: pedimos la edad de una persona. En este caso será preferible usar expresiones regulares, por ejemplo comprobar que el valor del campo se corresponda con uno ó dos dígitos, lo que haría válido 5, 44, 88, pero haría inválidos 148, -3.23 ó 5.25.

Escribe este código que es un ejemplo de validaciones con campos numéricos y comprueba sus resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">
body {background-color:yellow; font-family: sans-serif;}
label{color: maroon; display:block; padding:5px;}
</style>
<script type="text/javascript">

window.onload = function () {
document.formularioContacto.nombre.focus();
document.formularioContacto.addEventListener('submit', validarFormulario);
}
```

```
function validarFormulario(evObject) {
evObject.preventDefault();
var todoCorrecto = true;
var formulario = document.formularioContacto;
if (isNaN(formulario.edad.value)==true || /^[1-9]\d$/.test(formulario.edad.value)==false ) {alert ('Edad no valida');
todoCorrecto=false;}
if (isNaN(formulario.altura.value)==true || formulario.altura.value<=0 || formulario.altura.value>=2.50) {
alert ('Altura no valida'); todoCorrecto=false;}
if (todoCorrecto ==true) {formulario.submit();}
}
</script></head>
<body><div id="cabecera"><h1>Portal web aprenderaprogramar.com</h1><h2>Didáctica y divulgación de la
programación</h2></div>
<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="http://aprenderaprogramar.com">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Edad:</span> <input id="edad" type="text" name="edad" /></label>
<label for="email"><span>Altura:</span> <input id="altura" type="text" name="altura" /></label>
<label><input type="submit" value="Enviar"><input type="reset" value="Cancelar"></label>
</form></body></html>
```

El resultado esperado es que el formulario no se envíe si la edad no está formada por dos dígitos (no sería válido 124 ni -3.23) o si la altura es un valor negativo, mayor o igual a 2.50 ó no tiene formato numérico.

VALIDAR UNA DIRECCIÓN DE EMAIL

Para comprobar que el usuario introduce una dirección de email válida podemos hacer distintos tipos de comprobaciones. La más básica es comprobar que la cadena contiene cualquier número de caracteres seguido del carácter @ segudio de cualquier número de caracteres, seguido de un punto, y seguido de cualquier número de caracteres. Esta validación es amplia y no aceptaría la mayor parte de direcciones no válidas, por ejemplo no se aceptaría <<andres@gmailcom>> porque le falta el punto.

En general será suficiente usar una expresión regular simple del tipo:

```
/\S+@\S+\.\S+/
```

En muchas páginas de internet se encuentran expresiones más complejas que tratan de tener en cuenta posibilidades menos frecuentes de formato de correo electrónico, por ejemplo esta:

```
/ [a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-
9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/
```

Las comprobaciones se harían de la misma forma a como hemos visto en los ejemplos anteriores.

EJERCICIO

Dado el siguiente código HTML que contiene un formulario con tres campos (nombre, apellidos y email):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<style type="text/css">body {background-color:yellow; font-family: sans-serif;}
label{color: maroon; display:block; padding:5px;}
</style>
</head>
<body>
<div id="cabecera"><h1>Portal web aprenderaprogramar.com</h1><h2>Didáctica y divulgación de la programación</h2>
</div>
<!-- Formulario de contacto -->
<form name ="formularioContacto" class="formularioTipo1" method="get" action="http://aprenderaprogramar.com">
<p>Si quieres contactar con nosotros envíanos este formulario relleno:</p>
<label for="nombre"><span>Nombre:</span> <input id="nombre" type="text" name="nombre" /></label>
<label for="apellidos"><span>Apellidos:</span> <input id="apellidos" type="text" name="apellidos" /></label>
<label for="email"><span>Correo electrónico:</span> <input id="email" type="text" name="email" /></label>
<label><input type="submit" value="Enviar"><input type="reset" value="Cancelar"></label>
</form>
</body>
</html>
```

Crear el código JavaScript que cumpla con las siguientes funciones:

a) Si la longitud (número de caracteres) del campo nombre es mayor de 15 o igual a cero, el formulario no se enviará.

b) Si la longitud (número de caracteres) del campo apellidos es mayor de 30 o igual a cero, el formulario no se enviará.

c) Si la longitud (número de caracteres) del campo email es mayor de 35 o igual a cero, el formulario no se enviará. Si el email no contiene el carácter @ el formulario no se enviará.

d) Si se produce cualquiera de las circunstancias anteriores, debe aparecer un recuadro con color de fondo naranja y texto negro a la derecha de la casilla de introducción de datos, informando del problema detectado en ese campo (si es que ese campo presenta algún problema). Nota: estos mensajes se deben mostrar sólo si el campo es erróneo después de pulsado el botón enviar, y deben desaparecer si el usuario realiza un nuevo intento y el campo es correcto. Los mensajes se incorporarán al DOM (no serán mensajes usando alert).

Ejemplo de ejecución. El usuario deja el nombre, apellidos y correo electrónico vacíos. A la derecha de las casillas de introducción de datos aparecerá: El nombre no puede estar vacío. Los apellidos no pueden estar vacíos. El correo electrónico no puede estar vacío.

Ahora el usuario introduce nombre: <<Juan Manuel de Todos los Santos Efímeros Ecuánimes de Todos los días de la Tercera Edad>>. Como apellidos introduce: <<Suárez>>. Y como correo electrónico introduce: <<juanmanueldetodoslosantosefimerosecuanimesdetodoslosdias@gmail.com>>. Pulsa enviar. A la derecha de la casilla de introducción del nombre debe aparecer: "Nombre demasiado largo. Máximo 15 caracteres". A la derecha de apellidos no aparecerá nada porque el apellido es correcto. A la derecha del correo electrónico debe aparecer: "Correo electrónico demasiado largo. Máximo 35 caracteres". Ahora el usuario escribe como nombre: Juan Manuel. Como apellidos: Suárez. Y como correo electrónico juanmanuel@gmail.com. Pulsa enviar y el formulario es enviado.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01183E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

DISABLED JAVASCRIPT EN INPUT TEXT, SUBMIT, BUTTON, SELECT, LINK, STYLE... IMPEDIR ENVÍO DUPLICADO FORM. DESACTIVAR CSS (CU01183E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº83 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

DISABLED JAVASCRIPT

Diferentes objetos de tipo HTMLElement tienen disponible la propiedad disabled, que permite que estén inactivos o deshabilitados de modo que el usuario no podrá pulsar sobre ellos o elegirlos. Esto puede tener distintas utilidades: impedir el envío duplicado de un formulario por error, desactivar temporalmente algunas opciones, etc.

DESACTIVAR EL BOTÓN DE ENVÍO DE UN FORMULARIO. IMPEDIR ENVÍO DUPLICADO

Cuando un usuario pulsa el botón enviar de un formulario deberíamos esperar que no ejecute ninguna acción adicional hasta que el formulario sea enviado. No obstante, podría ocurrir que por error haga dos veces click, o que la conexión con el servidor se demore y tenga la impresión de que el formulario no se está enviando y vuelva a hacer click sobre el botón de envío. Estas situaciones pueden dar lugar a el envío duplicado de un formulario, situación no deseable en cuanto que duplica procesos, consume recursos y puede dar lugar a problemas como errores en procesos de compra o contabilización de solicitudes, etc.

Para evitar un posible click en la fracción de tiempo que transcurre entre el click en el botón de envío y la carga de la página de destino podemos usar la propiedad disabled (deshabilitado) aplicable a los objetos HTMLElement de tipo input y a objetos HTML de tipo button entre otros.

La sintaxis a utilizar es la siguiente:

```
nombreDelObjetoADeshabilitar.disabled = true;
```

Lo normal será deshabilitar el botón de envío en cuanto detectemos el evento click sobre el botón de envío ó submit sobre un formulario. Escribe este código y comprueba cómo se desactivan los botones cuando se pulsa el botón enviar (en este caso lo detectamos usando el evento submit):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
 <style type="text/css"> body {margin-left:30px; font-family: sans-serif;}
.estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
 .estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
 </style>

<script type="text/javascript">
window.onload = function () {
document.forms['formularioContacto'].addEventListener('submit', avisarUsuario);
}
```

```
function avisarUsuario(evObject) {
evObject.preventDefault();
var botones = document.querySelectorAll('.botonFormulario');
for (var i=0; i<botones.length; i++) {botones[i].disabled = true; }
var nuevoNodo = document.createElement('h2');
nuevoNodo.innerHTML = '<h2 style="color:orange;">Enviando... espere por favor</h2>';
document.body.appendChild(nuevoNodo);
var retrasar = setTimeout(procesaDentroDe2Segundos, 1000);
}

function procesaDentroDe2Segundos() {document.forms['formularioContacto'].submit();}
</script>
</head>
<body>
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplos JavaScript</h3>
  <div class="estiloForm">
    <form name ="formularioContacto" method="get" action="http://aprenderaprogramar.com">
     <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
     <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
     <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/>
     <input class="botonFormulario" type="submit" value="Enviar"/>
     <input class="botonFormulario" type="reset" value="Cancelar"/>
  </form>
  </div>
</body></html>
```

En el código anterior hemos activado un retardo utilizando setTimeout simplemente para poder comprobar cómo se desactivan los botones hasta que el formulario es enviado (es decir, el setTimeout no sirve realmente para nada, lo hemos incluido para poder ver el efecto de desactivar botones).

DESACTIVAR INPUTS, SELECTS O CHECKBOX

En ocasiones nos puede interesar desactivar elementos de un formulario hasta que el usuario no elija alguna opción previa requerida.

Podemos desactivar inputs de tipo text, inputs de tipo checkbox, o elementos de tipo select. Para ello debemos aplicar la sintaxis ya conocida:

```
nombreDelObjetoADeshabilitar.disabled = true;
```

Escribe el siguiente código donde vemos un ejemplo de cómo mantener desactivados ciertos elementos hasta que el usuario no escoja una opción que queremos sea escogida antes que las que mantenemos desactivadas. Comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> body, p {margin-left:30px; font-family: sans-serif;}
.estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
.estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
</style>
<script type="text/javascript">
window.onload = function () {
// Variable global por estar declarada sin hacer uso de var
elementosEnForm = document.forms['formularioContacto'].elements;
for (var i=0; i< elementosEnForm.length; i++) {
        if (elementosEnForm[i].type != 'radio') {elementosEnForm[i].disabled = true;}
        else {elementosEnForm[i].addEventListener('click', activarElementos);}
}
}

function activarElementos() { for (var i=0; i<elementosEnForm.length; i++) {elementosEnForm[i].disabled=false;} }
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
        <div class="estiloForm">
                <form name ="formularioContacto" method="get" action="http://aprenderaprogramar.com">
                <p>Debe elegir tratamiento para activar el formulario</p>
                <label>Tratamiento</label>
                <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
                <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
                <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
                <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
                <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
                <label>Ciudad</label><select id="ciudad" name="ciudad">
                <option value="">Elija opción</option><option value="Mexico">México D.F. (MX)</option>
                <option value="Madrid">Madrid (ES)</option><option value="Santiago">Santiago (CL)</option>
                </select><br/>
                <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
                <input name="Peliculas" type="checkbox"  />Películas
                <input class="botonFormulario" type="submit" value="Enviar"/>
                <input class="botonFormulario" type="reset" value="Cancelar"/>
        </form>
        </div>
</body></html>
```

El resultado esperado es que el usuario no pueda escribir en los input de tipo text, ni seleccionar opciones del select o de tipo checkbox, ni pulsar botón de envío o reset. ¿Por qué? Porque inicialmente establecemos la propiedad disabled de todos los elementos del formulario (excepto los input de tipo radio) a true. Esta situación no cambiará hasta que el usuario elija una opción de las presentadas como input de tipo radio. Cuando detectamos el evento click sobre un objeto de tipo radio (el usuario ha elegido una de las dos opciones que le pedimos que elija) hacemos que la propiedad disabled de todos los elementos del formulario quede establecida en true, con lo cual ya pueden introducirse textos, elegir opciones o pulsar los botones de envío o cancelación.

DESACTIVAR ELEMENTOS OPTION DE UN SELECT

En ocasiones nos puede interesar desactivar no un select de forma completa sino determinados elementos del select. Por ejemplo, supongamos que se trata de una aplicación web para un hotel donde existen tres habitaciones suite: “Mediterráneo”, “Atlántico” y “Pacífico”. Supongamos que la habitación Atlántico está en obras y queremos mantener la opción visible pero deshabilitada, de forma que el usuario no pueda elegir esa opción.

Escribe este código y comprueba cómo la opción “Atlántico” no está disponible para ser seleccionada.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> body, p {margin-left:30px; font-family: sans-serif;}
.estiloForm {background-color: #f3f3f3; border: solid 2px black; margin-left:20px; width: 330px; padding:10px; }
.estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
</style>
<script type="text/javascript">
window.onload = function () {
var habitacionesSelect = document.forms['formularioContacto'].elements['habitacion'];
// habitacionesSelect.options[2].disabled=true; ¿Qué ocurre si cambia el índice y no es 2?
for (var i=0; i<habitacionesSelect.length; i++) {
        if (habitacionesSelect[i].value =='atlantico') {habitacionesSelect[i].disabled=true;}
}
}
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
        <div class="estiloForm">
                <form name ="formularioContacto" method="get" action="http://aprenderaprogramar.com">
                <p>La suite Atlántico está en obras</p>
                <label>Habitación suite:</label><select id="habitacion" name="habitacion">
                <option value="">Elija opción</option>
                <option value="mediterraneo">Mediterráneo</option>
                <option value="atlantico">Atlántico</option>
                <option value="pacifico">Pacífico</option>
                </select><br/>
                <input class="botonFormulario" type="submit" value="Enviar"/>
                <input class="botonFormulario" type="reset" value="Cancelar"/>
        </form>
        </div>
</body></html>
```

Fíjate que podemos desactivar la opción directamente usando la sintaxis habitacionesSelect.options[2].disabled=true; pero que sin embargo lo hemos hecho con un bucle for. Usar un bucle for tiene un coste (recorrer los elementos) pero nos proporciona más seguridad al basar la selección en el value 'atlantico' en lugar de en un índice numérico que podría cambiar. No podemos decir que una opción sea buena y otra mala: simplemente son alternativas que tenemos para llegar a un mismo objetivo y como programadores deberemos elegir la que consideremos más adecuada.

DESACTIVAR OTROS ELEMENTOS COMO BUTTON ó LINK

Hay más elementos a los que se le puede aplicar la propiedad disabled, por ejemplo los elementos button. La sintaxis a utilizar es la misma que hemos visto anteriormente.

También podemos desactivar elementos link como los que se emplean para referenciar un archivo CSS externo o elementos style como los que se emplean para incluir CSS interno. Usando esta posibilidad podemos activar o desactivar CSS de una página web de forma bastante sencilla. Escribe este código y comprueba cómo se activa o desactiva CSS pulsando en el botón:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> .miEstilo{background-color: yellow; font-size:3em;} input{margin:10px;}
</style>
<script type="text/javascript">
function cambiarCSS() {
var nodoStyle = document.getElementsByTagName('style');
nodoStyle[0].disabled = !nodoStyle[0].disabled;
}
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<input type="button" value="Activar/Desactivar Css" onclick="cambiarCSS()"/></br>
<div class="miEstilo">Esto es un Ejemplo</div>
</body></html>
```

El resultado esperado es que el texto “Esto es un ejemplo” se muestre con fondo amarillo y texto grande al cargar la página. Cuando pulsemos en el botón “Activar/Desactivar CSS” el texto dejará de tener el fondo amarillo y se mostrará con tamaño de letra normal al quedar el nodo style con la propiedad disabled establecida a true. Si se vuelve a pulsar el botón, la propiedad disabled toma el valor opuesto al que tuviera anteriormente. Esto lo conseguimos con la sintaxis de la línea nodoStyle[0].disabled = !nodoStyle[0].disabled;

RESUMEN

La propiedad disabled de los objetos HTMLElement del DOM nos permite deshabilitar elementos de formularios pero también dejar inhabilitados otros nodos del DOM, con lo que podemos conseguir efectos interesantes.

Hay ciertos tipos de nodos, por ejemplo un elemento div, que no admiten la propiedad disabled.

EJERCICIO

Crear el código HTML y JavaScript que cumpla con las siguientes funciones:

a) Deberá presentarse un formulario con 2 elementos select. El primer elemento select permitirá elegir país y podrá elegirse entre México, España, Perú y Colombia.

b) El segundo elemento select permitirá elegir ciudad y podrá elegirse entre México D.F., Guadalajara, Madrid, Barcelona, Lima, Trujillo, Bogotá y Cali.

c) Si el usuario selecciona ciudad sin haber elegido país, deberá mostrarse como opción elegida de país el país correspondiente a la ciudad. Por ejemplo, si el usuario selecciona Bogotá deberá aparecer como país seleccionado Colombia.

d) Si el usuario selecciona un país en primer lugar, deberán deshabilitarse todas las opciones que no correspondan a ciudades de ese país. Por ejemplo si el usuario elige España, sólo podrá elegir entre Madrid y Barcelona, debiendo estar México D.F., Guadalajara, Lima, Trujillo, Bogotá y Cali deshabilitadas. Si el usuario cambia el país, deberán cambiar las ciudades cuya elección es posible.

e) Ampliación opcional para el ejercicio: introduce medidas de seguridad adicionales para que no puedan existir incoherencias entre el país seleccionado y la ciudad. Por ejemplo, imagina que el usuario elige en primer lugar como ciudad México D.F. y luego elige como país Perú. Introduce medidas de seguridad que impidan que esto suceda.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01184E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

¿QUÉ SIGNIFICA JAVASCRIPT VOID (0) Y JAVASCRIPT: EN HREF? ¿QUÉ DIFERENCIA RETURN FALSE Y PREVENTDEFAULT? (CU01184E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº84 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

ALGUNOS DETALLES JAVASCRIPT

Vamos a detenernos a estudiar algunos detalles de JavaScript.

BURBUJEO EN LA PROPAGACIÓN DE EVENTOS

Cuando hablamos sobre los eventos en apartados anteriores del curso indicamos que pueden producirse varios eventos simultáneamente y que en este caso la respuesta a los eventos sigue un orden. Si no se indica otra cosa, el evento "burbujea" desde dentro hacia fuera y a esto se le llamaba bubbling. Por ejemplo, si tenemos tres div, uno interno, otro intermedio y otro externo, y definimos una función de respuesta al evento click sobre los elementos div, cuando hacemos click en el div interno en primer lugar se ejecuta la función de respuesta para el div interno, luego para el div intermedio y luego para el div externo.

Ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> *{font-family:sans-serif;}
div {margin:20px; padding:20px; background-color: yellow; border-style:solid;}</style>
<script type="text/javascript">
window.onload = function () {
var elemsDiv = document.querySelectorAll('div');
for (var i=0;i<elemsDiv.length;i++) {elemsDiv[i].addEventListener("click", decirHola);}
}
function decirHola() {alert('Hola soy el '+this.className);}
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<div class="externo"> Soy el div externo:
        <div class="intermedio"> Soy el div intermedio
                <div class="interno"> Soy el div interno</div>
        </div>
</div>
</body></html>
```

El resultado esperado es que si hacemos click en el div interno se muestre por pantalla: Hola soy el interno, Hola soy el intermedio, Hola soy el externo.

Aquí comprobamos que primero se ejecuta la respuesta el evento asociado al elemento más interno, y seguidamente las respuestas de elementos envolventes sucesivamente hasta llegar a completar todos los eventos.

Podemos evitar que el evento burbujee indicando que no debe propagarse hacia elementos externos. Esto lo podemos hacer cambiando la función decirHola. Cámbiala por este código y comprueba lo que ocurre:

```
function decirHola(elEvento) {
alert('Hola soy el '+this.className);
 elEvento.stopPropagation();
}
```

Ahora el único evento que se ejecuta es el primero (más interno), y al encontrarnos con stopPropagation() queda detenida la propagación.

EVITAR LA ACCIÓN DE DEFECTO

Recordar que algunos eventos tienen una acción de defecto, por ejemplo el evento submit de un formulario tiene como acción de defecto el envío del formulario, o el evento click sobre un link tiene como acción de defecto el cargar la página referenciada por el link en el navegador.

La acción por defecto puede ser anulada introduciendo preventDefault() en la función manejadora del evento. Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> *{font-family:sans-serif;}
div {margin:20px; padding:20px; background-color: yellow; border-style:solid;}</style>
<script type="text/javascript">
window.onload = function () {
var elemsLink = document.querySelectorAll("div a");
var elemsDiv = document.querySelectorAll('div');
for (var i=0;i<elemsLink.length;i++) {
elemsLink[i].addEventListener("click", decirHola);
elemsDiv[i].addEventListener("click", decirHola);
}
}
function decirHola(elEvento) {
elEvento.preventDefault();
alert('Hola soy el '+this.parentNode.className);
elEvento.stopPropagation();
}
</script>
</head>
<body>
<h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<div class="externo"> Soy el div externo: <a href="http://aprenderaprogramar.com"> Aquí un link </a>
<div class="intermedio"> Soy el div intermedio <a href="http://aprenderaprogramar.es"> Aquí un link </a>
<div class="interno"> Soy el div interno <a href="http://aprendeaprogramar.es"> Aquí un link </a> </div>
</div>
</div>
</body></html>
```

El resultado esperado es que si hacemos click en el link más interno se muestre únicamente <<Hola soy el interno>>. Sin embargo, no se carga la página web referenciada por el link debido al preventDefault(), y no se ejecuta la respuesta a los eventos asociados a los div debido a que el evento no se propaga debido a stopPropagation().

AL ESTILO TRADICIONAL

Todavía se pueden encontrar muchos programadores o código existente donde para prevenir la acción de defecto se usa una sintaxis más tradicional basada en definir la respuesta al evento dentro del código HTML y prevenir la acción de defecto escribiendo return false. Ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> *{font-family:sans-serif;}
div {margin:20px; padding:20px; background-color: yellow; border-style:solid;}</style>
<script type="text/javascript">

function decirHola(elEvento, that) {
alert('Hola soy el '+that.parentNode.className);
return false;
}

</script>
</head>
<body>
<h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<div class="externo" onclick="return decirHola(event, this);"> Soy el div externo: <a onclick="return decirHola(event,
this);" href="http://aprenderaprogramar.com"> Aquí un link </a>
<div class="intermedio" onclick="return decirHola(event, this);"> Soy el div intermedio <a onclick="return
decirHola(event, this);" href="http://aprenderaprogramar.es"> Aquí un link </a>
<div class="interno" onclick="return decirHola(event, this);"> Soy el div interno <a onclick="return decirHola(event, this);"
href="http://aprendeaprogramar.es"> Aquí un link </a> </div>
</div>
</div>
</body></html>
```

En lugar de incluir el return al final de la función también podríamos introducirlo en línea con esta sintaxis: onclick="decirHola(event, this); return false;"

De este modo, la sentencia return queda en línea después de invocar la ejecución de la función de respuesta sin necesidad del return en la llamada ni dentro de la propia función.

En este caso al pulsar en el link interno se nos muestra: <<Hola soy el interno, Hola soy el intermedio, Hola soy el externo, Hola soy el >> que se corresponde con los eventos click en el link a, más click en cada uno de los tres div (suponiendo un total de 4 elementos). La propiedad className no está definida para el nodo padre del div más externo, de ahí que nos aparezca como una cadena vacía.

Fíjate en que en JavaScript return false tiene el mismo efecto que preventDefault(). En JavaScript return false no evita la propagación del evento, del mismo modo que preventDefault() tampoco la evita. Si quisiéramos evitar la propagación del evento hemos de introducir stopPropagation().

¿QUÉ SIGNIFICA JAVASCRIPT VOID?

Cuando se revisa código HTML y JavaScript con frecuencia nos encontramos con expresiones del tipo:

```
<a href="javascript:void(0);">  Aquí un texto </a>
```

¿Qué significa javascript:void(0)?

Para entender esto primeramente nos vamos a referir a qué significa javascript: en el contexto del atributo href cuando escribimos javascript: estamos indicando que en lugar de llevar a una dirección web, se ejecute el código javascript que vaya indicado a continuación de los dos puntos.

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function mostrarMensaje() {
alert('Aham, lo que parecía un link en realidad lo que está haciendo es ejecutar una función javascript!');
}
</script>
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<a href="javascript:mostrarMensaje();"> Pulsa aquí por favor </a>
</body></html>
```

Aquí tenemos más o menos claro que lo que se indica es: cuando se pulse sobre el enlace, ejecuta la función JavaScript indicada.

Un efecto parecido se puede lograr si escribimos Pulsa aquí por favor

Sin embargo, escribir como destino del href # mueve la vista al comienzo del documento html (para ver este efecto tiene que tratarse de un enlace colocado en una página web que tenga un scroll vertical).

Hay una forma sencilla de evitar el retorno al comienzo de la página en este caso: hacer que el evento devuelva false, con lo que la acción de defecto (ir al comienzo de la página no se ejecutará). El código sería este:

 Pulsa aquí por favor

Piensa en otra alternativa como esta: Pulsa aquí por favor

¿Qué hará este código? Este código ejecutará el código javascript después de los dos puntos. Después de los dos puntos tenemos una doble barra que es el símbolo de comentario en JavaScript. ¿Entonces que hará? Pues no hará nada, porque ejecutar un comentario equivale a no hacer nada.

Este tipo de código normalmente lo encontraremos en situaciones como esta:

 Pulsa aquí para imprimir

¿Qué se está haciendo aquí?

a) Se crea el aspecto de un link con el objetivo de "avisar" al usuario de que puede hacer click sobre ese elemento.

b) El link en sí no lleva a ningún lado

c) Como respuesta al evento click se ejecuta una función JavaScript.

Este planteamiento no parece muy adecuado: incluir un link que no lleva a ninguna parte simplemente para que el usuario sepa que puede pulsar ahí para ejecutar algo. ¿Tiene sentido? Posiblemente no demasiado, porque no tiene lógica crear un link que no linka a ningún lado: es confuso. La alternativa recomendada sería incluir el texto dentro de un elemento html sin necesidad de escribir una etiqueta de link, y dotarla del estilo adecuado (como si fuera un link, pero sin serlo) para que el usuario sepa que puede pulsar ahí. La idea sería de este tipo:

<span style="comoSiFueraLink" onclick="imprimir()> Pulsa aquí para imprimir

Esto es más correcto: aquí está claro lo que se quiere hacer y se hace de una forma coherente.

En muchas ocasiones nos encontraremos algo como esto:

Pulsa aquí para logarte

Void es un operador JavaScript que evalúa (ejecuta) la expresión que se le pasa como argumento y a continuación devuelve <<undefined>>. En este caso, javascript:void(0) ejecuta 0, que en realidad no tiene ningún efecto, y devuelve undefined. ¿Y qué significa esto? Pues lo mismo que href="javascript://", es decir, que simplemente se crea un link que no enlaza con ningún lado (o podríamos decir que enlaza con undefined, lo cual es enlazar con ningún lado).

¿Entonces tiene void alguna utilidad? Puede tenerla. Como hemos visto, nos puede servir para decir que el link no lleve a ningún lado, aunque por otro lado como respuesta al evento sí puede que se ejecuten ciertos efectos colaterales.

Otra utilidad sería dar lugar a la ejecución de algo devolviendo undefined. Por ejemplo, ejecuta este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
</head>
<body>
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplos JavaScript</h3>
<a href="javascript:void(document.body.style.backgroundColor='yellow');">
 Pulsa aquí para cambiar a color de fondo amarillo
</a>
</body>
</html>
```

Aquí tenemos un link que no lleva a ningún lado (ya que void termina devolviendo undefined), y sin embargo existe un efecto colateral derivado de la expresión que se le pasa como argumento a void.

Obtendríamos el mismo efecto escribiendo esto:

Pulsa aquí para cambiar a color de fondo amarillo

Que posiblemente es más claro (se dice que “oscurece menos el código”). Pero más claro todavía es no crear un link si realmente no es un link, por tanto preferimos usar esto otro:

Pulsa aquí para cambiar a color de fondo amarillo

Obviamente para usar esto tendremos que haber definido el estilo CSS .comoSiFueraLink para darle la apariencia al texto de un link.

Algunos expertos consideran que javascript void(0) es una mala práctica de programación (de hecho hay una expresión acuñada en referencia a ello: <<javascript void(0) must die>>).

Nosotros no recomendamos su uso, pero es útil conocer su significado porque ha sido usado y sigue usándose, con lo cual es posible que nos encontremos código que se use cuando estemos revisando código desarrollado por otros programadores.

RESUMEN

Con javascript:void(0) y otras expresiones similares se busca que el usuario perciba como algo donde puede hacer click un texto, por ejemplo “Imprimir”, y para ello se simula un link. De esta forma el navegador mostrará el texto con un color y cursor especial si se pasa el ratón encima de él. Esto no es una idea muy afortunada y sin embargo ha sido bastante usada debido a que es simple.

EJERCICIO

Escribe este código en un editor y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
</head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
<a href="javascript:void(0);" onclick="if (confirm('Are you sure?')) { var f = document.createElement('form');
 f.style.display = 'none'; this.parentNode.appendChild(f); f.method = 'POST'; f.action = this.href;var m =
document.createElement('input');
 m.setAttribute('type', 'hidden'); m.setAttribute('name', '_method'); m.setAttribute('value', 'delete'); f.appendChild(m);
 var s = document.createElement('input'); s.setAttribute('type', 'hidden'); s.setAttribute('name', 'authenticity_token');
 s.setAttribute('value', 'aprenderaprogramar.com='); f.appendChild(s);f.submit(); };return false;">Pulsa aquí para
proceder</a>
</body></html>
```

a) ¿Para qué se utiliza aquí javascript: void(0)?

b) Explica paso a paso qué es lo que hace el código JavaScript que se ha incluido como respuesta al evento click.

c) ¿Qué implica el uso de return false; en este código?

d) Modifica el código para que la apariencia y resultados sean iguales pero sin usar un elemento <a> ...

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01185E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

VALIDACIÓN DE FORMULARIOS JAVASCRIPT. EJEMPLO COMPROBAR CAMPO NO VACÍO, RADIO BUTTON, CHECKBOX, SELECT ELEGIDO (CU01185E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº85 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

VALIDACIÓN DE FORMULARIOS JAVASCRIPT

Con los conocimientos que hemos ido adquiriendo durante el curso somos capaces de realizar la validación de un formulario para evitar que sea enviado si no cumple con ciertos criterios. En caso de no cumplir, podemos mostrar un aviso al usuario para que corrija o rellene los datos necesarios.

EJEMPLO DE VALIDACIÓN DE UN FORMULARIO

En esta entrega te proponemos que estudies y ejecutes el código que te mostramos a continuación. Es importante que lo analices línea a línea y que comprendas el significado de cada instrucción. Si has venido siguiendo el curso debes ser capaz de comprenderlo totalmente, ya que no estamos introduciendo conceptos nuevos, sino viendo una aplicación de conceptos ya estudiados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<style type="text/css"> body {margin-left:30px; font-family: sans-serif;} h4 {margin:0;} div {float:left;}
.estiloForm, #validacionDatos {background-color: #f3f3f3; border: solid 2px black; margin-left:10px; width:300px; }
.estiloForm{ width: 330px; padding:10px;} #validacionDatos {height:226px; padding:10px; background-color: yellow;}
.estiloForm label {display: block; width: 120px; float: left; text-align:right; margin-bottom: 35px; padding-right: 20px;}
br {clear: left;} input[type="submit"], input[type="reset"] {margin:25px 5px 10px 5px;}
</style>
<script type="text/javascript">
window.onload = function () {
        formularioContacto.addEventListener("submit", validarFormulario);
        document.getElementById('validacionDatos').style.visibility = 'hidden';
        formularioContacto.addEventListener("change", ocultarAvisos);
}

function ocultarAvisos() {document.getElementById('validacionDatos').style.visibility = 'hidden';}

function validarFormulario(elEvento) {
elEvento.preventDefault(); //Impedir envío del formulario hasta que se realice la validación
var msgValidacion = '';
var casillaDatos = document.getElementById('validacionDatos'); //Nodo donde vamos a mostrar la validación de datos
casillaDatos.innerHTML = '<h4> Aviso datos a revisar o corregir: </h4>';
var radioButTrat = document.getElementsByName("tratamiento"); //Nodos radio buttons
var checkboxElements = new Array();
var elementosSelect = document.getElementsByTagName('select');
var elementosDelForm = document.getElementsByTagName('input'); //Elementos input

/*Validación de que se haya elegido un radio button*/
var radioButElegido = false;
for (var i=0; i<radioButTrat.length; i++) {if (radioButTrat[i].checked == true) { radioButElegido=true;} }
if (radioButElegido == false){msgValidacion = msgValidacion+'<p>(*) No hay elegido tratamiento sr. o sra. </p> ';}
```

```
/*Validación de que los campos input text no estén vacíos*/
var textosConformes = true;
for(var i=0; i<elementosDelForm.length;i++) {
        if (elementosDelForm[i].type == 'text' && elementosDelForm[i].value=='') {
                msgValidacion = msgValidacion+'<p>(*) Campo '+ elementosDelForm[i].name +' está vacío </p> ';
                textosConformes = false;
        }
}

/*Validación de que se haya elegido un elemento del select */
var ciudadElegida = true; indiceElegido = document.getElementById('ciudad').selectedIndex;
if( indiceElegido == null || indiceElegido == 0 ) {
msgValidacion = msgValidacion+'<p>(*) No hay elegida una ciudad. </p> ';
ciudadElegida = false;
}

/*Validación de que se haya elegido un checkbox*/
        for(var i=0; i<elementosDelForm.length;i++) {
                if (elementosDelForm[i].type == 'checkbox') {checkboxElements.push(elementosDelForm[i]);}
        }
var checkboxMarcado = false;
for (var i=0; i<checkboxElements.length;i++) {
        if (checkboxElements[i].checked ==true) {checkboxMarcado=true;}
}
if (checkboxMarcado==false) {msgValidacion = msgValidacion+'<p>(*) No hay elegida ninguna preferencia </p> ';}

/*Decisión final: mostrar avisos si hay fallos o enviar el formulario si está correcto*/
if (radioButElegido == false || checkboxMarcado == false || textosConformes == false || ciudadElegida == false ) {
document.getElementById('validacionDatos').style.visibility = 'visible';
casillaDatos.innerHTML= casillaDatos.innerHTML+msgValidacion;
}
else { alert ('Se ha realizado la validación de datos y es conforme, se procede al envío del formulario');
document.forms['formularioContacto'].submit(); }
}
</script></head>
<body><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>
  <div class="estiloForm">
    <form name ="formularioContacto" method="get" action="http://aprenderaprogramar.com">
    <label>Tratamiento</label> <input type="radio" name="tratamiento" id="tratarSr" value="Sr."/>Sr.
   <input type="radio" name="tratamiento" id ="tratarSra" value="Sra."/>Sra.<br/>
   <label>Nombre</label><input id="nombre" name="nombre" type="text"/><br/>
    <label>Apellidos</label><input id="apellidos" name="apellidos" type="text"/><br/>
    <label>Dirección</label><input id="direccion1" name="direccion1" type="text"/><br/>
    <label>Ciudad</label><select id="ciudad" name="ciudad"><option value="">Elija opción</option>
      <option value="Mexico">México D.F. (MX)</option>  <option value="Madrid">Madrid (ES)</option>
      <option value="Santiago">Santiago (CL)</option>
    </select><br/>
        <label>Preferencias</label><input name="Libros" type="checkbox" />Libros
        <input name="Peliculas" type="checkbox"  />Películas
        <input type="submit" value="Enviar"/> <input type="reset" value="Cancelar"/>
        </form>
  </div>
<div style="float:left;"><div id="validacionDatos" style="float:left;"><h4> Aviso datos a revisar o corregir: </h4></div>
</div></body></html>
```

El resultado esperado es que si hay algún elemento que el usuario ha olvidado rellenar o marcar, se mostrará un mensaje de aviso informando de aquellos elementos que presentan problemas. Por ejemplo, si el usuario intenta enviar el formulario sin haber elegido ninguna opción ni rellenado ningún dato se mostrará el mensaje:

Aviso datos a revisar o corregir:

(*) No hay elegido tratamiento sr. o sra.
(*) Campo nombre está vacío
(*) Campo apellidos está vacío
(*) Campo direccion1 está vacío
(*) No hay elegida una ciudad.
(*) No hay elegida ninguna preferencia

Cuando el usuario realiza un cambio en el formulario, los mensajes de aviso desaparecen (para ello hemos usado el evento onchange del formulario y modificado la propiedad css visibility del contenedor donde se muestran los mensajes). Cuando el usuario vuelve a pulsar el botón enviar, se le muestran los mensajes de aviso relativos a los campos que tienen problemas. Si ningún campo presenta problemas, se muestra el mensaje: “Se ha realizado la validación de datos y es conforme, se procede al envío del formulario” y el formulario es enviado a la url de destino.

INTRODUCIR EXPRESIONES REGULARES EN LA VALIDACIÓN DE FORMULARIOS

Una técnica bastante empleada y bastante útil es realizar validaciones basadas en comprobar que la entrada del usuario cumple con un determinado patrón. Por ejemplo, supongamos que solicitamos un número de teléfono y deseamos comprobar que exactamente tiene 9 dígitos. Por ejemplo 646 77 88 22 sería válido pero 646 77 88 22 11 no sería válido.

Esto se puede hacer de forma relativamente usando funciones para cadenas de caracteres y expresiones regulares. Ambas han sido explicadas en apartados anteriores del curso, por lo que no vamos a explicarlo aquí de nuevo. Revisa estos apartados del curso si tienes dudas. Un ejemplo de validación con expresiones regulares podría ser algo como esto:

```
var telefonoValue = document.getElementById('telefono').value;
if( !(/^\d{9}$/.test(telefonoValue)) ) {
alert ('Formato de teléfono no válido');
}
```

Si no entiendes el significado de este código, revisa los apartados relacionados con expresiones regulares del curso.

EJERCICIO

Un programador ha desarrollado un código y nos han pedido que lo revisemos. Escribe este código en un editor y responde a las siguientes preguntas:

```
<html>
  <head>
          <meta charset="utf-8">
          <style type="text/css">
          input {margin:10px;}
          </style>
    <script>
      function Valida(formulario) {
        /* Validación de campos NO VACÍOS */
        if ((formulario.campo1.value.length == 0) || (formulario.campo2.value.length ==0) ||
(formulario.cpostal.value.length ==0) || (formulario.dni.value.length ==0) || (formulario.email.value.length ==0)) {
          alert('Falta información');
          return false;
        }
        if (isNaN(parseInt(formulario.campo2.value))) {
          alert('El campo2 debe ser un número');
          return false;
        }
        /* validación del CÓDIGO POSTAL*/
        var ercp=/(^([0-9]{5,5})|^)$/;
        if (!(ercp.test(formulario.cpostal.value))) {
          alert('Contenido del código postal no es un código postal válido');
          return false; }
        /* validación del DNI */
        var erdni=/(^([0-9]{8,8}\-[A-Z])|^)$/;
        if (!(erdni.test(formulario.dni.value))) {
          alert('Contenido del dni no es un DNI válido.');
          return false;  }
        /* validación del e-mail */
        var ercorreo=/^[^@\s]+@[^@\.\s]+(\.[^@\.\s]+)+$/;
        if (!(ercorreo.test(formulario.email.value))) {
          alert('Contenido del email no es un correo electrónico válido.');
          return false; }
        /* si no hemos detectado fallo devolvemos TRUE */
        return true;
      }
    </script>
  </head>
  <body>
    <form name="miFormulario" onsubmit="return Valida(this);" action="http://aprenderaprogramar.com" >
      Campo1 <input type="text" name="campo1"><br>
      Campo2 (debe ser número) <input type="text" name="campo2"><br>
      Código postal <input type="text" name="cpostal"><br>
      DNI <input type="text" name="dni"><br>
      email <input type="text" name="email"><br>
      <input type="submit" value="Enviar" name="enviar">
    </form>
  </body>
</html>
```

a) ¿Cómo se realiza la comprobación de que no hay campos vacíos?

b) ¿Cómo se realiza la comprobación de que el campo 2 es un número?

c) ¿En qué consiste la validación del código postal? Explícala paso a paso. Pon ejemplos de tres valores válidos y tres valores no válidos.

d) ¿En qué consiste la validación del DNI? Explícala paso a paso. Pon ejemplos de tres valores válidos y tres valores no válidos.

e) ¿En qué consiste la validación del email? Explícala paso a paso. Pon ejemplos de tres valores válidos y tres valores no válidos.

f) ¿Cómo se consigue que no se envíe el formulario si no se cumplen los requisitos exigidos?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01186E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

IMAGE JAVASCRIPT. NEW IMAGE. COMPLETE, NATURALWIDTH. OBTENER TODAS LAS IMÁGENES DE UNA WEB CON DOCUMENT.IMAGES (CU01186E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº86 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

OBJETOS DE TIPO IMAGE JAVASCRIPT

Ya conocemos que podemos crear un nodo del DOM de tipo imagen usando una expresión como document.createElement('img'). Sobre ese nodo podemos especificar las propiedades de la imagen como alt, src, etc. y podemos insertar el nodo en el documento HTML. Existe una forma alternativa basada en los denominados objetos tipo Image que se crean invocando new Image() como veremos a continuación.

OBJETOS DE TIPO IMAGE EN JAVASCRIPT

JavaScript define el tipo de datos HTMLElement que nos permite representar nodos del DOM. Como subtipo de HTMLElement tenemos HTMLImageElement, que permite crear nodos <img> y dispone de atributos y métodos adicionales para este tipo de nodos de imagen.

Veamos en primer lugar cómo crear un objeto de tipo HTMLImageElement. La sintaxis a utilizar es la siguiente:

```
var image1 = new Image(opcionalWidth, opcionalHeight);
```

Los valores de width (anchura) y height (altura) son opcionales. Si se especifica un solo valor se entiende que se refiere al valor width para la imagen.

El contenido de un nodo (por ejemplo lo que visualizaremos si tratamos de mostrarlo por pantalla) será el código HTML generado para la etiqueta img. Escribe este código y comprueba sus resultados (cambia la ruta de la imagen utilizada si lo deseas):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploObjetos() {
var imagen1 = new Image(600, 400);
imagen1.src =
'http://aprenderaprogramar.com/images/thumbs_portada/thumbs_divulgacion/35_errores_en_programacion.jpg';
alert('Vamos a añadir un nodo al DOM de tipo: '+imagen1);
document.body.appendChild(imagen1);
alert ('El código HTML ha quedado así: \n\n' + document.body.innerHTML + '\n\n');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado esperado es que una vez pulsamos sobre “Probar” se muestre lo siguiente:

Vamos a añadir un nodo al DOM de tipo: [object HTMLImageElement]

El código HTML ha quedado así:

<h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3>

 Probar

<img src="http://aprenderaprogramar.com/images/thumbs_portada/thumbs_divulgacion/35_errores_en_programacion.jpg" height="400" width="600">

Además, debe aparecer la imagen una vez que añadimos el nodo img al elemento body del documento.

PROPIEDADES Y MÉTODOS DE OBJETOS TIPO IMAGE

Disponemos de propiedades y métodos aplicables a los objetos de tipo Image. Algunas propiedades a destacar son:

SINTAXIS	UTILIDAD	Ejemplos aprenderaprogramar.com
img1.alt	Define el atributo alt (texto alternativo) para la imagen	img1.alt = 'Humor informático';
img1.complete	Devuelve un valor booleano que es true si la imagen ha sido completamente cargada por el navegador o false en caso contrario.	document.getElementById("img1").complete; Nota: puede no dar los resultados deseados si la imagen está cacheada o si la imagen se ha definido a través de JavaScript.
img1.width	Define el atributo width (anchura) de la imagen, en píxeles	img1.width=200;
img1.height	Define el atributo height (altura) de la imagen, en píxeles	img1.height=200;
img1.naturalWidth	Devuelve el ancho de la imagen original en píxeles	img1.width=img1.naturalWidth; Nota: la url de la imagen debe estar definida para poder recuperar su tamaño
img1.naturalHeigth	Devuelve la altura de la imagen original en píxeles	img1.height=img1.naturalHeight; Nota: la url de la imagen debe estar definida para poder recuperar su tamaño
img1.src	La url de la imagen, relativa o absoluta	img1.src = 'dibujo.jpg';

OBTENCIÓN DE TODAS LAS IMÁGENES EN UN DOCUMENTO HTML

El objeto document tiene una propiedad relacionada con imágenes que es interesante, ya que nos permite recuperar una colección de objetos imagen con todas las imágenes presentes en el documento HTML, ordenadas según su orden de aparición en el documento HTML. Para ello se usa esta sintaxis:

```
var coleccionDeImagenes = document.images;
```

Dado que obtenemos una colección, podemos acceder a un elemento de la misma usando un índice (siempre que el índice sea válido). Por ejemplo coleccionDeImagenes[2] nos devolvería el nodo del DOM con la tercera imagen que aparece en el documento. También se admite la sintaxis document.images.item(2) con el mismo resultado.

También podemos usar la propiedad length de las colecciones para saber cuántas imágenes han sido encontradas. Por ejemplo alert ('Se han encontrado '+document.images.length +' imágenes');

EJERCICIO

Un programador ha desarrollado un código y nos han pedido que lo revisemos. Escribe este código en un editor y responde a las siguientes preguntas (nota: cambia la ruta de las imágenes si quieres):

```
<html>
<head>
<script type="text/javascript">
image01= new Image()
image01.src="http://i877.photobucket.com/albums/ab336/cesarkrall/Divulgacion/logonotplusplus.png"
image02= new Image()
image02.src="http://i877.photobucket.com/albums/ab336/cesarkrall/Divulgacion/DV00405A_1.jpg"
function rollover(imagename, newsrc){
document.images[imagename].src=newsrc.src
}
</script>
</head>
<body style="text-align:center; margin:50px;">
<p> Pasa el mouse sobre la imagen </p>
<a  href="#" onmouseover="rollover('example', image02)"
onmouseout="rollover('example', image01)">
<img src="http://i877.photobucket.com/albums/ab336/cesarkrall/Divulgacion/logonotplusplus.png" name="example">
</a>
</body>
</html>
```

a) ¿En qué ámbito se encuentran image01 e image02?

b) ¿Qué tipo de objetos son image01 e image02?

c) ¿Aparecerán errores debido a la falta de puntos y coma de terminación en las instrucciones de JavaScript? ¿Por qué?

d) Al acceder a una imagen con la sintaxis document.images[imagename], ¿se está usando un índice numérico para acceder a la colección? Si es numérico, indicar qué valores son los que se usan. Si no es numérico, indicar qué tipo de índice es.

e) El efecto esperado es que la imagen que se muestra cambie cuando pasamos el puntero del ratón por encima de ella. ¿Por qué se produce ese cambio? Explícalo brevemente.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01187E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

ERROR, THROW Y TRY CATCH JAVASCRIPT . MESSAGE, FILENAME, LINENUMBER. EVAL Y RANGEERROR, REFERENCE ERROR. EJEMPLOS (CU01187E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº87 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

TRATAMIENTO DE ERRORES

Los programadores JavaScript se enfrentan con frecuencia a la aparición de errores durante la ejecución del código en los navegadores. Al igual que otros lenguajes de programación, JavaScript permite la captura de esos errores con instrucciones que permiten identificar y tratar dichos errores.

ERROR, THROW Y TRY CATCH

La gestión de errores en tiempo de ejecución tiene como elementos principales las sentencias try – catch, throw, y el tipo de datos definido por JavaScript Error.

El tipo de datos Error es un tipo de datos predefinido de JavaScript que permite crear objetos de tipo Error. Los objetos de tipo Error son creados automáticamente por JavaScript cuando se produce un error en tiempo de ejecución, pero también podemos definir la creación de objetos Error a través del código.

Un objeto Error representa un error y tiene propiedades asociadas (por ejemplo propiedades que informan del tipo de error de que se trata). Normalmente un error no se crea y "se guarda", sino que un error "se lanza" (is thrown). Al lanzarse un error, el flujo o ejecución prevista del script se verá alterado. Esa alteración puede dar lugar a la detención del script si el error impide que prosiga la ejecución, o a que el script continúe ejecutándose con anomalías, etc.. Como forma de prevenir que el script se detenga sin más, o que aparezcan disfunciones no previstas, existe la posibilidad de capturar y tratar el error usando las sentencias try – catch.

Un esquema típico para tratamiento de errores será el siguiente:

```
try {
//Código que vamos a ejecutar
// Si se produce un error se lanza una excepción y se salta al catch
}
catch (e) {
  // e representa el error lanzado
  // mensajes de alerta, acciones a ejecutar, etc.
}
```

Vamos a partir del siguiente ejemplo que al ser ejecutado dará lugar a que salte un error. Escribe este código, activa la consola de tu navegador y comprueba cómo al ejecutar el código te aparece un mensaje de error cuando haces click sobre el texto "Probar".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
cambiarImagen(imagen1);
alert('Se cambió la imagen'+window.imagen1);
alert('A continuación se le solicitarán los datos fiscales');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body>
</html>
```

El resultado esperado es que no ocurra nada y que en la consola se muestre un mensaje de error similar a este: <<ReferenceError: cambiarImagen is not defined>>. Este error nos informa de que se está invocando una función que no está definida (además tenemos un segundo error, ya que imagen1 tampoco está definido).

Supongamos que este error aparezca "sin querer", debido a un error del programador o debido a que el usuario o el código HTML no facilitan los datos que se espera. Un bloque try catch puede envolver a un fragmento de código (tan amplio como se desee) y establecer un tratamiento para el error, de modo que el script continuará ejecutándose siempre que sea posible.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        cambiarImagen(imagen1);
        alert('Se cambió la imagen'+imagen1);
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre una ventana de mensaje de alerta con un texto similar al siguiente: <<Se produjo un error. Referencia: ReferenceError: cambiarImagen is not defined>> y que seguidamente se muestre el mensaje <<A continuación se le solicitarán los datos fiscales>>.

La diferencia entre este código y el anterior donde no existía un bloque try catch es que como comprobamos aquí el error se captura, es tratado y la ejecución del script continúa. En el caso anterior no llegábamos a ver el mensaje "A continuación se le solicitarán los datos fiscales" porque el script quedaba detenido.

Un bloque try catch se introducirá, típicamente, en fragmentos de código donde sea previsible que pueda producirse un error, con el fin de capturarlo y tratarlo.

Otro esquema típico para tratamiento de errores será el siguiente:

```
try {
//Código que vamos a ejecutar
if (evaluación indica que existe un error) {throw new Error("Descripción del error"); }
}
catch (e) {
  // e representa el error lanzado
  // mensajes de alerta, acciones a ejecutar, etc.
}
```

Escribe este código y comprueba sus resultados al introducir diferentes números:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        var numeroUsuario = prompt('Introduzca un número del 1 al 9 por favor: ');
        if (isNaN(numeroUsuario)|| numeroUsuario<1 ||numeroUsuario>9) {
                throw new Error('Número introducido no válido');
        }
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que si introducimos un número que se considera válido (por ejemplo el 4), la ejecución se realice con normalidad. Por el contrario, si introducimos sun número que se considera no válido como el 88, se lanzará la excepción y se mostrará el mensaje <<Número introducido no válido>>. Aquí realmente no necesitábamos lanzar una excepción (ya que podríamos haber controlado el proceso de otra manera), por lo que este código debemos tomarlo únicamente como “un ejemplo” y no como una referencia de cómo hacer las cosas. Lo más habitual entre programadores es no lanzar excepciones excepto ante circunstancias que difícilmente pueden ser tratadas de otra manera (por ejemplo que el acceso a un recurso que normalmente esté disponible, como un fichero, pueda no estar disponible en un momento concreto).

CLÁUSULA FINALLY

Una instrucción try además de bajo el esquema try {...} catch { ... } que hemos visto anteriormente, se puede combinar con una cláusula finally de alguna de estas maneras: try {...} catch {...} finally {...} y también con try {...} finally {...}.

Las sentencias incluidas dentro de una cláusula finally se ejecutarán independientemente de que se haya producido un error o no durante la ejecución del bloque try. El objetivo habitual de una cláusula finally es liberar un recurso (por ejemplo cerrar un archivo) que haya podido ser comprometido anteriormente.

El esquema habitual será el siguiente:

```
abrirElRecurso(); //Por ejemplo accedemos a un fichero
try {
  // Ejecutamos acciones previstas con el recurso
  escribirEnElFichero(fechaActual);
}
finally {
  cerrarElRecurso(); // Por ejemplo cerramos el fichero
}
```

Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemploError() {
try{
        var numeroUsuario = prompt('Introduzca un número del 1 al 9 por favor: ');
        if (isNaN(numeroUsuario)|| numeroUsuario<1 ||numeroUsuario>9) {
                throw new Error('Número introducido no válido');
        }
}
catch(e){ alert('Se produjo un error. Referencia: '+e);}
finally {alert('Se ejecuta siempre haya o no captura de error');}
alert('A continuación se le solicitarán los datos fiscales');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploError()"> Probar </div>
</body></html>
```

El resultado esperado es que el mensaje <<Se ejecuta siempre haya o no captura de error>> aparezca tanto cuando se lanza un error como cuando no se lanza.

TIPOS ESPECIALIZADOS DE ERROR

Además de los objetos Error que indican un error genérico, podemos lanzar errores especializados. Entre los errores especializados tenemos los siguientes:

Tipo de error	Significado	Ejemplo
EvalError	Representa un error relacionado con la ejecución de la función eval()	throw new EvalError();
RangeError	Representa que un valor está fuera del rango permitido para una variable o parámetro.	throw new RangeError();
ReferenceError	Representa que se ha intentado invocar una función u objeto y no existe en ese ámbito	throw new ReferenceError();
SyntaxError	Representa un error en la sintaxis del código que se pretende ejecutar con eval()	throw new SyntaxError();
TypeError	Representa un error debido a que una variable o parámetro no tienen un tipo válido.	throw new TypeError();
URIError	Representa un error cuando se pasan parámetros no válidos a las funciones encodeURI() ó decodeURI()	Throw new URIError();

ANIDAMIENTO DE TRY

Se pueden incluir bloques try dentro de otros bloques try. Normalmente cada bloque try llevará su catch correspondiente, pero en caso de que un bloque try interno a otro carezca de catch, en caso de lanzarse una excepción se accederá al catch del bloque externo.

PROPIEDADES DE LOS OBJETOS ERROR

La creación de objetos Error puede realizarse indicando simplemente new Error() o bien introduciendo parámetros. Los parámetros admitidos son:

```
new Error (opcional_message, opcional_fileName, opcional_lineNumber);
```

Esta información puede ser usada durante el tratamiento del error de la forma en que se considere oportuno. Por ejemplo podríamos escribir:

catch(e) { alert ('Se produjo un error. Referencia: '+e.message + ' Línea: '+e.lineNumber +' Fichero: '+e.fileName);}

Los errores que se lanzan automáticamente llevarán asociado un contenido para estas propiedades de forma automática. Al mensaje descriptivo se puede acceder con la propiedad message, y al nombre de archivo y número de línea con las propiedades lineNumber y fileName.

En los errores que lanzamos nosotros con new Error las propiedades del objeto pueden ser establecidas por nosotros, pero si no establecemos las propiedades, éstas tomarán los valores asociados automáticamente.

Otra propiedad de los objetos Error es name. Esta propiedad inicialmente toma el valor <<Error>>, pero podemos establecer por ejemplo e.name ='Error-Usuario'.

Al invocar e.toString() se devuelve la concatenación de name con message.

EJERCICIO

Un programador ha desarrollado un código y nos han pedido que lo revisemos. Escribe este código en un editor, ejecútalo y responde a las siguientes preguntas:

```
<html>
 <head>
<meta charset="utf-8">
<style type="text/css"> input {margin:10px;} </style>
<script>
function validarPassword(password){
  try  {
    if(password.length < 5 ) {      throw "SHORT";    }else if(password.length > 10 ) {  throw "LONG";    }
    alert("Password Validated!");
  } catch(e) {
    if(e == "SHORT"){  alert("Not enough characters in password!"); }
    else if(e == "LONG"){ alert("Password contains too many characters!");   }
  }finally{    document.miFormulario.password.value="";   }
  alert("La revisión ha terminado.");
}
 </script>
   </head>
   <body>
      <form name="miFormulario" onsubmit="validarPassword(document.getElementById('pass').value)" action="#" >
         Nombre de usuario: <input type="text" name="campo1"><br>
         Password: <input id="pass" type="password" name="password"><br>
         <input type="submit" value="Comprobar" name="comprobar">
      </form>
   </body>
</html>
```

a) Busca información en internet y respode: ¿Qué significado tiene una instrucción como throw "SHORT";? ¿A qué da lugar? ¿Qué diferencia hay entre throw "SHORT" y throw new Error('SHORT')?

b) ¿Cuál es el objetivo que parecía pretender cumplir el autor del código?

c) ¿En qué casos se ejecuta la cláusula finally incluida en el código?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01188E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

COOKIES JAVASCRIPT: ¿QUÉ SON? FECHA EXPIRES, MAX AGE. PATH, DOMAIN, SECURE. CREAR, LEER, BORRAR, AÑADIR. DOCUMENT.COOKIE . EJEMPLOS (CU01188E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº88 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

COOKIES CON JAVASCRIPT

Los programadores JavaScript se enfrentan con frecuencia a la aparición de errores durante la ejecución del código en los navegadores. Al igual que otros lenguajes de programación, JavaScript permite la captura de esos errores con instrucciones que permiten identificar y tratar dichos errores.

TRASLADAR INFORMACIÓN DURANTE LA NAVEGACIÓN

Hemos visto muchos ejemplos donde trabajamos sobre un documento HTML, pero obviamente durante el desarrollo de aplicaciones reales el usuario visitará numerosos documentos HTML que se corresponderán con diferentes urls durante una navegación a través de una página web o una aplicación web.

Para situarnos, procede a hacer lo siguiente: crea un archivo html con el siguiente contenido y guárdalo con el nombre de archivo inicio.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var miVariableGlobal = 8;
window.onload =
alert ('En esta url mi variable global vale: ' + miVariableGlobal);
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" ><a href="item1.html"> Pulse aquí para visitar la sección "Animales
salvajes"</a></div>
</body>
</html>
```

Seguidamente crea un documento HTML con el siguiente contenido y guárdalo con el nombre de archivo item1.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
window.onload = alert ('En esta url mi variable global vale: ' + miVariableGlobal);
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" ><a href="inicio.html"> Pulse aquí para volver</a></div>
</body>
</html>
```

Comprueba que al cargar la página inicio.html se muestra por pantalla el mensaje << En esta url mi variable global vale: 8>>. Sin embargo, cuando hacemos click en el enlace y llegamos al archivo item1.html no se muestra nada. De hecho, si desplegamos la consola del navegador observaremos que aparece un error del tipo << ReferenceError: miVariableGlobal is not defined>>. Lo que esto nos deja claro es que las variables JavaScript tienen una vida limitada al propio documento HTML en el que van insertas y su información desaparece cuando se carga un nuevo documento HTML.

Cargar un archivo común donde definamos unas variables nos puede servir para disponer de dichas variables con su valor inicial en diferentes urls, pero ¿qué ocurre si el valor de esa variable se modifica y continúa la navegación? Que no dispondremos de información sobre dicha modificación en el resto de urls, ya que "se perderán".

Lo que vamos buscando es la persistencia de la información, es decir, que no desaparezca la información cuando cambiamos de url. Esto hay diferentes maneras de abordarlo: se puede transmitir información almacenándola en base de datos y recuperándola posteriormente, también usando formularios para enviar información, y de otras maneras. En estos ejemplos el trabajo (base de datos, recuperación de información de formularios) se basaría principalmente en lenguajes que trabajan del lado del servidor. Pero nosotros vamos buscando una forma de trabajar del lado del cliente, en este caso con JavaScript.

CONCEPTO DE COOKIE

Las cookies fueron creadas por trabajadores de la empresa Netscape como forma de dar una respuesta sencilla a la necesidad de almacenar información relacionada con la identificación de usuarios y acciones que un usuario desarrolla durante la navegación. Por ejemplo se planteaba la siguiente cuestión: si un usuario accede a una tienda web y queremos que pueda ir agregando productos a un carrito de compras, ¿cómo saber qué productos ha almacenado cuando cambia de url? Sin el uso de alguna forma de dotar de persistencia a la información, la información se perdía. Y cierta información sería muy costoso manejarla con herramientas como bases de datos o formularios.

La forma de solucionar esto inventar las cookies. Las cookies son información que se almacena en el navegador de forma persistente, es decir, que no desaparece cuando el usuario navega a través de diferentes urls. Además, las cookies son enviadas al servidor cuando el usuario hace una petición, de modo que el servidor puede conocer esa información y actuar en consecuencia.

Las cookies pueden ser creadas de diferentes maneras, por ejemplo:

a) Ser creadas por el servidor, y enviarlas al navegador para que las almacene. Supongamos que entramos en una página web como los foros de aprenderaprogramar.com. En ese momento somos usuarios anónimos, pero una vez introduzcamos nuestro nombre de usuario y password (por ejemplo supongamos que somos el usuario Albert Einstein), el servidor envía una cookie al navegador que podría ser:

session_id_foros_apr = 6n4465736gf9863b52e641757fa0b7db, donde sesión_id_foros_apr es el nombre la cookie y la cadena de letras y números el valor que tiene la cookie, lo que permitirá saber al servidor que quien hace una petición es la misma persona (el mismo computador cliente) que había hecho una petición anteriormente. La comprobación de que sea el mismo computador cliente se basa en comparar el valor de esa cadena larga: si coincide, es el mismo usuario, si es diferente, es otro

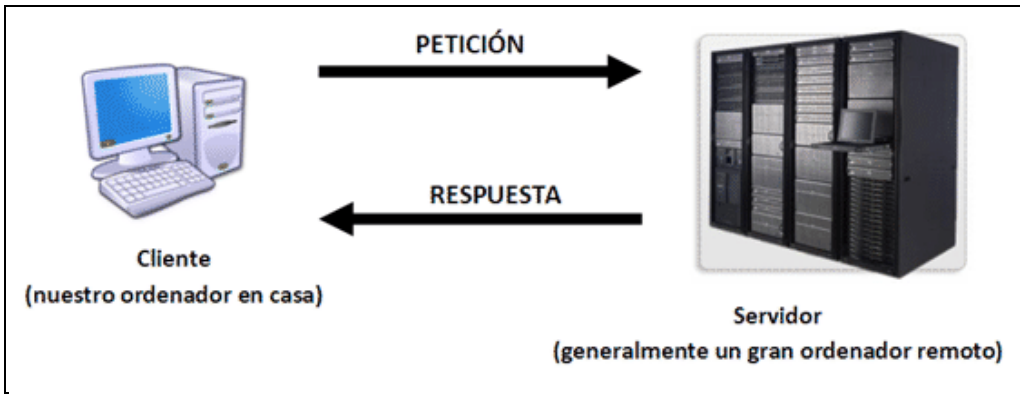
usuario. Un usuario no puede conocer el valor de la cookie que tiene otro debido a que existen prácticamente infinitas posibilidades de combinación de letras y números. Si se usan cadenas arbitrariamente largas, es prácticamente imposible averiguar por casualidad o mediante pruebas el contenido de una cookie.

Ahora cada vez que cambiemos de url esta cookie es enviada al servidor. El servidor lee la cookie y comprueba: ¡esta cookie pertenece al usuario Albert Einstein!

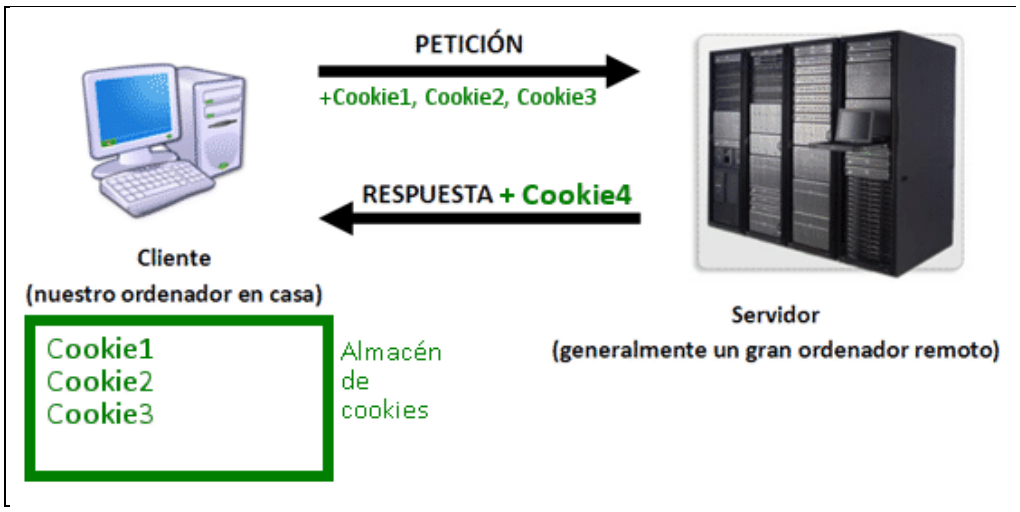
Aunque el usuario haya cambiado de url, la cookie informa en cada momento que él es Albert Einstein, con lo cual no hace falta estar introduciendo en cada url que se visite el nombre de usuario y password.

b) Ser creadas mediante JavaScript en el navegador, almacenarse en el navegador y enviarse posteriormente al servidor en cada comunicación que tenga lugar.

Simplificadamente podríamos decir que navegar sin cookies sería algo similar a lo que se muestra en este esquema, donde cada petición al servidor equivale a una "petición nueva":



Mientras que navegar con cookies sería algo similar a lo que se muestra en este esquema, donde el navegador está almacenando información en un almacén de cookies y con cada petición que hace esas cookies son transmitidas (y además pueden irse añadiendo más cookies a las ya existentes previamente):



Resumiendo: las cookies son datos que se almacenan en el navegador del usuario.

Las cookies pueden presentar algunos problemas de seguridad, pero en general se considera que si se utilizan de forma adecuada son un mecanismo que resulta práctico para realizar muchas de las tareas que normalmente se requieren en la navegación web.

TIEMPO DE VIDA DE LAS COOKIES

Las cookies son datos temporales, es decir, su intención no es almacenarse "para siempre", sino almacenarse por un tiempo para facilitar la navegación. Una cookie puede tener asociada una fecha de borrado o expiración, en cuyo caso permanecerá en el navegador del usuario hasta que llegue dicha fecha (a no ser que el usuario decida hacer un borrado de cookies). En este caso, puede ocurrir que el usuario cierre el navegador y lo abra al cabo de unas horas o días y la información en forma de cookies siga estando ahí. Las cookies con fecha de borrado se suelen llamar cookies persistentes, porque no se destruyen excepto cuando llega la fecha de expiración.

Otras cookies no tienen fecha de borrado o expiración, o si la tienen es muy corta (pongamos que una hora de duración). Si la cookie no tiene fecha de borrado, se destruye cuando se cierra el navegador.

Tener en cuenta que los navegadores pueden almacenar información de otras maneras además de como cookies (por ejemplo como opciones de configuración, perfiles de usuario, contraseñas, etc.).

Tener en cuenta también que las cookies (al igual que JavaScript) pueden desactivarse en los navegadores. La mayoría de los usuarios navegan con cookies activadas (al igual que con JavaScript activado), pero teóricamente un usuario puede deshabilitarlas.

CONTENIDOS DE LAS COOKIES

Las cookies podemos verlas como pequeños ficheros de texto que se almacenan en el navegador, cuyo contenido es el siguiente:

1. Un par nombre – valor que define la cookie. Por ejemplo el nombre puede ser user_name_foros_apr y el valor 6n4465736gf9863b52e641757fa0b7db

2. Una fecha de caducidad (en algunos casos, estará indefinida, con lo cual la cookie será borrada cuando se cierre el navegador). La fecha se expresa en tiempo UTC (ver entregas del curso donde se explican los formatos de tiempo) o como un número de segundos dede el momento actual.

3. El dominio y ruta del servidor donde la cookie fue definida, lo que permite que si existieran dos cookies con el mismo nombre se pudiera saber qué cookie corresponde a cada url que se visita. No está permitido falsear dominios, es decir, si el dominio desde el que se establece una cookie es aprenderaprogramar.com no se podría poner como información asociada a la cookie que el dominio es microsoft.com, sino que la cookie únicamente puede ir asociada a aprenderaprogramar.com. Esto permite que si se está navegando por un sitio, no haya necesidad de enviar todas las cookies almacenadas en el navegador al servidor de ese sitio, sino únicamente las cookies relacionadas con ese sitio.

La ruta permite especificar un directorio específico al cual se deba considerar asociada la cookie. De este modo, el navegador no tendría que enviar la cookie a todas las páginas de un dominio, sino solo a las páginas concretas cuya ruta esté definida. Normalmente la ruta definida es simplemente << / >> lo que significa que la cookie es válida en todo el dominio.

COOKIES CON JAVASCRIPT: DOCUMENT.COOKIE

Las cookies podemos verlas como pequeños ficheros de texto que se almacenan en el navegador.

Para crear una cookie con JavaScript podemos usar una sintaxis como esta:

```
document.cookie =
'nombreCookie=valorCookie;
expires=fechaDeExpiración;
path=rutaParaLaCookie';
```

Podemos dejar sin especificar expires (la cookie se borrará al cerrar el navegador) y path (la cookie quedará asociada al dominio), con lo que la definición quedaría:

```
document.cookie = 'nombreCookie = valorCookie;';
```

También se pueden indicar como parámetros de la cookie domain (dominio) y secure (en este caso a secure no se le asigna ningún valor, basta con incluir la palabra. Si se incluye, significa que la cookie sólo se enviará si la conexión se está realizando bajo un protocolo seguro como https).

En lugar de expires se puede usar max-age. En este caso, en lugar de especificar una fecha concreta se especifica el número de segundos desde la creación de la cookie hasta su caducidad. Por ejemplo: "theme=blue; max-age=" + 60*60*24*30 + "; path=/; domain=aprenderaprogramar.com; secure" serviría para indicar que el nombre de la cookie es theme, su valor blue, su fecha de expiración 30 días (expresado en segundos resulta 30 días * 24 horas/día * 60 minutos/hora * 60 segundos/minuto").

Cuando se van creando cookies, éstas se van añadiendo a document.cookie (es decir, document.cookie no funciona como una propiedad que se vaya sobreescribiendo, sino que cada definición de document.cookie añade una cookie a la colección de cookies).

Una cookie se puede redefinir: para ello simplemente hemos de usar document.cookie indicando el mismo nombre de cookie que uno que existiera previamente. Los datos de esa cookie serán sobreescritos quedando reemplazados los anteriormente existentes por los nuevos.

Una cookie se puede eliminar: para ello hemos de usar document.cookie indicando el nombre de cookie que queremos eliminar y establecer una fecha de expiración ya pasada (por ejemplo del día de ayer, o simplemente indicar expires=Thu, 01 Jan 1970 00:00:00 UTC;). El navegador al comprobar que la fecha de caducidad de la cookie ha pasado, la eliminará.

RECUPERAR EL CONTENIDO DE COOKIES

Para recuperar el contenido de cookies hemos de trabajar con document.cookie como si fuera una cadena de texto donde las cookies se organizan de la siguiente manera:

```
nombreCookieA=valorCookieA; nombreCookieB=valorCookieB; ... ; nombreCookieN = valorCookieN;
```

Para recuperar el valor de la cookie, hemos de buscar dentro de esa cadena de texto. Para ello usaremos las herramientas que nos proporciona JavaScript, y hay varias maneras de hacerlo. A continuación vamos a ver un ejemplo.

Escribe este código y guárdalo con el nombre de archivo inicioCookie.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var miVariableGlobal = 8;
document.cookie = 'cookieNombreUsuario = Alberto; expires=Thu, 12 Aug 2049 20:47:11 UTC; path=/';
document.cookie = 'cookieEdadUsuario='+miVariableGlobal+'; expires=Thu, 12 Aug 2049 20:47:11 UTC; path=/';
alert ('El contenido de document.cookie es: '+document.cookie);
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" ><a href="item1Cookie.html"> Pulse aquí para visitar la sección "Animales
salvajes"</a></div>
</body>
</html>
```

Con este código hemos definido dos cookies. Al ejecutar el código el resultado esperado es que se muestren por pantalla: << El contenido de document.cookie es: cookieNombreUsuario=Alberto; cookieEdadUsuario=8 >>

Escribe ahora este otro código y guárdalo con el nombre de archivo item1Cookie.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
//Función para encontrar el valor de una cookie
function readCookie(nombre) {
        var nombreIgual = nombre + "=";
        var numeroCookies = document.cookie.split(';');
        for(var i=0;i < numeroCookies.length;i++) { //Recorremos todas las cookies
                var valorCookie = numeroCookies[i]; //Analizamos la cookie actual
                while (valorCookie.charAt(0)==' ') {valorCookie = valorCookie.substring(1,valorCookie.length); }
//Eliminamos espacios
                if (valorCookie.indexOf(nombreIgual) == 0) {return
valorCookie.substring(nombreIgual.length,valorCookie.length);} //Devolvemos el valor
        }
        return null; //Si numeroCookies es cero se devuelve null
}
```

```
window.onload = function () {alert ('El valor de la cookieNombreUsuario es: '+readCookie('cookieNombreUsuario'));
alert ('El valor de la cookieEdadUsuario es: '+readCookie('cookieEdadUsuario'));
};

</script></head>
<body><div id="numeroCookiesbecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos
JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" ><a href="inicioCookie.html"> Pulse aquí para volver</a></div>
</body></html>
```

Prueba ahora a entrar en la página inicioCookie.html y pulsar sobre el enlace que nos lleva a item1Cookie.html. Al entrar en item1Cookie.html el resultado esperado es que se muestre por pantalla: << El valor de la cookieNombreUsuario es: Alberto, El valor de la cookieEdadUsuario es: 8 >>

Vamos a destacar algunas cuestiones del código que hemos usado:

a) Las cookies se crean en una url (inicioCookie.html) y se almacenan en el navegador.

b) Las cookies se recuperan en otra url (item1Cookie.html) usando la función readCookie. De esta manera tenemos una solución al problema de transmitir información entre distintas urls que habíamos planteado al principio de esta explicación. La función readCookie utiliza distintas construcciones y funciones nativas de JavaScript que deberías ser capaz de entender completamente si has venido siguiendo el curso desde el principio. Si no entiendes algo de este código, repasa las entregas anteriores del curso.

c) Las cookies que se reciben son sólo las asociadas al dominio. ¿Y cuál es el dominio? En este caso el asociado a la ruta de nuestro computador donde estemos trabajando. Si estuviéramos trabajando sobre una web on-line, el dominio serían todas las páginas pertenecientes al sitio web.

d) Las cookies que se han creado tienen fecha de caducidad 2049. Es decir, no se borrarán del navegador hasta 2049, excepto si realizamos el borrado de cookies del navegador (al borrado de este tipo de datos se le denomina en sentido amplio “Limpieza de la caché del navegador”).

EJERCICIO

Un programador ha desarrollado este código. Revísalo y responde a las siguientes preguntas:

```
function createCookie(name,value,days) {
	if (days) {
		var date = new Date();
		date.setTime(date.getTime()+(days*24*60*60*1000));
		var expires = "; expires="+date.toGMTString();
	}
	else var expires = "";
	document.cookie = name+"="+value+expires+"; path=/";
}

function eraseCookie(name) { createCookie(name,"",-1); }
```

a) ¿Para qué sirve la función createCookie? Explica paso a paso qué es lo que hace esta función.

b) ¿Para qué sirve la función eraseCookie? Explica paso a paso qué es lo que hace esta función.

c) Escribe el código de una función de nombre mostrarTodasLasCookies() que muestre el nombre y valor de todas las cookies existentes. Por ejemplo, si hay dos cookies deberá mostrarse algo como esto:

Hay 2 cookies en el documento

Cookie 1 con nombre: cookieNombreUsuario y valor: Alberto

Cookie 2 con nombre cookieEdadUsuario y valor 8

d) Usando las funciones createCookie, eraseCookie y mostrarTodasLasCookies() crea un script que pida 3 nombres y valores de cookies al usuario, y cree las cookies correspondientes. A continuación deberá mostrar las cookies existentes y sus valores. Finalmente, deberá borrar la última cookie existente y volver a mostrar todas las cookies y sus valores.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01189E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

USE STRICT JAVASCRIPT. QUÉ ES STRICT MODE (MODO ESTRICTO). WITH. THE GOOD PARTS. HACIA NUEVAS VERSIONES. (CU01189E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº89 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

STRICT MODE

El lenguaje JavaScript ha sido denostado por algunos por considerarlo poco consistente y alabado por otros por su gran potencia. Se suele citar el libro "JavaScript: The Good Parts" (JavaScript: las partes buenas) como referente para ilustrar que JavaScript tiene cosas buenas y otras cosas que no son buenas y por tanto supuestamente no se deberían usar.

Nosotros consideramos que las personas que adquieran un conocimiento avanzado de JavaScript deben guiarse por su propio criterio y decidir cómo usarlo. Por el contrario, las personas que sólo tienen un conocimiento superficial tienen que guiarse por lo que hacen o proponen otros. Te animamos a que progresivamente vayas conformando tu propio criterio.

DIRECTIVA USE STRICT

La directiva use strict es una directiva que no supone una instrucción de código, sino que indica el modo en que el navegador debe ejecutar el código JavaScript. Podríamos hablar de dos modos de ejecución JavaScript: el <<normal mode>>, que es el que hemos estudiado hasta ahora, y el <<strict mode>>, que vamos a explicar.

Declarar que se use strict mode supone algunos cambios en cuanto al código que admite o no admite el navegador. Por ejemplo en strict mode es obligatoria la declaración de variables, mientras que en el modo normal no es necesario declarar una variable para usarla.

Para indicar que el código debe ser considerado en modo estricto se escribe lo siguiente:

```
'use strict';
```

Esta directiva la escribiremos normalmente al comienzo de un fichero, en cuyo caso afectará a todo el código y todas las funciones existentes dentro de él, o bien al comienzo de una función, en cuyo caso afectará sólo al código en el ámbito de dicha función.

Para situarnos, activa la consola del navegador y procede a hacer lo siguiente: crea un archivo html con el siguiente contenido y guárdalo con el nombre de archivo sinstrict.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() { nombre = 'Carlos'; alert('Soy '+nombre); }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que no haya mensajes de error y que al pulsar en “Probar” por pantalla se muestre el mensaje <<Soy Carlos>>.

Crea ahora un archivo html con el siguiente contenido y guárdalo con el nombre de archivo constrict.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
'use strict';
nombre = 'Carlos';
alert('Soy '+nombre);
}
</script>
</head>
<body>
<div id="cabecera">
<h2>Cursos aprenderaprogramar.com</h2>
<h3>Ejemplos JavaScript</h3>
</div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

El resultado esperado es que cuando se pulse sobre “Probar” no se muestre ningún mensaje y podremos observar un error en consola del tipo << ReferenceError: assignment to undeclared variable nombre>>.

Aquí vemos cómo el strict mode está afectando a qué hace el intérprete del navegador. Sin strict mode, asumen que cualquier variable no declarada debe considerarse creada implícitamente. Con strict mode, asume que una variable no declarada generará un error.

Ten en cuenta que algunos navegadores, especialmente los más antiguos, no reconocen la directiva strict mode, y en estos navegadores el resultado es que no tiene ningún efecto (como si no existiera).

¿Por qué apareció el strict mode? En versiones antiguas de JavaScript no existía el strict mode. Este surgió debido a que parte de la comunidad de programadores se quejaba porque JavaScript era poco seguro a la hora de programar y se consideraba que sería adecuado introducir restricciones que forzaran a que el código debiera cumplir mayores requisitos para garantizar su coherencia. Por ejemplo, es habitual considerar que es una mala práctica no declarar las variables antes de usarlas porque puede inducir a errores (por ejemplo, si tenemos una variable cuyo nombre es <<aceptado>> y aparece otra variable cuyo nombre es <<aceptada>>, si ambas están declaradas se entiende que son variables distintas. Pero si no están declaradas, ¿son variables distintas o podría tratarse de un error al escribir el nombre de la variable?).

Además de no permitir el uso de variables declaradas, hay otras prácticas de programación que son admitidas por JavaScript clásico pero que no son admitidas en strict mode.

Citaremos algunas:

Práctica admitida en modo normal pero no admitida en strict mode	Ejemplo
Uso de variables no declaradas	nombre = 'juan';
Borrar una variable u objeto usando delete	delete nombre;
Definir una propiedad dos veces	var x = {persona:'juan', persona:'juan'};
Nombres de parámetros duplicados	function saludar(persona, persona) {};
Usar eval como nombre de variable	eval = 'aprobado';
En una función si no se conoce this es el objeto global window	En una función si no se conoce this es undefined
Otras	Hay más restricciones que impone el strict mode, relacionadas con evitar el uso de sintaxis no adecuada, instrucciones poco adecuadas, prácticas inseguras y mejora de la seguridad para el usuario.

En resumen, el strict mode supone que muchos fallos que JavaScript "se tragaba", dejen de ser aceptados y aparezcan como errores no admitidos (que detendrán la ejecución del script o deberán ser capturados).

STRICT MODE Y LAS NUEVAS VERSIONES DE JAVASCRIPT

Strict mode trata de hacer más segura y fiable la programación JavaScript, y elimina algunas de las denominadas "Bad parts" o partes malas del lenguaje.

La aparición de strict mode posiblemente trata de ser una transición entre la situación y el JavaScript aceptado por navegadores antiguos y lo que serán las versiones del futuro de JavaScript, donde todo o parte de las características del strict mode pasarán a ser características intrínsecas de JavaScript.

Muchos programadores y librerías de JavaScript trabajan con strict mode, y otros muchos no lo hacen.

Programar bajo strict mode puede considerarse una buena práctica. Sin embargo, no es posible o no es aconsejable tomar scripts, páginas web o aplicaciones web y aplicarles el strict mode porque pueden dejar de funcionar en algunos navegadores, especialmente los más antiguos.

Por otro lado, usar strict mode puede inducir que el código tenga algunos comportamientos diferentes a lo esperado. También puede haber problemas si se combinan partes de código o archivos que usan strict mode con otros que no lo usan. Por ello, en caso de usar strict mode se recomienda que se hagan

pruebas en navegadores que no soportan strict mode y en navegadores que sí lo soportan, para evitar problemas de compatibilidad, es decir, que los scripts funcionen bien en algunos navegadores pero no en otros.

Usar strict mode no nos convertirá en mejores programadores, pero posiblemente nos ayudará a que no se escapen errores que podrían escaparse sin usarlo.

WITH JAVASCRIPT

Hay una cláusula o sentencia de manipulación de objetos JavaScript de la que no hemos hablado: with. Se usa de la siguiente manera:

```
with (expresión que crea un contexto de referencia) {
  sentencias donde el contexto de referencia por defecto es el indicado
}
```

Una aplicación de with podría ser aplicar propiedades a un objeto, por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
with(document.getElementById('pulsador').style) {
   backgroundColor = 'yellow';
   color = 'black';
   width = '200px';
   padding = '20px';
            fontSize = '32px';
}
 }
</script>
</head>
<body>
<div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body>
</html>
```

En JavaScript el uso de with se considera una mala práctica. Los motivos para ello son que puede dar lugar a comportamientos inesperados, problemas de rendimiento y problemas de seguridad.

Un problema potencial es que si se trata de invocar una propiedad que no existe en el contexto del with, en lugar de crearse una nueva propiedad, se crea una variable global completamente nueva e independiente. Esto puede ser difícil de detectar y dar lugar a problemas.

Ejemplo:

```
function ejemplo() {
with(document.getElementById('pulsador').style) {
   backgroundColor = 'yellow';
   color = 'black';
   width = '200px';
   padding = '20px';
   font_Size = '32px';
}
}
```

Aquí parece que font_Size aluda al tamaño de fuente del nodo con id <<pulsador>>, o al menos que sería una propiedad del objeto asociado. Sin embargo, no es así. Al llevar el guión bajo en font_Size no se reconoce la propiedad y se crea una variable global nueva.

Otro problema del with es que obliga al intérprete a buscar primero entre todas las propiedades del objeto referenciado, y si no se encuentra el nombre utilizado, entonces buscarlo en otro contexto o crear algo nuevo. Esto lleva a que trabajar con with conlleve un mal rendimiento. Especialmente en bucles dentro de with, el rendimiento o velocidad de ejecución se ve drásticamente reducido.

Por estos y otros motivos with ha pasado a considerarse no permitido en strict mode (donde obtendremos un error de tipo <<SyntaxError: strict mode code may not contain 'with' statements>>) y se considera que terminará por desaparecer (o ser replanteado de otra manera).

EJERCICIO 1

Dado este fragmento de código. Revísalo y responde a las siguientes preguntas:

```
var persona1 = {};
Object.defineProperty(persona1, "edad", { value: 42, writable: false });
persona1.edad = 19;
```

a) Explica paso a paso el significado de este código (busca información en internet si te es preciso).

b) Crea un pequeño script donde se ejecute este código y se muestre un mensaje por pantalla informando del valor de la edad. Activa la consola para comprobar si aparece algún error.

c) Crea el mismo script pero usando strict mode. Activa la consola y comprueba si aparece algún error. ¿Qué diferencias observas entre la ejecución con strict mode y sin strict mode? ¿Qué explicación le darías a estas diferencias? ¿Crees que sería positivo que este código se escribiera en strict mode o no? ¿Por qué?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

EJERCICIO 2

Crea y ejecuta un script que use with y que esté:

a) En modo normal. ¿Cuál es el código y cuál es el resultado que obtienes?

b) En strict mode. ¿Cuál es el código y cuál es el resultado que obtienes?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01190E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT HOISTING. ERRORES FRECUENTES DE PROGRAMADORES Y CONSEJOS. PROBLEMAS PRECISIÓN DECIMAL (CU01190E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº90 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

HOISTING JAVASCRIPT

La programación JavaScript tiene algunos detalles que conviene ir conociendo poco a poco, ya que en algunos momentos pueden generarnos quebraderos de cabeza si no los conocemos. Uno de estos detalles es el hoisting de las declaraciones de variables, que vamos a explicar.

ELEVACIÓN O HOISTING DE DECLARACIONES DE VARIABLES

Como hemos visto a lo largo del curso, JavaScript admite el uso directo de variables sin necesidad de declaración previa. También hemos dicho que recomendamos declarar las variables.

Cuando se declara una variable en un punto intermedio del código, dicha declaración es "elevada" (hoisted) a la parte inicial del código en el ámbito donde se encuentra dicha variable. Como ámbito más básico podemos pensar en una función. Supongamos que dicha función comienza con diversas instrucciones en las líneas iniciales y que en la línea 15 introducimos una declaración como var x;

Debido a que JavaScript realiza el izado o elevación de las declaraciones de variables, el intérprete JavaScript ejecutará el código como si dicha declaración se encontrara en la primera línea de la función en lugar de en la línea 15.

El hoisting afecta sólo a la declaración de variables. Si hacemos una declaración-inicialización, se realiza el izado sólo de la declaración, pero no de la asignación. Esto implica que la variable constará como declarada desde el comienzo del ámbito, pero el valor asignado no será conocido hasta que se llegue a la línea correspondiente. Por ejemplo var x; es izado pero var x=67; no es izado en su totalidad. En este caso se iza var x; mientras que x=67 no surtirá efecto hasta que se llegue a la línea correspondiente.

Con un ejemplo lo entenderemos mejor. Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
nombre = 'Carlos';
alert('Soy '+nombre);
var nombre;
alert('Soy yo otra vez: '+nombre);
 }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla el mensaje <<Soy Carlos>> y seguidamente el mensaje <<Soy yo otra vez: Carlos>>.

Este resultado puede resultar un poco contradictorio respecto a lo que podríamos esperar (sobre todo si hemos programado en otros lenguajes). Mucha gente piensa que al declarar var nombre; en un punto intermedio sería como si esa variable se destruyera y volviera a ser creada con un contenido <<undefined>>. Sin embargo, esto no ocurre en este caso en JavaScript porque lo que hace el intérprete es:

a) Izar la declaración de variable al comienzo de la función

b) Ejecutar el código

Por tanto lo que se ejecuta realmente después de este "preprocesamiento" es lo siguiente:

```
function ejemplo() {
var nombre;
nombre = 'Carlos';
alert('Soy '+nombre);
alert('Soy yo otra vez: '+nombre);
 }
```

Esto puede generar efectos un tanto extraños o confusos. La recomendación para evitarlos es:

a) Declarar siempre las variables antes de usarlas

b) Realizar la declaración de variables al comienzo del ámbito donde se van a usar, en este caso sería al comienzo de la función. Si JavaScript las va a izar, mejor que se vea claramente dónde se sitúan que tenerlas entremezcladas en el código.

Escribe y comprueba los resultados de este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
nombre = 'Carlos';
alert('Soy '+nombre+ ' ' +apellidos);
var nombre = 'Juan';
var apellidos = 'Fernández';
alert('Soy '+nombre+ ' ' +apellidos);
 }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla: << Soy Carlos undefined>> y a continuación << Soy Juan Fernández>>. Aquí vemos que var nombre = 'Juan'; tiene el mismo efecto que si escribiéramos nombre = 'Juan'; y que la declaración de apellidos combinada con una inicialización, no es izada completamente: sólo se iza la declaración, mientras que la asignación queda pendiente y no se ejecuta hasta llegar a la línea correspondiente.

No comprender el mecanismo de hoisting ocasiona errores en el código. Es algo que los programadores JavaScript deben conocer y saber manejar.

ERRORES FRECUENTES DE PROGRAMADORES JAVASCRIPT

Es frecuente oir ¡JavaScript no funciona!, pero JavaScript sí funciona. De hecho, es una tecnología cada vez más usada. También es cierto que es un lenguaje más rico y complejo de lo que la mayoría de la gente suele creer. La mayor parte de las ocasiones los problemas se deben a que los programadores no entienden lo que ocurre o han cometido equivocaciones.

Existen diferentes listados de errores frecuentes cometidos por programadores JavaScript (sobre todo por programadores que están empezando con JavaScript). A continuación resumimos algunos de estos errores frecuentes. Tenlos en cuenta cuando crees código JavaScript:

Equivocación frecuente	Ejemplo	Explicación y solución
Pensar que el código no funciona a pesar de estar bien programado	No se muestra un efecto que tenía que mostrarse	Muy frecuente: problemas de compatibilidad del navegador. No todos los navegadores, en especial los más antiguos, se comportan como sería de esperar. Realiza un test con otros navegadores.
Mal uso de this	Se invoca this pensando que se refiere a un objeto y se está refiriendo a window, o al revés	Revisar las entregas de este curso donde se habla de this. Es frecuente poder solucionar este error usando bind.
Mal uso del operador = dentro de if	if (numero = 20) { ... }	Esto devuelve siempre true. Error por no usar el operador correcto en comparaciones, == ó ===, normalmente por despiste.

JavaScript no crea ámbitos locales en bucles for	for (var i = 0; i < 10; i++) {
	alert(i);
	}
	alert (i);
		En otros lenguajes i es una variable local que se destruye al terminar el bucle. En JavaScript no, sigue existiendo (como si se tratara de una variable normal) y además con el valor que causa la salida del bucle, en este ejemplo 10.

Pensar que un elemento no responde sin darnos cuenta de que está mal escrito	getElementByID('myId');	Basta cambiar una minúscula por una mayúscula, o al revés, para que el código no responda. En este ejemplo tendría que ser byId en lugar de byID.
Sobreescribir una variable global dentro de una función sin querer	Cuando el código es largo y complejo, podemos repetir nombres de variables sin querer.	Restringir el uso de variables globales. Crear espacios de nombres.

Equivocación frecuente	Ejemplo	Explicación y solución
Hacer roturas de línea sin pensar que equivale a que exista un ; de cierre	var txt = '<ul id="lista1"> - Aprender - Programa ';	Un salto de línea equivale a un ; de cierre. Para concatenar la cadena usar + y delimitar los fragmentos con apóstrofes: var txt = '<ul id="lista1">' + ' - Aprender ' + ' - Programa ' + ' ';
Repetir nombres de funciones	A diferencia de otros lenguajes como Java, en JavaScript no se puede repetir el nombre de una función ni siquiera teniendo distinto número de parámetros.	Mantener un buen control de nombres.

Llamadas a funciones sin los parámetros necesarios	Llamar a obtenerDomicilio (direccion, ciudad, provincia, pais) sin pasarle el país.	Function obtenerDomicilio (direccion, ciudad, provincia, pais) {
		pais = pais || "Colombia"; // Si no se recibe pais, indicar el valor que se debe tomar
		}

Pensar que un objeto no definido contiene null	Intentar comprobar si un objeto contiene null pero el objeto no está definido	Comprobar primero si el objeto está definido y luego comprobar si contiene null.
		if(typeof(myObject) !== 'undefined' && myObject !== null) { //código
		}

No tener en cuenta algunas peculiaridades de sintaxis	No usar sintaxis JavaScript para invocar la clase de un elemento HTML ó la propiedad for.	Usar element.className para clases, element.htmlFor para atributos for.

Tratar de operar con precisión decimal	var x = 0.1;
	var y = 0.2;
	var z = x + y // no da 0.3
	if (z == 0.3) { // ¡No cumple!
		JavaScript (y muchos más lenguajes) tiene dificultades con la precisión decimal. Realizar comparaciones basadas en enteros:
		var x = 0.1;
		var y = 0.2;
		var z = x*10 + y*10
		if (z == 3)

Confundir objetos y propiedades con arrays asociativos	nombreObjeto.['nombre'] alude a una propiedad de un objeto, no a un elemento de un array.	Recordar que JavaScript no admite arrays asociativos como otros lenguajes.
Errores en la concepción del código	Crear closures indebidamente y muchos otros errores a la hora de concebir el código	Deben ir entendiéndose y corrigiéndose progresivamente mediante práctica y estudio.
Pensar que JavaScript es un mal lenguaje de programación	Pensar que JavaScript es un mal lenguaje cuando el problema es que no se conoce	Estudiar JavaScript

EJERCICIO

Dado este fragmento de código. Revísalo y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var x = 'Hola amigos'; // variable global
function ejemplo(){
 alert( x ); // esperamos el valor global
 var x;
 x = 'Saludos desde Costa Rica'; // redefinimos la variable en contexto local
 alert( x );  // esperamos el nuevo valor local
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

a) Realiza una prueba pulsando en “Probar”. ¿Qué resultados obtienes? ¿Cómo se explican esos resultados?

b) Cambia el código y declara la variable x dentro de la función al mismo tiempo que la inicializas. ¿Qué resultados obtienes? ¿Cómo se explican esos resultados?

c) De los dos casos anteriores (a y b). ¿En cuáles se produce hoisting: en el a), en el b) ó en ambos?

d) En este caso, ¿el hoisting está afectando a los resultados obtenidos? ¿Por qué? ¿Cuáles serían los resultados de ejecutar estos códigos si no existiera hoisting?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01191E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

JAVASCRIPT. REGLAS DE ESTILO (MANUALES O CONVENCIONES). CÓMO CREAR OBJETOS Y ARRAYS. EJEMPLOS Y EJERCICIO (CU01191E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº91 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

REGLAS DE ESTILO JAVASCRIPT

Los programadores JavaScript suelen atenerse a unas reglas de estilo a la hora de escribir código. Estas reglas son definidas por programadores expertos (gurús) o por empresas como Google ó Microsoft y no son de obligado cumplimiento. Tan sólo son recomendaciones que buscan que el código sea más fácil de leer y entender y siga unos estándares.

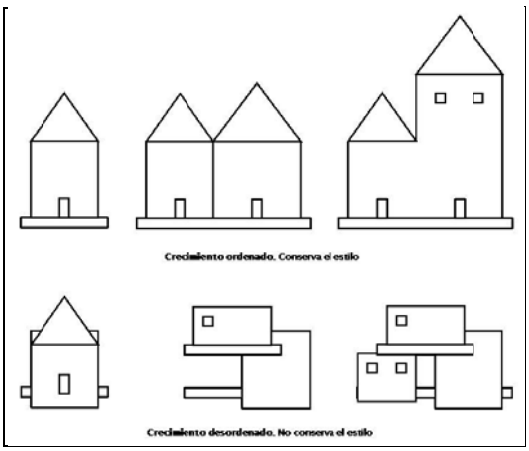
MANUALES DE ESTILO

En una empresa en la que trabajan 100 ó 1000 programadores cada programador no puede "hacer las cosas a su manera". En estas empresas es frecuente que cada programador desarrolle un fragmento de una aplicación y luego esos fragmentos hay que unirlos. Para que esa unión sea sólida y no haya problemas de integración y mantenimiento posterior, todos los programadores deben atenerse a unas mismas normas.

Lo ideal sería que en una aplicación donde han trabajado 20 programadores, un supervisor no fuera capaz de detectar qué parte del código ha hecho cada programador. Esto es difícil, porque cada programador deja su impronta, pero siguiendo unas reglas predefinidas se puede conseguir como resultado un código bien estructurado y bastante homogéneo.

Si una aplicación se ha creado siguiendo unas reglas, cuando se hagan ampliaciones o correcciones del código, se deberían seguir usando esas reglas para mantener el código homogéneo y hacerlo mantenible. Si no se mantiene un orden y un criterio de estilo, las aplicaciones terminan por hacerse demasiado complicadas y difíciles de entender.

La idea puede ser similar a la construcción de un edificio: en todas las plantas deben respetarse unas normas (altura de las puertas, pintura aplicada, dimensiones de los pasillos, etc.). Además cuando el edificio se amplíe deben seguir respetándose esas normas. Si no se hace así, el aspecto será el de "una chapuza" o una unión de cosas en lugar de un edificio.



REGLAS DE ESTILO

A continuación presentamos una lista de reglas de estilo que suelen recomendarse para la programación JavaScript. Como hemos indicado, son opcionales, pero un programador que no sigue unas reglas de estilo no suele considerarse un buen programador. Más que seguir estrictamente estas reglas que presentamos a continuación, lo importante es que cuando crees código sigas unas reglas de estilo (que pueden ser estas o similares a estas) y que no las modifiques. De esa manera crearás código homogéneo.

También ten en cuenta que las reglas aquí presentadas son una forma resumida de reglas respecto a lo que sería un manual de estilo propio de una empresa, que sería mucho más extenso. El objetivo de presentar aquí estas reglas es didáctico y para familiarizarnos con ellas, no consideres que son obligatorias ni que aquí tienes todas las reglas necesarias.

Reglas de estilo recomendadas a la hora del uso del lenguaje JavaScript:

REGLA DE ESTILO	EJEMPLO
Declarar siempre las variables usando var. No hacerlo dificulta interpretar en qué ámbito se encuentra la variable y puede ocasionar que una variable esté en ámbito global indebidamente.	var nodosDiv;

Usa siempre puntos y coma para delimitar el final de una instrucción o línea. Cuando una función funciona como una expresión, debe delimitarse con punto y coma final. En cambio, en las declaraciones de funciones no se usa punto y coma final (ver ejemplo)	var mivar1 = function() {
	  return true;
	}; // Aquí incluir punto y coma final
	
	function miFuncion1() {
	  return true;
	} // Aquí no incluimos punto y coma final

No declarar funciones dentro de bloques (condicionales, bucles…). Si se requiere crear funciones dentro de bloques, usarlas como expresiones que se asignan a una variable.	if (x) {
	  function miFuncion1() {…} // ¡incorrecto!
	}
	if (x) {
	  var mivar1 = function() {…}; // ¡correcto!
	}

Hay que tener cuidado a la hora de crear closures porque pueden generar referencias circulares que impliquen desbordamiento de memoria.	No utilizar:
	function miFuncion1(element, a, b) {
	  element.onclick = function() { /* código que usa a y b, crea el closure que mantiene una referencia a element a pesar de no usarlo, mientras que element mantiene una referencia al closure, circularidad */ };
	  }
	Para resolver esta situación se usa:
	function miFuncion1(element, a, b) {
	  element.onclick = miFun2(a, b);
	}
	function miFun2(a, b) {
	  return function() { /* código que usa a y b */ };
	  }

REGLA DE ESTILO	EJEMPLO
No usar with	Evitar el uso de esta sentencia
Limitar el uso de this	this puede tener distintos significados según el contexto. Usarlo sólo cuando sea realmente necesario.
No usar for-in para recorrer arrays.	Para recorrer arrays debe usarse un bucle for tradicional.
Crear arrays usando sintaxis de literal en lugar de new Array	Recomendado: var a = [x1, x2, x3]; No usar: var a = new Array(x1, x2, x3);
Crear objetos usando sintaxis de literal en lugar de new Object	Recomendado: var a = { }; No usar: var a = new Object ();
Crea el javascript en archivos js independientes en lugar de embeberlo en ficheros html	Esto hará reusable el código y hará más pesada la carga de los archivos html.
Usa === y !== en condicionales para realizar comparaciones preferentemente sobre == y !=	if (a === b) { ... }
Evita el uso de eval	Busca alternativas y evita usar la función eval

Incluye código previendo una llamada a una función en la que falten parámetros	function myFunction(x, y) {
	if (y === undefined) { y = 0; }
	}
	Esto se escribe más fácil y cómodo así:
	function myFunction(x, y) {
	y = y || 0;
	}

Reduce el acceso al DOM.	Si tienes que acceder varias veces a un elemento del DOM, no uses el acceso tipo getElementById repetidas veces. Usalo una sola vez y almacena el resultado en una variable. De esta manera mejoras el rendimiento.

Reglas de estilo recomendadas para la presentación de código JavaScript:

REGLA DE ESTILO	EJEMPLO
Usar la sintaxis camelCase para declarar nombres de variables y funciones (primera letra minúscula e intercalar mayúsculas).	nombreDeFuncionAsi, nombreDeVariableAsi, nombreDeMetodoAsi,
Crear los nombres de funciones y variables usando sólo 26 letras (a hasta z, sin la ñ), diez números (0 al 9) y el guión bajo _. Evitar uso de la ñ o signos como $, %, &, etc. Tampoco uses el guión medio.	var anyo = 2089;

REGLA DE ESTILO	EJEMPLO
Si una variable no va a cambiar de valor nunca se declara con todas sus letras en mayúsculas.	var NUMERODEAVOGADRO;
Los nombres de ficheros se escriben completamente en minúsculas. En el caso de ficheros html, se usa la extensión html en lugar de htm.	nombredeficherotodominusculas.js otroNombre.html

Incluye las llaves en la misma línea de aquello que abren (no pongas las llaves por ejemplo debajo de una condición de un if, sino al lado)	if (condicion) {
	// ...
	} else {
	// ...
	}

Usa siempre indentado para aquello que está dentro de un bloque. Por ejemplo, el código dentro de un bucle for estará desplazado hacia la derecha. Usa siempre el mismo indentado.	for (var i=0; i<10; i++) {
	//aquí codigo
	}

Las líneas no se deben alargar tanto como para no ser visibles. Se usa la referencia de 80 caracteres. La línea no debe tener más de 80 caracteres. Si se excede esta longitud, la línea deberá dividirse.	esto.aquello.alli.hacer = function(
	unArgumentoParaLaFuncionDeNombre1,
	unArgumentoParaLaFuncionDeNombre2,
	unArgumentoParaLaFuncionDeNombre3,
	unArgumentoParaLaFuncionDeNombre4) {
	// código
	};

Usa comillas simples en lugar de comillas dobles	var nombre = 'carlos';

Separa las sentencias en distintas líneas. No escribas una detrás de otra separadas por ;	var x=1; var y=2; // No
	var x=1;
	var y=2;

Usa espacios separadores: esto hace más fácil de leer el código	while(x<3&&y<5&&z==10){llamarA();} //No
	while (x < 3 && y < 5 && z == 10) {
	llamarA();
	} //Sí

El código debe estar correctamente comentado. Los bloques /* ... */ se usarán para documentación formal (explicación de funciones, parámetros que intervienen, etc. y los comentarios en línea // para aclaraciones puntuales.	Para documentar proyectos extensos se recomienda seguir unas normas de comentarios y usar herramientas específicas que crean la documentación de forma automática, como JSDoc.
Respetar el estilo del código en edición	Si nos encargan realizar unos cambios en un código que tiene un estilo, respetar el estilo de dicho código aunque no coincida con el que nosotros usemos habitualmente. Es importante que dentro de un código se mantenga siempre el mismo criterio.

EJERCICIO

Un programador ha creado este código y nos han pedido que lo mejoremos. Revísalo y responde a las siguientes cuestiones:

```
<HTML><HEAD><TITLE>JavaScript Index</TITLE><script Language="JavaScript">
function goback(){alert("Good Bye!");history.go(-1);}
function getthedate() {Todays = new Date();
TheDate = "" + (Todays.getMonth()+ 1) +" / "+ Todays.getDate() + " / " +
Todays.getYear()
document.clock.thedate.value = TheDate;
}

var timerID = null;
var timerRunning = false;
function stopclock (){
        if(timerRunning)
                clearTimeout(timerID);
        timerRunning = false;
}

function startclock () { stopclock(); getthedate()
        showtime();
}

function showtime () {
        var now = new Date(); var hours = now.getHours(); var minutes = now.getMinutes();
        var seconds = now.getSeconds()
        var timeValue = "" + ((hours >12) ? hours -12 :hours)
        timeValue += ((minutes < 10) ? ":0" : ":") + minutes
        timeValue += ((seconds < 10) ? ":0" : ":") + seconds
        timeValue += (hours >= 12) ? " P.M." : " A.M."
        document.clock.face.value = timeValue;
        timerID = setTimeout("showtime()",1000); timerRunning = true;
}
</script>
</HEAD>
<BODY bgcolor="#00FFFF" onLoad="startclock()">
<CENTER><h2>Esto es un reloj hecho con JavaScript</h2>
<table border><tr>
  <td><form name="clock" onSubmit="0"></td>
</tr>
<tr>
  <td colspan=2>Hoy es: <input type="text" name="thedate" size=12  value=""></td>
  <td colspan=2>La hora es: <input type="text" name="face" size=12
      value=""></td></form>
</tr>
</table>
</CENTER>
<hr>
<center>
<h3>
[<a href="http://aprenderaprogramar.com">Volver</a>]
</h3></center></BODY></HTML>
```

a) Reescribe el código HTML que presenta distintas deficiencias y no se ajusta a las normas de estilo habituales.

b) Reescribe el código JavaScript para que cumpla con las normas de estilo que hemos estudiado.

c) Corrige el código del reloj para que se vea una mejor presentación y funcione correctamente.

d) Incluye comentarios en el código indicando qué es lo que hacen las diferentes partes del código JavaScript.

e) Como resultado de este ejercicio debes presentar el código con todos los cambios antes mencionados.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01192E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

GUÍA DE ESTILO JAVASCRIPT PARA COMENTARIOS DE PROYECTOS. JSDOC. @CONSTRUCTOR, @DEPRECATED, ETC. EJEMPLOS (CU01192E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº92 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

GUÍA DE ESTILO PARA COMENTARIOS

Los programadores JavaScript suelen atenerse a unas reglas de estilo a la hora de escribir código. Esto no sólo es válido para el código en sí, sino también para los comentarios. En proyectos de gran extensión pueden existir miles de líneas de código JavaScript. Este código debe estar bien comentado para facilitar su comprensión y mantenimiento.

MANUALES DE ESTILO

En una empresa en la que trabajan 100 ó 1000 programadores cada programador no puede "hacer las cosas a su manera". En estas empresas es frecuente que cada programador desarrolle un fragmento de una aplicación y luego esos fragmentos hay que unirlos. Para que esa unión sea sólida y no haya problemas de integración y mantenimiento posterior, todos los programadores deben atenerse a unas mismas normas.

Lo ideal sería que en una aplicación donde han trabajado 20 programadores, un supervisor no fuera capaz de detectar qué parte del código ha hecho cada programador. Esto es difícil, porque cada programador deja su impronta, pero siguiendo unas reglas predefinidas se puede conseguir como resultado un código bien estructurado y bastante homogéneo.

Ya hemos hablado de reglas de estilo a la hora de crear código en relación formas de uso preferentes del lenguaje y en relación a cuestiones organizativas relacionadas con el código.

Vamos a centarnos ahora en los comentarios dentro del código JavaScript. No hay unas reglas obligatorias que cumplir en relación a los comentarios. Cada empresa puede usar diferentes estilos a la hora de comentar el código, si bien es cierto que suele hacerse de forma similar cuando hablamos de programación profesional.

Nosotros vamos a presentar aquí unas reglas de estilo para comentarios que se corresponden con las que utilizan grandes empresas, pero ten en cuenta que ni son obligatorias ni todas las empresas siguen los mismos criterios.

REGLAS DE ESTILO PARA COMENTARIOS

A continuación presentamos una lista de reglas de estilo que suelen recomendarse para la programación JavaScript. Como hemos indicado, son opcionales, pero un programador que no sigue unas reglas de estilo no suele considerarse un buen programador. Más que seguir estrictamente estas reglas que presentamos a continuación, lo importante es que cuando crees código sigas unas reglas de estilo (que pueden ser estas o similares a estas) y que no las modifiques. De esa manera crearás código homogéneo.

También ten en cuenta que las reglas aquí presentadas son una forma resumida de reglas respecto a lo que sería un manual de estilo propio de una empresa, que sería mucho más extenso. El objetivo de presentar aquí estas reglas es didáctico y para familiarizarnos con ellas, no consideres que son obligatorias ni que aquí tienes todas las reglas necesarias.

Las reglas aquí presentadas siguen el estándar JSDoc. Seguir un estándar tiene ventajas como poder generar de forma automática documentación de proyectos usando el software adecuado.

Regla 1. El mejor comentario es el implícito o auto-documentador

Considera que estás revisando un código y que en una función encuentras esto:

```
function f33 (a, b) { ... }
```

¿Te haces una idea de lo que hace esta función? ¿Te haces una idea de qué tipo de parámetros recibe la función? Difícilmente nos podremos hacer una idea si los nombres de variables, funciones, parámetros, etc. son cortos y no descriptivos.

Sería mucho mejor esto:

```
function getFullName (name, surname) { ... }
```

Fíjate que estamos usando nombres descriptivos. En este caso hemos usado el inglés para que tengas en cuenta que el inglés es el lenguaje más universalmente utilizado en programación. No solo lo utilizan quienes hablan inglés, sino que lo utilizan muchas empresas aunque sean de habla hispana. Si prefieres utilizar el español puedes escribirlo así:

```
function obtenerNombreCompleto (nombre, apellidos) { ... }
```

Este código, aunque no hemos comentado nada, podemos decir que está autodocumentado, porque se entiende (aproximadamente) lo que hace simplemente leyéndolo. Esto es un buen estilo de programación.

Regla 2. Usa un estilo de comentarios y no lo cambies, pero respeta el existente

Si estamos trabajando en un nuevo proyecto usaremos un estilo para comentarios que debemos mantener en todo el proyecto. Sin embargo, si estamos revisando el código desarrollado por otras personas y observamos que siguen un estilo de comentarios diferente al que nosotros estemos habituados a usar, deberemos respetarlo (para no afectar a la homogeneidad del código).

El estilo de comentarios que vamos a proponer aquí es un estilo similar al que se usa en otros lenguajes de programación como C++ y Java.

Regla 3. Usa comentarios multilínea en cabeceras de funciones y en línea en otros casos

Para comentar lo que hace una función usaremos comentarios de este tipo:

```
/**
 * El comentario comienza con una barra y dos asteriscos.
* Cada nueva línea lleva un asterisco al comienzo.
 * @param {string} nombre indica que una función recibe un parámetro de tipo string y que
      * el nombre del parámetro es nombre.
 * @descriptor Cada descriptor que añadamos irá en una línea independiente.
 */
```

Si una línea es demasiado larga la dividimos en varias líneas. La primera estará sin indentar y el resto indentada (desplazada hacia dentro) respecto de la primera.

Los comentarios puntuales en línea se harán con //. Por ejemplo:

var tmpPers; //Variable temporal que almacenará un objeto Persona

Regla 4. Usa un comentario al principio de un archivo js para describir su contenido

En un proyecto podemos tener decenas de archivos js correspondientes a código JavaScript. Al principio de cada archivo deberíamos incluir un comentario descriptivo de qué contiene dicho archivo que comenzará con la etiqueta @fileoverview. Por ejemplo:

```
/**
 * @fileoverview Menú aprMenu, desplegable con efecto expansión suavizado
 *
 * @version          2.2
 *
 * @author           César Krall <cesarkrall@aprenderaprogramar.com>
 * @copyright        aprenderaprogramar.com
 *
 * History
 * v2.2 – Se mejoró el efecto de expansión de los submenús dándole efecto aceleración
 * v2.0 – Se evitó que quedaran supersupuestos textos de submenús
 * v1.1 – Se mejoró la compatibilidad con navegadores Opera
 * ----
 * La primera versión de aprMenu fue escrita por Karl Monitrix
 */
```

Regla 5. Documenta las funciones incluyendo los parámetros que usa y lo que devuelve

Un comentario típico para describir una función incluirá una descripción de lo que hace la función, la descripción de sus parámetros, y la descripción de lo que devuelve la función.

La descripción de un parámetro se hace comenzando con @param

La descripción de lo que devuelve la función se hace comenzando con @return. Si la función no devuelve nada se omitirá esta etiqueta.

Ejemplo:

```
/**
 * Verifies if the string is in a valid email format
 * @param  {string}
 * @return  {boolean}
 */
```

Regla 6. Usa comentarios para constructores y para documentar las propiedades

Si estamos definiendo un tipo de objeto usaremos una declaración @constructor para indicar que estamos definiendo el constructor para un tipo de objeto. Además, las propiedades serán comentadas para reflejar su significado. La etiqueta @type se usará para describir qué tipo de dato corresponde a la propiedad.

Ejemplo:

```
/** @constructor */
project.MiDefinicionDeTipoDeObjeto = function() {
  /**
   * Propiedad que indica el número máximo de colores permitidos.
   * @type {number}
   */
  this.maxNumeroColores = 4;
}
```

Regla 7. Usa etiquetas normalizadas

Cuando queramos incluir una descripción de un elemento usaremos etiquetas normalizadas. Por ejemplo una etiqueta normalizada para describir al autor de un código es @author. En cambio no es normalizada <<@autor>> ni <<@Author>> ni <<@authors>>.

A continuación indicamos la lista de etiquetas normalizadas recomendadas:

Etiqueta	Descripción	Ejemplo
@author	Indica el nombre del autor del código	@author César Krall
@const {type}	Indica que una propiedad o variable serán constantes y su tipo	/** * El nombre de la empresa. * @const {string} */
@constructor	Indica que se define un tipo de objeto	/** * Representa un círculo. * @constructor */ function Circulo() { ... }
@deprecated	Indica que una función, método o propiedad no deberían usarse por ser obsoletas. Debe indicarse cuál es la función, método o propiedad que debe ser usada en su lugar.	@deprecated Usar getNombre().

@enum	Indica que se trata de una enumeración	/**
		* Enum para tres estados posibles.
		* @enum {number}
		*/
		project.TresEstado = {
		SI: 1,
		NO: -1,
		DESCONOCIDO: 0
		};

@extends	Usado con @constructor para indicar que un tipo de objeto hereda de otro tipo de objeto	/** * Una lista de objetos persona. * @constructor * @extends apr.bc.BasicList */
@fileoverview	Comentario para describir los contenidos de un fichero	Ver el ejemplo de la regla 4 expuesto anteriormente.
@interface	Usado para indicar que una función define una interface	/** * Una forma. * @interface */ function Forma() {}; Forma.prototype.dibujar = function() {};

@implements	Usado con @constructor para indicar que un tipo de objeto implementa una interface	/**
		* @constructor
		* @implements {Forma}
		*/
		function Square() {};
		Square.prototype.dibujar = function() {
		...
		};

Etiqueta	Descripción	Ejemplo
@license	Indica los términos de licencia	/* Copyright aprenderaprogramar.com, 2002-2005 * This script is developed by aprenderaprogramar. Visit us at www.aprenderaprogramar.com. * This script is distributed under the GNU Lesser General Public License. * Read the entire license text here: http://www.gnu.org/licenses/lgpl.html */
@override	Indica que un método o propiedad de una subclase sobreescribe el método de la superclase. Si no se indica otra cosa, los comentarios y documentación serán los mismos que los de la superclase.	/** * @return {number} Sueldo. * @override */
@param {type}	Indica un parámetro para un método, función o constructor y el tipo de dato que es.	* @param {string}
@return {type}	Indica qué devuelve un método o función y su tipo.	* @return {boolean}
@see	Indica una referencia a documentación más amplia	@see http://aprenderaprogramar.com
@supported	Indica navegadores para los que se conoce que aceptan el código	@supported Testado en IE10+ y en FF26+
@type {Type}	Identifica el tipo de una variable, propiedad o expresión.	@type {number}
Otros	Existen más etiquetas que no vamos a estudiar ahora	Por ejemplo @code, @define, @dict, @export, @expose, @externs, @inheritDoc, @lends, @preserve, @noalias, @nocompile, @nosideeffects, @private, @protected, @public, @struct, @suppress, @template, @this, @typedef y otros)

GENERADORES DE DOCUMENTACIÓN AUTOMÁTICOS

Existen generadores automáticos de documentación de proyectos JavaScript. Estos generadores lo que hacen básicamente es extraer los comentarios en el código y transformarlos en un documento (por

ejemplo un documento HTML) organizado donde se pueden leer todos los detalles del código con una presentación adecuada.

En este curso no vamos a entrar a estudiar esta posibilidad (que sí es habitual usar en empresas), aunque si lo deseas puedes hacer pruebas por tu cuenta.

Cada empresa o equipo de desarrolladores puede seguir distintos estándares o usar distintos generadores de documentación, por ello deberás atenerte en cada empresa a la herramienta que se esté usando para la documentación.

Las reglas de documentación que hemos expuesto están adaptadas al estándar JSDoc (http://usejsdoc.org/). Puedes hacer pruebas usando este estándar si lo deseas.

EJERCICIO

Un programador ha creado este código y nos han pedido que a) Lo comentemos de forma adecuada. b) Lo indentemos y organicemos de forma adecuada. Revísalo, coméntalo y organízalo conforme a los estándares que hemos explicado, incluyendo comentarios de cabecera de función y comentarios en línea para describir los aspectos más relevantes:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Cursos aprende a programar</title><meta charset="utf-8">
<meta name="description" content="Aprende a programar con cursos reconocidos por su calidad didáctica: HTML, CSS,
JavaScript, PHP, Java, Visual Basic, Joomla, pseudocodigo, diagramas de flujo, algoritmia y más.">
<meta name="keywords" content="HTML, CSS, JavaScript, Java, Visual Basic, Joomla, PHP, pseudocodigo, diagramas de
flujo, cursos, tutoriales">
<style type="text/css"> *{font-family: verdana, sans-serif;} #principal{text-align:center;width:95%; margin: 0 auto;}
.tituloCurso {color: white; float:left; padding: 36px 44px; font-size: 2.65em; font-style:bold; text-decoration:none;}
a:hover{color:orange !important;}</style>
<script type="text/javascript">
function ejemplo(){
var nodosTituloCurso = document.querySelectorAll(".tituloCurso");
var contador = 0;
var nodosCambiados = new Array();
var tocaCambiar = true;
setTimeout(function() {ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar)}, 1500);
}
function ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar) {
        if (contador<nodosTituloCurso.length){
        var indice = Math.floor((Math.random() * (nodosTituloCurso.length)));
        if (nodosCambiados.length!=0) {
                for (var i=0; i<nodosCambiados.length; i++) {
                        if(nodosCambiados[i]==indice) {tocaCambiar = false;}
                }
        }
        if (tocaCambiar==true) {
        cambiarNodo(nodosTituloCurso[indice]);
        nodosCambiados.push(indice);
        contador = contador+1;
        setTimeout(function() {ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar)}, 1500);
        } else {tocaCambiar=true; ejemploAccion(nodosTituloCurso, nodosCambiados, contador, tocaCambiar);}
        }
```

```
		else { //Caso base fin de la recursión
			document.body.style.backgroundColor='black';
			document.getElementById('principal').style.color='white';
			for (var i=0; i<nodosTituloCurso.length; i++) {
				nodosTituloCurso[i].style.color='yellow';
			}
		}
}

function cambiarNodo(elNodo){elNodo.style.backgroundColor = 'black';}
</script>
</head>
<body onload="ejemplo()">
<div id="principal"><h1>Cursos de programación</h1>
<h3>Reconocidos por su calidad didáctica</h3>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=28&Itemid=59"> Fundamentos</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188"> Java</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=69&Itemid=192"> HTML</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=75&Itemid=203"> CSS</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206"> JavaScript</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=70&Itemid=193"> PHP</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=38&Itemid=152"> Joomla</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=37&Itemid=61"> Visual Basic</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=28&Itemid=59"> Pseudocódigo</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=section&layout=blog&id=7&Itemid=26">
Libros/ebooks</a></div>
<div ><a class="tituloCurso"
href="http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=64&Itemid=87">Cursos tutorizados</a></div>
</div>
</body>
</html>
```

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01193E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

¿QUÉ ES Y PARA QUÉ SIRVE AJAX? JAVASCRIPT ASÍNCRONO, XML Y JSON . VENTAJAS E INCONVENIENTES DE AJAX. XMLHTTPREQUEST . EJEMPLO. (CU01193E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº93 del Tutorial básico "JavaScript desde cero".

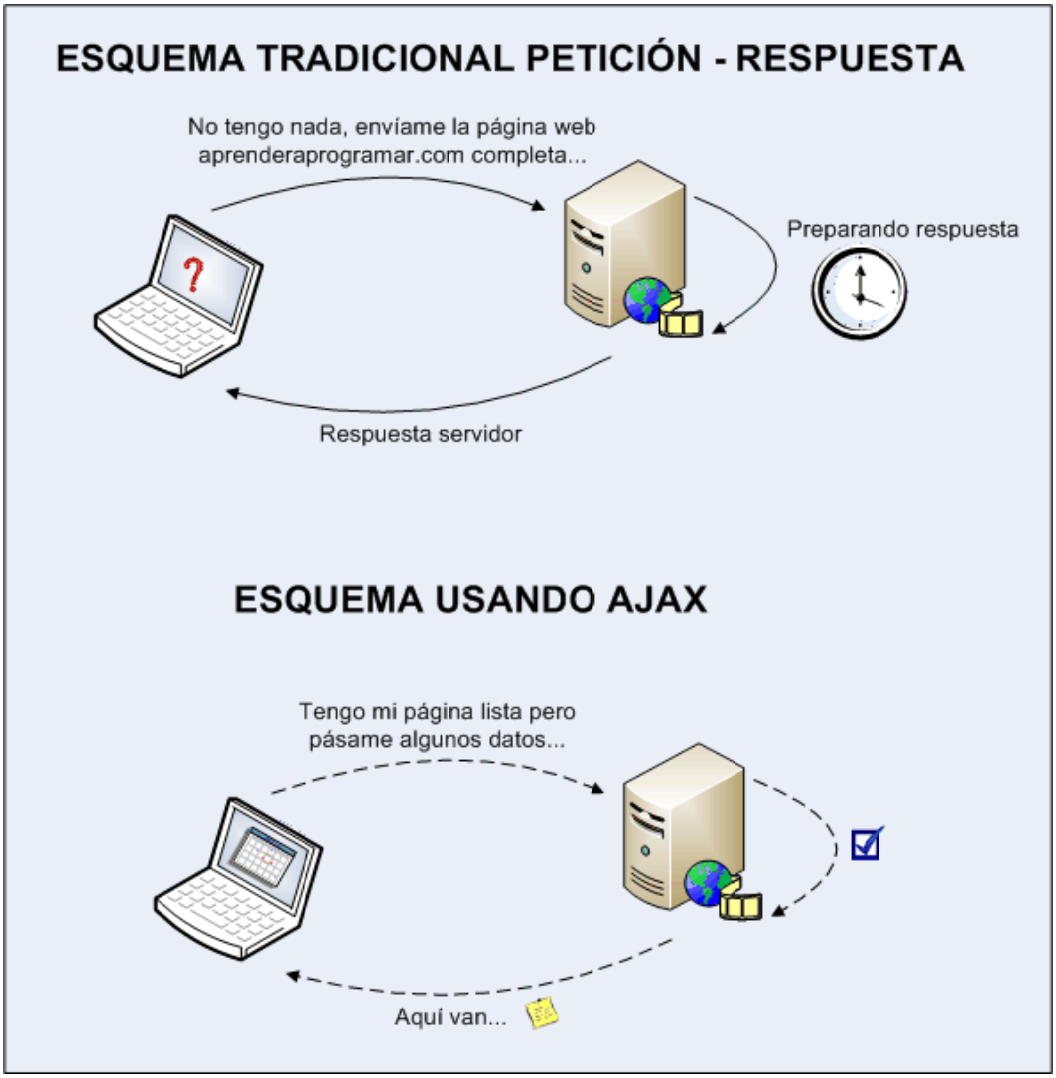
Autor: César Krall

LADO DEL CLIENTE... ¿Y MÁS ALLÁ?

Hasta ahora nos hemos centrado en JavaScript del lado del cliente. Pero las aplicaciones web o apps para smartphones tienen que conectar con servidores (donde normalmente residen los datos en bases de datos). Las peticiones a servidores se suelen hacer con lenguajes como PHP y el inconveniente que tienen son que ralentizan las páginas web.

AJAX ¿QUÉ ES Y PARA QUÉ SIRVE?

Veamos un esquema para comprender mejor la idea que hay detrás del uso de Ajax. Este esquema lo iremos comentando y comprendiendo a medida que avancemos en la explicación.



Realizar peticiones al servidor y esperar respuesta puede consumir tiempo (el tiempo necesario para recargar una página completa). Para agilizar los desarrollos web surgió Ajax (inicialmente Asynchronous JavaScript And XML, aunque hoy día ya no es una tecnología ligada a XML con lo cual no pueden asociarse las siglas a estos términos), una tecnología que busca evitar las demoras propias de las peticiones y respuestas del servidor mediante la transmisión de datos en segundo plano usando un protocolo específicamente diseñado para la transmisión rápida de pequeños paquetes de datos.

Con Ajax, se hace posible realizar peticiones al servidor y obtener respuesta de este en segundo plano (sin necesidad de recargar la página web completa) y usar esos datos para, a través de JavaScript, modificar los contenidos de la página creando efectos dinámicos y rápidos.

En el esquema anterior vemos las ideas en torno a Ajax de forma gráfica. En la parte superior hemos representado lo que sería un esquema de comunicación tradicional: el cliente solicita una página web completa al servidor. El servidor recibe la petición, se toma su tiempo para preparar la respuesta y la envía. El resultado, una pequeña demora debido al tiempo que tarda en llegar la petición al servidor, el tiempo que éste tarda en preparar la respuesta, y el tiempo que tarda en llegar la respuesta más recargarse en el navegador.

En la parte inferior vemos lo que sería un esquema de comunicación usando Ajax: el cliente tiene una página web cargada (puede ser una página web completa, o sólo el esqueleto de una página web). El cliente sigue trabajando y en segundo plano (de ahí que hayamos dibujado con líneas punteadas las comunicaciones) le dice al servidor que le envíe un paquete de datos que le hacen falta. El servidor procesa la petición. Ahora la respuesta es mucho más rápida: no tiene que elaborar una página web completa, sino sólo preparar un paquete de datos. Por tanto el tiempo de respuesta es más rápido. El servidor envía el paquete de datos al cliente y el cliente los usa para cambiar los contenidos que se estaban mostrando en la página web.

VENTAJAS E INCONVENIENTES DE AJAX

Las ventajas que proporciona Ajax son varias:

a) No es necesario recargar y redibujar la página web completa, con lo que todo es más rápido.

b) El usuario no percibe que haya demoras: está trabajando y al ser las comunicaciones en segundo plano no hay interrupciones.

c) Los pasos que antes podía ser necesario dar cargando varias páginas web pueden quedar condensados en una sola página que va cambiando gracias a Ajax y a la información recibida del servidor.

Como todo en la vida, Ajax también tiene inconvenientes:

a) El usuario puede perder la capacidad para hacer cosas que hacía con webs tradicionales puesto que no hay cambio de página web. Por ejemplo usar los botones de avance y retroceso del navegador o añadir una página a favoritos puede dejar de ser posible. Esto en algunos casos no es deseable.

b) El desarrollo de aplicaciones web se puede volver más complejo. Supongamos que antes tuviéramos un proceso en el que avanzábamos a través de varias páginas web como 1, 2, 3. De este modo la organización resulta sencilla. Si condensamos todo en una sola página web: 1, escribir y depurar el código puede volverse más complicado. En sitios complejos, puede ser muy difícil depurar errores.

c) Existen problemas y restricciones de seguridad relacionados con el uso de Ajax. Hay que tener en cuenta que por motivos de seguridad no todos los procesos se pueden realizar del lado del cliente (que por su propia naturaleza es "manipulable"). También existen restricciones de seguridad para impedir la carga de contenidos mediante Ajax desde sitios de terceras partes.

d) La indexación para los motores de búsqueda se ve dificultada, con lo cual nuestros sitios web pueden perder visibilidad en los buscadores. No es lo mismo un contenido "constante" o aproximadamente estático, fácilmente rastreable para un buscador, que un contenido "cambiante" en función de la ejecución de JavaScript, difícilmente rastreable para un buscador.

¿MEJOR USAR O NO USAR AJAX?

Como todo, hay que usar las cosas en su justa medida. Ajax bien usado puede ser muy útil para una página web. Ajax mal usado puede ser un desastre para una página web.

¿ES AJAX UN LENGUAJE DE PROGRAMACIÓN?

No, Ajax es un conjunto de técnicas que se usan para lograr un objetivo y se basa en lenguajes ya existentes como JavaScript.

Podríamos dar esta definición de Ajax: "Ajax es un conjunto de métodos y técnicas que permiten intercambiar datos con un servidor y actualizar partes de páginas web sin necesidad de recargar la página completamente".

Aunque Ajax se pensó inicialmente para transferir datos en un solo formato (XML), actualmente Ajax permite la transmisión de datos en múltiples formatos: XML, JSON, EBML, texto plano, HTML, etc.

¿QUÉ ES XML Y QUÉ ES JSON?

XML (extensible markup language) es un lenguaje de etiquetas que se usa para almacenar y enviar información. No vamos a estudiar XML, simplemente pondremos un ejemplo para hacernos una idea de qué son datos en formato XML. XML se usa para múltiples aplicaciones.

JSON (JavaScript Object Notation) es un formato para el intercambio de datos que se usa para almacenar y enviar información, basado en la notación literal de objetos de JavaScript. No vamos a

estudiar JSON, simplemente pondremos un ejemplo para hacernos una idea de qué son datos en formato JSON. JSON se usa como alternativa al XML en AJAX.

Ejemplo datos en formato XML	Ejemplo datos en formato JSON

<Curso>
<Titulo>JavaScript desde cero</Titulo>
<Autores>
<Nombre>César Krall</Nombre>
<Nombre>Bill Clinton</Nombre>
</Autores>
<WebPublicacion>
<Url>aprenderaprogramar.com</Url>
<PageRank>10</PageRank>
</WebPublicacion>
</Curso>
	{
	"Curso": {
	"Titulo": "JavaScript desde cero",
	"Autores": {
	"Nombre": [
	"César Krall",
	"Bill Clinton"
	]
	},
	"WebPublicacion": {
	"Url": "aprenderaprogramar.com",
	"PageRank": "10"
	}
	}
	}

Esto nos sirve para hacernos una idea de qué es un paquete de datos que se intercambia entre el cliente y el servidor a través de Ajax.

Como curiosidad, tener en cuenta que el término XHTML hacer referencia a la combinación de HTML con XML en el sentido de escribir HTML siguiendo las reglas de sintaxis propias de XML. Bajo XHTML por ejemplo, no pueden existir etiquetas que no se cierren, ya que XML impone la obligación de que todas las etiquetas se cierren.

UN EJEMPLO SIMPLE DE USO DE AJAX

En este curso nos centramos en JavaScript del lado del cliente y no vamos a estudiar Ajax. Hemos hecho una breve introducción a Ajax porque es una tecnología interesante para aquellas personas que tras terminar el curso quieran seguir profundizando en programación web y por ello hemos considerado conveniente hablar brevemente sobre Ajax. El ejercicio-ejemplo que se plantea es un ejercicio opcional, por tanto puedes saltarlo y pasar a la siguiente entrega del curso.

EJERCICIO OPCIONAL

Este ejercicio no forma parte del curso, por eso se califica como opcional. El ejercicio consiste en ver un breve ejemplo de uso de Ajax para realizar sugerencias de búsqueda (algo parecido a lo que hacen los buscadores cuando muestran sugerencias una vez hemos empezado a introducir una palabra).

Para desarrollar este ejercicio necesitas tener conocimientos básicos de PHP y disponer de un servidor (puede ser un servidor remoto o un servidor local como WAMP ó XAMPP o similar).

Escribe este código y guárdalo **en el servidor** con el nombre de archivo pruebaAjax.html:

```
<!DOCTYPE html><html><head><title>Cursos aprende a programar</title><meta charset="utf-8">
<script>
function mostrarSugerencia(str) {
var xmlhttp;
if (str.length==0) { document.getElementById("txtSugerencia").innerHTML=""; return; }
xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {
    document.getElementById("txtSugerencia").innerHTML=xmlhttp.responseText;
    }
}
xmlhttp.open("GET","listadoDatos.php?q="+str,true);
xmlhttp.send();
}
</script></head>
<body><h3>Empieza a escribir un nombre en la casilla para obtener una sugerencia:</h3>
<form action="">
Nombre: <input type="text" id="txt1" onkeyup="mostrarSugerencia(this.value)" />
</form>
<p>Sugerencias: <span id="txtSugerencia"></span></p>
</body></html>
```

Escribe este código y guárdalo **en el servidor** con el nombre de archivo listadoDatos.php:

```
<?php // Rellenamos un array con nombres
$a[]="Ana"; $a[]="Belinda"; $a[]="Cinderella"; $a[]="Diana"; $a[]="Eva"; $a[]="Fiona"; $a[]="Gabriela"; $a[]="Hilda";
$a[]="Idaira"; $a[]="Johanna"; $a[]="Kitty"; $a[]="Linda"; $a[]="Nina"; $a[]="Ofelia"; $a[]="Petunia"; $a[]="Amanda";
$a[]="Raquel"; $a[]="Cindy"; $a[]="Doris"; $a[]="Eve"; $a[]="Evita"; $a[]="Sunniva"; $a[]="Tania"; $a[]="Ursula";
$a[]="Violeta"; $a[]="Liza"; $a[]="Elizabeth"; $a[]="Ellen"; $a[]="Walda"; $a[]="Vicky";

// Rescatamos el parámetro q que nos llega mediante la url que invoca xmlhttp
$q=$_REQUEST["q"]; $hint="";

// Buscamos la coincidencia en el primer caracter entre nombres del array con el primer caracter de q
if ($q !== ""){
        $q=strtolower($q); $len=strlen($q);
  foreach($a as $name) {
                if (stristr($q, substr($name,0,$len))) {
                        if ($hint==="") { $hint=$name; }
                        else { $hint .= ", $name"; }
    }
  }
 }
// Devolvemos "no hay sugerencias" si no se encuentran coincidencias
// o los datos en $hint (nombres separados por comas) si se encontraron coincidencias
echo $hint==="" ? "No hay sugerencias" : $hint;
?>
```

Accede al archivo html invocando la ruta correcta (que dependerá de la ruta dentro de la que esté alojado en el servidor. La ruta será por ejemplo http://aprenderaprogramar.com/pruebaAjax.html).

Fíjate en que lo interesante de este ejemplo es que se recuperan datos que se encuentran en el servidor.

Responde a estas preguntas:

a) ¿Qué sugerencias se muestran si escribes la letra L en el cuadro de texto donde se pide el nombre?

b) ¿Qué función JavaScript se dispara cada vez que se pulsa una tecla?

c) ¿Se usa XML en este código?

d) ¿Se usa JSON en este código?

e) Busca información en internet sobre las instrucciones JavaScript / Ajax que se usan en el código y explica cada una de ellas.

f) Reescribe el código para que todo el proceso tenga lugar del lado del cliente. Es decir, traslada los datos al lado del cliente y hazlo todo con código JavaScript sin necesidad de utilizar Ajax.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01194E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

PROGRAMACIÓN WEB-APP JAVASCRIPT. LIBRERÍAS, FRAMEWORKS. JQUERY, ANGULARJS. VENTAJAS. DIFERENCIAS. (CU01194E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº94 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

LIBRERÍAS Y FRAMEWORKS JAVASCRIPT

Un programador que no conoce las bases de un lenguaje tiene difícil crear buenas aplicaciones usando ese lenguaje. Pero conocer las bases no significa que todo deba ser programado por nosotros. Cuando se trabaja a nivel profesional, entra en juego el concepto de productividad. El uso de librerías y frameworks es una forma de tener una buena productividad como programadores y de eso vamos a hablar a continuación.

PRODUCTIVIDAD COMO PROGRAMADORES

Una frase muy utilizada en los ambientes de programación es: “no trates de inventar la rueda”. Esta frase alude a que existen numerosas funcionalidades que son usadas con frecuencia por los programadores, están bien estudiadas y resueltas (posiblemente por expertos con un mayor nivel de conocimiento que nosotros). Dichas soluciones están disponibles a través de internet y por tanto es más inteligente y más productivo utilizar algo que funciona bien y está ya desarrollado que tratar de crearlo nosotros desde cero.

Veamos un ejemplo de una situación que se nos puede presentar como programadores: supongamos que desarrollamos una aplicación web para una asociación que tiene un salón de actos que puede reservarse por días. Queremos incluir un calendario en nuestra página web. Crear un calendario con un buen diseño y una buena funcionalidad (que muestre al abrirse la fecha actual, que permita avanzar o retroceder meses o años, etc.) puede requerir bastantes horas de trabajo, posiblemente días. Ahora bien, ¿es este un problema frecuente dentro de los trabajos de programación? Obviamente sí, muchas aplicaciones web trabajan con calendarios para permitir seleccionar fechas: por ejemplo aplicaciones para realizar reservas o compras en hoteles, restaurantes, etc. o para pedir cita con el médico o con una esteticista.

Esta tarea por ser muy común está bien estudiada y resuelta por la comunidad de programadores, por tanto no tiene sentido que tratemos de inventar la rueda construyéndonos nosotros mismos nuestro propio calendario (a no ser que tratemos de hacer algo innovador y exclusivo). Para resolver esta tarea lo más lógico será recurrir a código disponible a través de librerías JavaScript.

¿QUÉ ES UNA LIBRERÍA JAVASCRIPT? ¿QUÉ ES UN FRAMEWORK JAVASCRIPT? ¿QUÉ ES UNA API?

Entre los recursos que facilitan el trabajo a los programaodres tenemos librerías, frameworks y apis. Estos términos muchas veces se entremezclan entre sí, ya que no existe un límite o definición clara para cada uno de ellos. Nosotros vamos a dar aquí unas definiciones, pero ten en cuenta que son sólo algo orientativo.

Librería: conjunto de código que contiene funciones que permiten realizar tareas que nos facilitan la programación. Por ejemplo, si tenemos que calcular el factorial de un número y tenemos una librería

que nos ofrece dicha función, simplemente tendremos que llamar calcularFactorial(n) para obtener el factorial de dicho número, sin necesidad de escribir el código que resuelve dicha tarea (ya que ha sido escrito por otra persona). Una librería muy conocida JavaScript es Jquery.

Framework: un framework ("marco de trabajo") puede verse como una gran librería o conjunto de librerías donde además de facilitársenos funciones para su uso, se suele disponer de una sintaxis o metalenguaje específico del framework y una forma de organización del código específica para dicho framework. Por tanto para usar un framework no basta con conocer el nombre de una función que queramos usar: hay que conocer qué sintaxis utiliza el framework, qué obligaciones impone (a la hora de organizar el código, archivos) y su lógica o filosofía de trabajo. El uso de frameworks se ha extendido debido a la gran complejidad de las aplicaciones que se desarrollan hoy en día: los frameworks facilitan su organización y mantenimiento. Un framework muy conocido JavaScript es AngularJS.

API: api (application programming interface o interface de programación para aplicaciones) suele aludir a la especificación de librerías amplias con funciones y métodos (signaturas) utilizadas en torno a una tecnología, usualmente por toda la comunidad de programadores, aunque no siempre. El código de implementación (es decir, el detalle de la programación) suele estar oculto o no disponible, pero sin embargo la especificación suele estar muy bien documentada. Una API puede estar asociado al lenguaje en sí, o bien haber sido creada por una empresa o casa comercial independiente. Son ejemplos de apis muy conocidas JavaScript la api Canvas y la api de Google Maps.

Además de estos términos es frecuente que en torno a la programación surjan otros que vamos a citar brevemente:

IDE: un IDE (integrated development environment ó entorno de desarrollo integrado) es una aplicación que facilita la programación en un lenguaje. Un IDE suele incluir un editor, herramientas de generación de código automática, de corrección sintáctica, de ejecución de código, depuración, etc. Son IDEs que se pueden usar en JavaScritp Eclipse, Aptana, Codelobster, etc.

SDK: son las siglas de software development kit ó kit de desarrollo de software. Es un paquete de software que puede incluir varias cosas útiles para la programación en un determinado lenguaje, desde compilador, depurador, ejemplos, organizadores de ficheros, etc.

Toolkit: término que puede englobar librerías e incluso frameworks. Suele usarse para aludir a un conjunto de herramientas de desarrollo que facilita una empresa, o una comunidad de software libre. Estas herramientas pueden incluir cosas diversas, por ejemplo formatos prediseñados de botones, tablas, cajas de diálogo, etc. Un ejemplo de toolkit sería GWT (Google Web Toolkit).

Ten en cuenta que las definiciones anteriores son imprecisas en el sentido de que estos términos no tienen unos límites claros y distintas empresas o comunidades de programadores los utilizan de distinta manera.

Es imposible conocer todas las herramientas de tipo librería, framework, api, ide, sdk, toolkit, etc. para un lenguaje porque hay cientos. Una empresa (o una persona) elige una o varias de estas herramientas y desarrolla con ellas ignorando a las demás.

¿CUÁLES SON LAS MEJORES LIBRERÍAS Y FRAMEWORKS JAVASCRIPT?

Es difícil decir cuáles son los mejores frameworks o librerías JavaScript por varios motivos:

a) Posiblemente dependa de cuáles sean nuestros objetivos en una aplicación concreta. Por ejemplo habrá librerías o frameworks que manipulen muy bien el DOM pero ofrezcan pocas funciones matemáticas, mientras que otros quizás ofrezcan muchas funciones matemáticas pero no manipulen bien el DOM.

b) Las librerías y frameworks (y el resto de elementos en torno al software) están evolucionando continuamente. Por ello una librería que sea la más usada en 2089 puede dejar de serlo en 2090 debido al cambio y auge de unas frente a otras. ¿Cuál será la librería más usada el próximo año? No es fácil saberlo.

c) Algunos proyectos de librerías y frameworks se abandonan por parte de sus creadores, quedando como "reliquias fósiles".

d) No todos los programadores tienen los mismos gustos o ideas: a unos les puede gustar algo que a otros no le gusta.

Lo que sí es cierto es que aunque sea difícil decir cuáles son las mejores, sí hay una forma indirecta que nos indica por lo menos la popularidad o apoyo que tienen librerías y frameworks entre los programadores. Esta forma es simplemente medir el número de descargas o de páginas web que usan una determinada librería o framework.

De acuerdo con esto, podemos citar las siguientes librerías y frameworks JavaScript como muy populares (ten en cuenta que el grado de popularidad va cambiando con el tiempo):

Librería	Logo	Descripción
Jquery		Librería o biblioteca JavaScript creada por John Resig en 2006 y que se ha convertido en una de las más usadas. Se complementa con jQuery UI, una biblioteca que facilita distintos componentes (menús desplegables, calendarios, etc.)
AngularJS		Framework JavaScript desarrollado por Google.
Backbone		Framework JavaScript creado por Jeremy Ashkenas.
Ember		Framework JavaScript creado por Yehuda Katz.

Existen decenas de librerías y frameworks que no hemos citado (como Foundation, React, Three, Meteor, Underscore…) . Existen librerías o frameworks que tienen fines muy específicos, por ejemplo si necesitamos crear gráficos (de barras, de líneas, circulares, etc.) sobre datos nos resultará de interés una librería especializada en la creación de gráficos como ChartJS, D3.js, ó Google Charts.

En resumen: el mundo de librerías y frameworks javascript es muy amplio, y no podemos pretender conocer ni saber usar todos ellos. Tendremos que conocerlos y utilizarlos a medida que los vayamos necesitando para proyectos concretos.

VENTAJAS E INCONVENIENTES DEL USO DE FRAMEWORKS JAVASCRIPT

El uso de frameworks JavaScript tiene ventajas e inconvenientes. Como ventajas podemos citar las siguientes:

a) Elevan la productividad de los programadores: se pueden hacer cosas complejas en poco tiempo.

b) Elevan la claridad y mantenibilidad del código. Al partir de un código bien pensado y estructurado, los proyectos resultan más fáciles de mantener.

c) Reutilizan código creado por grandes programadores, posiblemente un código más efectivo y conciso del que nosotros seríamos capaces de crear.

d) Código gratuito y abierto, en general con buena documentación para los programadores.

e) Permiten el uso de funciones sin conocer el detalle del código (abstracción).

Y como desventajas:

a) Los frameworks evolucionan e incluso en algunos casos dejan de ser mantenidos y dejan de ser usados. Esto genera un problema para el mantenimiento o continuidad de sitios web.

b) Pueden generar sobrecarga. Si se utiliza un framework cuyo contenido y tiempo de carga resultan pesados para el navegador y por el contrario no se le saca partido, estaremos generando una sobrecarga innecesaria.

c) Problemas de compatibilidad. Algunos frameworks son incompatibles entre sí y usar dos de ellos al mismo tiempo puede generarnos problemas. También pueden existir problemas de compatibilidad con algunos navegadores.

d) Aprender a usar un framework es algo restringido y que únicamente nos sirve para manejar dicho framework, mientras que aprender un lenguaje de programación es algo universal que tendrá múltiples aprovechamientos y aplicaciones.

e) Si los desarrolladores cambian y no manejan el framework que se venía empleando en un sitio web, puede existir dificultad para continuar el desarrollo (o mantenerlo si ya se había concluido) debido a que tendrán que adaptarse a las particularidades del framework que se estuviera empleando, rediseñar el sitio o como peor opción, hacer un collage de diferentes frameworks o estilos.

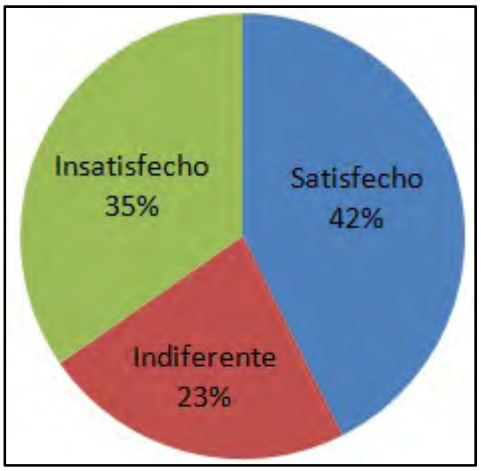
Si te preguntas si usar una librería o framework puede ser o no conveniente, la respuesta es que posiblemente dependa de lo que quieras hacer. Pero ten en cuenta que las grandes multinacionales como Google, FaceBook, Microsoft, etc. y las grandes empresas usan frameworks porque en general se acepta que las ventajas que aportan son mucho mayores que sus inconvenientes.

EJERCICIO OPCIONAL

Este ejercicio no forma parte del curso, por eso se califica como opcional. El ejercicio consiste en ver un breve ejemplo de uso de una librería JavaScript para creación de un gráfico.

Considera que como resultado de realizar una encuesta a los clientes de un hotel se han obtenido estos resultados: 35 % insatisfechos, 42 % satisfechos y 23 % indiferentes.

Se pide utilizar una librería o framework JavaScript especializada en la creación de gráficos para crear un gráfico circular que refleje dichos resultados. El resultado a obtener debe ser similar a este:



Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01195E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

API CANVAS HTML Y JAVASCRIPT. EJEMPLOS. DIBUJAR CÍRCULOS, FORMAS, GRÁFICOS, ANIMACIONES, JUEGOS, ETC. (CU01195E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº95 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

CANVAS COMO SUPERFICIE GRÁFICA

En el pasado, la creación de efectos gráficos y animaciones no resultaba sencilla dentro de páginas web. Un recurso que fue muy utilizado fue insertar applets Java u objetos o películas flash (archivos swf), con el inconveniente de que esto implicaba hacer uso de Flash Player, un plugin adicional que se debía instalar en los navegadores. La tendencia actual es que la animación gráfica quede dentro de los estándares web definidos por HTML, JavaScript y CSS sin dependencias de terceras partes.

Dentro del desarrollo del estándar de HTML se ha incluido una nueva etiqueta o elemento HTML: <canvas>. Esta etiqueta funciona igual que cualquier otra en el sentido de que se abre como <canvas> y se cierra como </canvas>.

El elemento canvas es un lienzo encima del cual se pueden dibujar gráficos y animaciones. Permite generar gráficos en pantalla, crear animaciones, manipular imágenes e incluso video, etc. En este sentido, permite generar las animaciones gráficas necesarias utilizando HTML+JavaScript+CSS, de modo que todos los desarrollos web puedan completarse con lo que se denominan lenguajes web sin necesidad de recurrir a tecnologías de terceras partes como Flash o los applets de Java.

La incorporación de toda esta potencia gráfica corre a cargo de los navegadores. Dada la complejidad que conlleva, todo lo relacionado con canvas se dice que constituye una API (interfaz de programación de aplicaciones). Algunos navegadores antiguos no admiten los elementos <canvas>, pero todos los navegadores modernos sí lo hacen.

El elemento canvas constituye un soporte al que se le da uso a través de JavaScript.

Nosotros en este curso no vamos a estudiar la api Canvas. Simplemente nos limitaremos a citar algunas de las posibilidades que el uso de este elemento junto a JavaScript permite para los desarrollos web. La creación de elementos gráficos avanzados y animaciones avanzadas constituye un área de especialización dentro de los desarrollos web, por tanto no lo consideramos un conocimiento “básico”. No obstante, es adecuado tener una idea de qué se puede hacer con canvas y JavaScript.

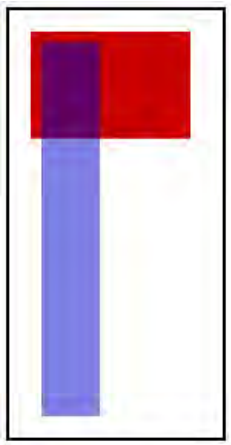
UTILIZANDO CANVAS

Un elemento HTML canvas tiene propiedades y métodos que podemos usar a través de JavaScript, de la misma forma que usamos propiedades y métodos sobre el resto de elementos HTML como <img>, <input>, <label>, etc.

No vamos a detenernos a estudiar el API Canvas, pero queremos mostrar brevemente algunos ejemplos que nos den una idea de qué es y para qué sirve. Escribe este código en en editor de textos y guárdalo con un nombre de archivo como ejemplo1.html:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="text/javascript">
    function dibujar() {
      var canvas = document.getElementById("canvas");
        var ctx = canvas.getContext("2d");
        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (20, 20, 150, 100);
        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 350);
    }
  </script>
 </head>
 <body onload="dibujar();">
   <canvas id="canvas" width="200" height="400" style="border-style:solid;">
   Su navegador no tiene soporte para mostrar el contenido
   </canvas>
 </body>
</html>
```

El resultado obtenido será algo similar a esto:



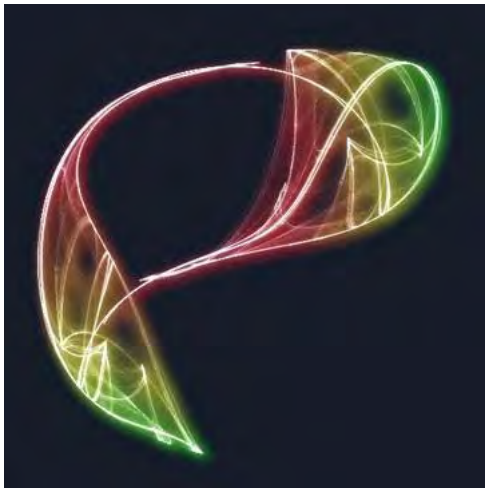
Si investigas un poco sobre el código podrás comprender con facilidad qué es lo que hace. No vamos a entrar en detalles porque no es el objetivo de este curso.

Quizás pienses que esto es relativamente poco útil, sin embargo hay un gran potencial detrás del elemento canvas unido a la potencia de JavaScript. La creación anterior implica que generamos una imagen a partir de código, esto implica un potencial enorme, ya que la imagen puede estar ligada a datos (introducidos por el usuario, extraídos de una base de datos, tomados de otra página web, etc.), y también ligada a eventos (lo cual permite crear movimientos en respuesta a la acción del usuario, incluso crear juegos) o ligada al tiempo (animaciones, imágenes que van cambiando en el tiempo).

Escribe este código en en editor de textos y guárdalo con un nombre de archivo como ejemplo2.html:

```
<html>
<head><title>Ejemplo canvas de w3.org curso JavaScript aprenderaprogramar.com </title><head>
<body>
<h1> Ejemplo canvas curso JavaScript aprenderaprogramar.com </h1>
<canvas id="canvasGlowing" width="800" height="450">
El navegador no admite canvas</canvas>
</body>
<script>
 var canvas = document.getElementById('canvasGlowing');
 var context = canvas.getContext('2d');
 var lastX = context.canvas.width * Math.random();
 var lastY = context.canvas.height * Math.random();
 var hue = 0;
 function line() {
  context.save();
  context.translate(context.canvas.width/2, context.canvas.height/2);
  context.scale(0.9, 0.9);
  context.translate(-context.canvas.width/2, -context.canvas.height/2);
  context.beginPath();
  context.lineWidth = 5 + Math.random() * 10;
  context.moveTo(lastX, lastY);
  lastX = context.canvas.width * Math.random();
  lastY = context.canvas.height * Math.random();
  context.bezierCurveTo(context.canvas.width * Math.random(),
                  context.canvas.height * Math.random(),
                  context.canvas.width * Math.random(),
                  context.canvas.height * Math.random(),
                  lastX, lastY);

  hue = hue + 10 * Math.random();
  context.strokeStyle = 'hsl(' + hue + ', 50%, 50%)';
  context.shadowColor = 'white';
  context.shadowBlur = 10;
  context.stroke();
  context.restore();
 }
 setInterval(line, 150);

 function blank() {
  context.fillStyle = 'rgba(0,0,0,0.1)';
  context.fillRect(0, 0, context.canvas.width, context.canvas.height);
 }
 setInterval(blank, 140);
</script>
</html>
```

El resultado esperado es que se dibujen líneas creando un efecto tridimensional en la pantalla (ten en cuenta que en navegadores antiguos no se mostrará el resultado, sólo los navegadores modernos admiten canvas).



¿PARA QUÉ SIRVE CANVAS?

El api Canvas combinado con CSS, HTML y JavaScript nos permite hacer todo aquello que se nos pueda ocurrir. Vamos a citar algunas de las aplicaciones habituales:

a) Dibujar formas (líneas, puntos, círculos, etc.)

b) Dibujar texto

c) Dibujar encima de imágenes o manipular imágenes

d) Dibujar gráficos

e) Crear animaciones de todo tipo (matemáticas, escolares, publicidad, etc.).

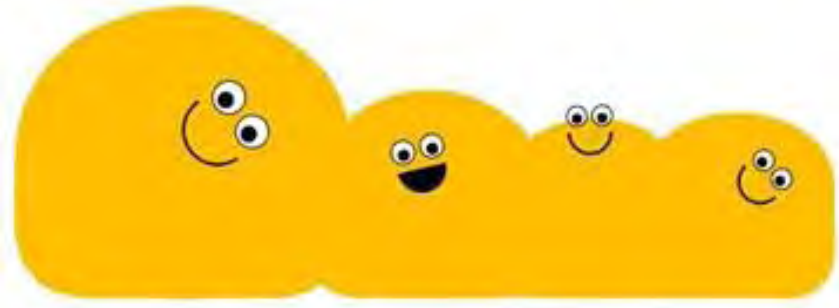
f) Crear espacios tridimensionales por donde el usuario puede moverse

g) Crear juegos

h) Manipular videos por ejemplo crear efectos gráficos sobre ellos

i) Hacer todo lo anterior en respuesta a eventos (por interacción con el usuario)

j) Hacer todo lo anterior ligado a datos variables



Las bolas de la imagen anterior se pueden dibujar con relativa facilidad con Canvas. También se pueden dotar de animación, hacer que salten, se dividan, etc.

Si escribes canvas examples en un buscador como google o bing podrás acceder a numerosos ejemplos de uso de canvas. Muchos de ellos facilitan el código fuente de forma ordenada para que te sea fácil reproducirlo en tu ordenador o incorporarlo a tus páginas web.

VENTAJAS E INCONVENIENTES DE CANVAS

La principal ventaja de Canvas es permitir la creacion y manipulación de todo tipo de formas e imágenes sin dependencias de plugins o tecnologías externas a los propios lenguajes de desarrollo web y navegadores. Igualmente es una gran ventaja la facilidad que brinda para la interacción gráfica con el usuario o para ligar el aspecto gráfico a datos que cambian dinámicamente.

Canvas es una herramienta que, al igual que todo, bien usado resulta muy útil y mal usado puede generarnos problemas. Un inconveniente ligado a canvas es pretender hacer páginas webs muy animadas y con esto sobrecargar el computador del usuario. Ten en cuenta que crear animaciones consume recursos y puede ralentizar la navegación por las páginas web o crear molestias al usuario.

EJERCICIO OPCIONAL

Este ejercicio no forma parte del curso, por eso se califica como opcional. El ejercicio consiste en dibujar la bandera de tu país usando canvas y JavaScript. La respuesta al ejercicio debe ser el código que genere la bandera.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01196E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

API DRAG AND DROP HTML Y JAVASCRIPT. EVENTOS. EFECTOS. DATATRANSFER. EJEMPLOS. ARRASTRAR Y SOLTAR. (CU01196E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº96 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

DRAG AND DROP

Seguramente hayas trabajado con Windows y con el explorador de archivos de Windows, donde se pueden cambiar archivos de ubicación simplemente haciendo click sobre ellos, arrastrándolos y soltándolos. Este proceso de arrastrar y soltar (drag and drop) resulta muy cómodo para realizar determinadas tareas y se ha incorporado a los desarrollos web a través de lo que se ha llamado Api Drag and Drop .

Con el api Drag and Drop podremos arrastrar y soltar imágenes, textos, enlaces, archivos o cualquier elemento que se nos ocurra, gracias a que la nueva especificación HTML incorpora todo lo necesario para facilitar que los programadores puedan incorporar esta potencialidad en desarrollos web y apps para smartphones.

EVENTOS DRAG AND DROP

Un aspecto importante es que con la nueva especificación disponemos de nuevos eventos para facilitar la programación del arrastre y soltado. Para entender estos eventos debemos tener en cuenta que toda operación de arrastre y soltado implica a dos elementos: el elemento origen que se selecciona y arrastra (al que a veces se llama fuente o source) y el elemento de destino, que es el elemento encima del cual se suelta el elemento que se ha arrastrado.

Nombre del evento	Descripción aprenderaprogramar.com
dragstart	Se dispara sobre el elemento origen cuando comienza el arrastre. En este momento se definen los datos asociados al elemento de origen.
drag	Similar al evento mousemove, pero afecta al elemento origen que se ha seleccionado y se está arrastrando.
dragend	El elemento origen que se ha arrastrado sufre el evento dragend cuando termina el arrastre. Esto no aplica si lo que se arrastra es un archivo desde el sistema operativo hacia el navegador.
dragenter	Se dispara sobre el elemento de destino cuando el puntero del ratón entra dentro del área en que se puede realizar el soltado.
dragover	Similar al evento mousemove, pero afecta al elemento destino cuando sobre él se está arrastrando algo y todavía no se ha soltado.
drop	Evento que se dispara en el elemento de destino cuando se suelta algo que se estaba arrastrando sobre él.
dragleave	Evento que se dispara en el elemento de destino cuando el puntero del ratón sale del área ocupada por este elemento.

Una cuestión a tener en cuenta es que para evitar acciones de defecto del navegador sobre elementos destinados a ser arrastrados y soltados deberemos usar preventDefault(), instrucción de la que ya hemos hablado anteriormente en este curso.

Todos los eventos de arrastre tienen una propiedad denominada dataTransfer que se usa para almacenar la información sobre lo que se está arrastrando. La información sobre lo que se está arrastrando se compone de dos elementos: el tipo o formato de lo que se arrastra, y su valor. Por ejemplo el tipo puede ser texto y su valor “aprenderaprogramar.com”. O el tipo podría ser una url y su valor “http://aprenderaprogramar.com”.

En este curso no vamos a entrar a estudiar en detalle la api drag and drop, no obstante, al igual que la api Canvas, puede ser muy interesante para determinadas aplicaciones y tiene un gran potencial.

Algunos navegadores antiguos no admiten drag and drop, pero todos los navegadores modernos sí lo hacen.

Veamos brevemente un ejemplo que nos dé una idea de qué es y para qué sirve drag and drop. Escribe este código en en editor de textos y guárdalo con un nombre de archivo como ejemplo1.html:

```
<!DOCTYPE HTML>
<html>
<head><meta charset="utf-8"><title>Ejemplo drag and drop aprenderaprogramar.com</title>
<style> *{font-family:sans-serif;}
#div1 {width:350px;height:70px;padding:30px;border:1px solid #aaaaaa;}
</style>
<script>
function allowDrop(ev) {ev.preventDefault();}
function drag(ev) {ev.dataTransfer.setData("text", ev.target.id);}
function drop(ev) {
   ev.preventDefault();
   var data = ev.dataTransfer.getData("text");
   ev.target.appendChild(document.getElementById(data));
            document.getElementById(data).style.color="blue";
            setTimeout(function() {cambiarEstilo (document.getElementById(data))}, 700);
}
function cambiarEstilo(elemento) {
if (elemento.style.color=="blue") { elemento.style.color="red"; } else {elemento.style.color="blue" }
setTimeout(function() {cambiarEstilo (elemento)}, 700);
}
</script>
</head>
<body>
<p>Mueve el texto al rectángulo:</p>
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div><br/>
<div id="drag1" draggable="true" ondragstart="drag(event)" style="border-style:solid; display:inline-block;
padding:5px;">
Curso JavaScript aprenderaprogramar.com
</div>
</body>
</html>
```

El resultado esperado es que se nos muestre un texto dentro de un rectángulo junto a un rectángulo vacío. Cuando arrastremos el texto sobre el rectángulo vacío, la caja del texto quedará dentro del rectángulo vacío y empezará a parpadear en color azul y rojo (ten en cuenta que en navegadores antiguos no se mostrará el resultado, sólo los navegadores modernos admiten drag and drop).

LOS ARCHIVOS Y DRAG AND DROP

Hay muchos aspectos de drag and drop que habría que citar pero que no vamos a abordar en este curso. No obstante, hay uno de ellos que merece ser comentado por su especial importancia: drag and drop facilita el arrastre de archivos, no sólo dentro del navegador, sino desde nuestro ordenador (nuestro sistema operativo) al navegador.

Esta potencialidad junto al uso de otras técnicas de programación hace más fácil de lo que era antes la operación con archivos, bajada de archivos desde el servidor a nuestro computador, o subida de archivos desde nuestro computador al servidor. Aunque no vayamos a profundizar en esto, nos ha parecido interesante citarlo para que lo tengas presente por si te surje la necesidad de realizar operaciones con archivos en aplicaciones web usando drag and drop.

¿PARA QUÉ PUEDE SERVIR DRAG AND DROP?

Drag and drop facilita la interacción del usuario con el navegador y puede ser empleado para todo aquello que se nos ocurra. Entre las aplicaciones habituales podemos indicar las siguientes:

a) Permitir al usuario arrastrar los archivos a subir a un servidor (por ejemplo archivos de imágenes).

b) Permitir al usuario realizar selecciones arrastrando, por ejemplo permitir que elija por un lado un modelo de sillón y por otro lado un modelo de tapizado para el sillón, arrastrando la opción elegida a partir de varias opciones posibles.

c) Aplicaciones de comercio electrónico donde el usuario elige lo que va a comprar y lo arrastra a su cesta de la compra.

c) Crear juegos donde el usuario arrastra y suelta elementos.

d) Permitir al usuario ordenar elementos (por ejemplo cambiar el orden de una colección de fotografías arrastrando y soltando éstas a distintas posiciones).

e) Crear aplicaciones de diseño donde los diseños se crean arrastrando y soltando elementos. Puede usarse para cualquier tipo de aplicación: diseño industrial, jardinería, decoración, ingeniería, y por supuesto para diseño web de páginas web completas o de elementos determinados como formularios. Combinando drag and drop con otras técnicas puede permitirse editar propiedades de los elementos, cambiar su aspecto, moverlos de sitio, etc.

f) Permitir crear efecto de ventanas tipo Windows en desarrollos web, que el usuario puede arrastrar y soltar en diferentes puntos, simulando que estuviera trabajando con una aplicación de escritorio.

g) Crear efectos de desplegado: podemos simular que una imagen se va desplegando o un texto se va haciendo visible o cambiando su aspecto arrastrando un elemento.

PARA QUIEN QUIERA INVESTIGAR

Para quien quiera investigar y profundizar en el api drag and drop citamos algunos elementos relevantes que deberían estudiarse con detenimiento para el correcto manejo de esta api:

a) Eventos dragstart, drag, dragend, dragenter, dragover, drop, dragleave.

b) Objeto dataTransfer

c) Métodos setData, getData, clearData y setDragImage.

d) Propiedades types, files, dropEffect, effectAllowed.

EJERCICIO OPCIONAL

Este ejercicio no forma parte del curso, por eso se califica como opcional. El ejercicio consiste en partir de una imagen cuadrada y crear a partir de ella cuatro imágenes de igual tamaño que serían las piezas del puzzle. A la izquierda de la pantalla deben mostrarse las cuatro imágenes en orden aleatorio (puzzle desordenado). A la derecha de la pantalla debe verse un cuadrado con los cuatro bordes de las piezas del puzzle. El usuario deberá arrastrar cada pieza del puzzle a su posición correcta. Cuando el puzzle esté correctamente construido debe aparecer el mensaje: ¡Enhorabuena: puzzle correctamente construido!

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01197E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

API GEOLOCATION Y API GOOGLE MAPS JAVASCRIPT. EJEMPLOS. INSERTAR MAPAS INTERACTIVOS, SATÉLITE, ETC. EN WEBS. EFECTOS. (CU01197E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº97 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

GEOLOCALIZACIÓN Y MAPAS CON JAVASCRIPT

En los últimos años ha tenido un gran auge el uso de mapas interactivos, sistemas de información geográfica, aplicaciones para smartphones que ayudan a calcular rutas o que muestran información sobre una ciudad o un monumento basándose en la localización geográfica. Existen herramientas JavaScript preparadas para aplicaciones de este tipo.

Las herramientas que incorporan los navegadores modernos relacionadas con geolocalización se han llamado “Api geolocation”. Los navegadores modernos junto con herramientas que facilitan empresas como Google dan pie a que los programadores puedan incorporar esta potencialidad en desarrollos web y apps para smartphones.

EL OBJETO NAVIGATOR.GEOLOCATION

El objeto navigator.geolocation nos facilita métodos para trabajar con la geolocalización del usuario. Tener en cuenta que muchos navegadores están configurados para impedir que se aplique la geolocalización (detección de la ubicación del usuario) por motivos de privacidad. Por tanto los resultados obtenidos pueden ser:

a) Podemos obtener la geolocalización (el usuario tiene configurado el navegador para permitirlo).

b) La geolocalización se obtiene sólo si después de que se pida permiso al usuario para facilitar su localización, el usuario da una respuesta positiva.

c) No se puede obtener la geolocalización (el usuario tiene configurado el navegador para no permitirlo).

Método de navigator.geolocation	Descripción aprenderaprogramar.com
getCurrentPosition (funcionDeProceso, funcionDeError, configuration)	Devuelve la información sobre la localización del navegador. Sólo es obligatorio incluir la funcionDeProceso. Esta función es la función que recibirá como parámetro la información de geolocalización devuelta por el método. Si se incluye funcionDeError, esta es la función a la que se enviará un objeto de tipo PositionError que lleva la información relativa al error que se ha producido. El parámetro configuration permite pasar un objeto con información de configuración relacionada con la geolocalización (como enableHigAccuracy para obtener máxima precisión, timeout para indicar el tiempo máximo de espera ó maximumAge para indicar el tiempo máximo que se almacenarán ubicaciones obtenidas en caché).
watchPosition (funcionDeProceso, funcionDeError, configuration)	Revisa la localización del navegador periódicamente e informa sobre cambios en la localización. Sólo es obligatorio incluir la funcionDeProceso. Esta función es la función que recibirá como parámetro la información de geolocalización devuelta por el método. Este método está pensado para smartphones o tablets donde el usuario se va desplazando y cambiando su localización.
clearWatch(id)	El valor de retorno de watchPosition puede ser almacenado en una variable de la misma forma que se hacía con setInterval(). La vigilancia puede detenerse usando el método clearWatch de forma similar a como se usa el clearInterval.

Veamos brevemente un ejemplo que nos dé una idea de qué es y para qué sirve geolocation. Escribe este código en en editor de textos y guárdalo con un nombre de archivo como ejemplo1.html:

```
<!DOCTYPE html>
<html lang="es">
<head><meta charset="utf-8"><title>Curso JavaScript aprenderaprogramar.com</title>
<script type="text/javascript">
function iniciar(){
var boton=document.getElementById('obtener');
boton.addEventListener('click', obtener, false);
}

function obtener(){navigator.geolocation.getCurrentPosition(mostrar, gestionarErrores);}

function mostrar(posicion){
var ubicacion=document.getElementById('localizacion');
var datos='';
datos+='Latitud: '+posicion.coords.latitude+'<br>';
datos+='Longitud: '+posicion.coords.longitude+'<br>';
datos+='Exactitud: '+posicion.coords.accuracy+' metros.<br>';
ubicacion.innerHTML=datos;
}

function gestionarErrores(error){
alert('Error: '+error.code+' '+error.message+ '\n\nPor favor compruebe que está conectado '+
'a internet y habilite la opción permitir compartir ubicación física');
}

window.addEventListener('load', iniciar, false);

</script>
</head>
<body>
<div id="localizacion">
<button id="obtener" style="margin:30px;">Obtener mi localización</button>
</div>
</body>
</html>
```

El resultado esperado depende de la configuración del usuario: si el usuario admite facilitar su ubicación el resultado obtenido será del tipo:

```
Latitud: 67.3409361
Longitud: -52.9290236
Exactitud: 44 metros.
```

Estos valores reflejan la posición geográfica del usuario. Podemos hacer uso de ellos por ejemplo para mostrar un mapa de la zona donde se encuentra el usuario. Esto lo haríamos modificando el código anterior de la siguiente manera:

```
function mostrar(posicion){
var ubicacion=document.getElementById('localizacion');
var mapurl='http://maps.google.com/maps/api/staticmap?center='+
posicion.coords.latitude+','+posicion.coords.longitude+
'&zoom=12&size=800x600&sensor=false&markers='+posicion.coords.latitude+
','+posicion.coords.longitude;
ubicacion.innerHTML='<img src="'+mapurl+'">';
}
```

Ahora bien, con el código anterior estamos mostrando una imagen. ¿Podemos incluir un mapa interactivo de Google en nuestra página web? La respuesta es que sí, como veremos a continuación.

USO DE GOOGLE MAPS PARA MOSTRAR UN MAPA INTERACTIVO

Google permite el uso de su api de mapas para mostrar mapas en páginas web. Existen multitud de posibilidades: crear mapas temáticos, cromáticos, personalizados, rutas, etc. y podemos mostrar una zona del mundo, un país, ciudad o calles. También podemos crear efectos y dibujar sobre los mapas. Nosotros en este curso no vamos a estudiar el api de Google Maps en profundidad, simplemente mostraremos un ejemplo para ejemplificar que es muy sencillo y tiene grandes posibilidades el uso de maps en nuestros sitios web.

Escribe este código en en editor de textos y guárdalo con un nombre de archivo como ejemplo2.html:

```
<!DOCTYPE html>
<html>
 <head> <title>Simple Map</title>
   <meta charset="utf-8"><title>Curso JavaScript aprenderaprogramar.com</title>
   <style> *{font-family:sans-serif;}
            h1 {margin: 150px 0 30px 0; text-align:center; }
            #lienzoMapa {width:800px; height:600px; margin: 0 auto; border: 5px solid; }
   </style>
   <script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
   <script>
var map;
function initialize() {
 var mapOptions = {
   zoom: 4,
   center: new google.maps.LatLng(9.814103, -83.118708)
 };
 map = new google.maps.Map(document.getElementById('lienzoMapa'), mapOptions);
}
google.maps.event.addDomListener(window, 'load', initialize);
   </script>
 </head>
 <body>
 <h1>Utiliza el mapa</h1>
   <div id="lienzoMapa" ></div>
 </body>
</html>
```

El resultado esperado es que se muestre un mapa en el cual podemos desplazarnos, hacer zoom, e incluso llegar a la vista de “street view” o visión fotográfica de una localización.

Para saber las coordenadas de un lugar, podemos acceder a la web de google maps, situarnos sobre el sitio deseado, pulsar botón derecho y elegir ¿Qué hay aquí?. Se nos mostrará la dirección y coordenadas del lugar.

Hay que tener en cuenta que hemos indicado la posibilidad de obtener coordenadas del usuario o de obtener coordenadas buscándolas en Google Maps. ¿Hay más formas de obtener coordenadas? Sí. Además de JavaScript, existen otras vías para hacer geolocalización. Por ejemplo usando la IP del usuario, que puede determinarse usando lenguajes del lado del servidor como PHP, o usando servicios web de geolocalización.

PARA QUIEN QUIERA INVESTIGAR

Para quien quiera investigar y profundizar en las posibilidades del api de Google Maps puede visitar la documentación del api: https://developers.google.com/maps/documentation/javascript/.

EJERCICIO

Crea una página web donde se muestre información sobre tu ciudad conteniendo un mapa interactivo de Google Maps, de acuerdo con esta estructura para el documento HTML.

<table>
<tr><td colspan="2">Visita aquiElNombreDeTuCiudad</td></tr>
<tr><td colspan="2">Aquí un texto descriptivo sobre la ciudad</td></tr>
<tr><td>Aquí una imagen de tu ciudad</td><td>Aquí otra imagen de tu ciudad</td></tr>
<tr><td colspan="2">Aquí más texto descriptivo sobre la ciudad</td></tr>
<tr><td colspan="2">Aquí un mapa interactivo de Google Maps dimensiones 800x600 pixeles de tu ciudad</td></tr>
<tr><td colspan="2">Aquí un pie de página</td></tr>
</table>

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01198E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

SESSIONSTORAGE Y LOCALSTORAGE. DIFERENCIAS. GUARDAR DATOS EN CACHÉ Y PERSISTENCIA CON JAVASCRIPT (CU01198E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº98 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

UN MUNDO POR DELANTE CON JAVASCRIPT

En la actualidad la programación JavaScript constituye un área de especialización dentro de los desarrollos web. Las grandes empresas tienen programadores exclusivamente dedicados a JavaScript. En este curso hemos visto los fundamentos del lenguaje y de algunas apis útiles. Vamos a citar algunas apis que no hemos nombrado por el momento.

WEB STORAGE O ALMACENAMIENTO WEB JAVASCRIPT

Se han desarrollado funcionalidades que permiten el almacenamiento de información a través de JavaScript. Desde los momentos iniciales de internet, en que las webs se limitaban a mostrar información, hoy día cada vez se produce un mayor flujo de información entre usuarios y servidores y procesos donde intervienen datos que fluyen de un lado a otro. De la idea o forma de trabajo inicial en el que todo el trabajo de datos recaía en el servidor, se está pasando cada vez más a dar un mayor peso al manejo de datos del lado del cliente. Y ahí JavaScript adquiere un papel fundamental.

Inicialmente el almacenamiento de datos usando JavaScript era relativamente conflictivo: las variables desaparecían cuando se pasaba de una página web a otra. Hoy día existen herramientas como las que vamos a ver a continuación que facilitan el trabajo a los programadores permitiendo la persistencia de la información sin necesidad de almacenarla en el servidor.

Anteriormente en este curso hemos visto cómo se pueden manejar cookies con JavaScript. Las cookies presentan limitaciones por su naturaleza, de modo que se restringen al manejo de cadenas de texto sin integración en un sistema de datos bien estructurado.

En este caso vamos a referirnos a algunas tecnologías como Web Storage (no vamos a tratar aquí Indexed Database). Algunas de sus posibilidades todavía no están estandarizadas, pero otras están siendo ya usadas por la comunidad de programadores.

Con la api Web Storage se ha pretendido hacer una reformulación y potenciación del manejo de datos con algo más potente que las cookies. Con Web Storage se utiliza el ordenador del usuario (su memoria o su disco duro) para almacenar información útil para la navegación web. Esa información almacenada se recupera cuando es necesaria. Las posibilidades de almacenamiento se dividieron en dos partes, una destinada a almacenar información con tiempo de vida la duración de una sesión web (sessionStorage, por ejemplo para mantener los productos en un carrito de compra) y otra destinada a mantener los datos de forma permanente (localStorage, por ejemplo para mantener un informe de negocios), de forma similar a como haría una aplicación de escritorio.

La información almacenada queda relacionada con un sitio o aplicción web, de forma que no puede ser utilizada por un sitio o aplicación distinta a la que creó los datos.

Algunos navegadores pueden no admitir esta funcionalidad, en especial los más antiguos.

SESSION STORAGE

Escribe el siguiente código y guárdalo en un archivo html:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Web Storage API ejemplo aprenderaprogramar.com</title>
<script type="text/javascript">
function addProducto(){
var producto=document.getElementById('producto').value;
var precio=document.getElementById('precio').value;
sessionStorage.setItem(producto,precio); //ó sessionStorage[producto]=precio
mostrarDatos(producto);
}

function mostrarDatos(){
var datosDisponibles=document.getElementById('datosDisponibles');
datosDisponibles.innerHTML='';
for(var i=0;i<sessionStorage.length;i++){
		var producto=sessionStorage.key(i);
		var precio=sessionStorage.getItem(producto);
		datosDisponibles.innerHTML += '<div>'+producto+' - '+precio+'</div>';
		}
}

function limpiarVista() {
var datosDisponibles=document.getElementById('datosDisponibles');
datosDisponibles.innerHTML='Limpiada vista. Los datos permanecen.';
}

function borrarTodo() {sessionStorage.clear(); mostrarDatos(); }

</script>
</head>
<body>
<form name="formulario">
<p>Nombre del producto:<br><input type="text" name="producto" id="producto"></p>
<p>Precio:<br><input type="text" name="precio" id="precio"></p>
<p><input type="button" onclick="addProducto()" name="guardar" id="guardar" value="guardar"></p>
<p><input type="button" onclick="limpiarVista()" name="limpiar" id="limpiar" value="Limpiar Vista"></p>
<p><input type="button" onclick="borrarTodo()" name="borrar" id="borrar" value="Borrar todo"></p>
</form>
<br/>
<div id="datosDisponibles">No hay información almacenada</div>
</body>
```

El resultado esperado es que se nos permita ir añadiendo productos con su precio a la lista, limpiar la vista de productos y precios pero manteniendo almacenados los mismos, o bien borrarlos completamente pulsando el botón “Borrar todo”.

Resumen de métodos y propiedades de sessionStorage:

Método o propiedad de sessionStorage	Descripción aprenderaprogramar.com
sessionStorage.setItem('clave', 'valor');	Guarda la información valor a la que se podrá acceder invocando a clave. Por ejemplo clave puede ser nombre y valor puede ser Francisco.
sessionStorage.getItem('clave')	Recupera el value de la clave especificada. Por ejemplo si clave es nombre puede recuperar “Francisco”.
sessionStorage[clave]=valor	Igual que setItem
sessionStorage.length	Devuelve el número de items guardados por el objeto sessionStorage actual
sessionStorage.key(i)	Cada item se almacena con un índice que comienza por cero y se incrementa unitariamente por cada item añadido. Con esta sintaxis rescatamos la clave correspondiente al item con índice i.
sessionStorage.removeItem(clave)	Elimina un item almacenado en sessionStorage
sessionStorage.clear()	Elimina todos los items almacenados en sessionStorage, quedando vacío el espacio de almacenamiento.

PERSISTENCIA DE LOS DATOS CON SESSION STORAGE

Los datos con sessionStorage son accesibles mientras dura la sesión de navegación. Los datos no son recuperables si:

a) Se cierra el navegador y se vuelve a abrir.

b) Se abre una nueva pestaña de navegación independiente y se sigue navegando en esa pestaña.

c) Se cierra la ventana de navegación que se estuviera utilizando y se abre otra.

LOCALSTORAGE

Con sessionStorage se puede tratar adecuadamente el flujo de información durante sesiones de navegación (por ejemplo, mantener los datos de un carrito de la compra). ¿Pero qué ocurre si quisiéramos almacenar esa información más allá de una sesión y que estuviera disponible tanto si se cierra el navegador y se vuelve a abrir como si se continúa navegando en una ventana distinta de la inicial?

Esto trata de ser respondido por localStorage. Este objeto tiene las mismas propiedades y métodos que sessionStorage, pero su persistencia va más allá de la sesión.

Sin embargo sessionStorage tiene limitaciones importantes: el almacenamiento de los datos depende, en última instancia, de la voluntad del usuario. Aunque la forma en que se almacenan los datos depende del navegador, podemos considerar que una limpieza de caché del navegador hará que se borren los datos almacenados con localStorage. Si el usuario no limpia la caché, los datos se mantendrán durante mucho tiempo. En cambio hay usuarios que tienen configurado el navegador para que la caché se limpie en cada ocasión en que cierren el navegador. En este caso la persistencia que ofrece localStorage es similar a la que ofrece sessionStorage.

No podemos confiar el funcionamiento de una aplicación web a que el usuario limpie o no limpie caché, por tanto deberemos seguir trabajando con datos del lado del servidor siempre que deseemos obtener una persistencia de duración indefinida.

EL EVENTO STORAGE

Dado que localStorage permite que se reconozcan datos desde distintas ventanas ¿Cómo detectar en una ventana que se ha producido un cambio en los datos? Para ello se definió el evento storage: este evento se dispara cuando tiene lugar un cambio en el espacio de almacenamiento y puede ser detectado por las distintas ventanas que estén abiertas.

Para crear una respuesta a este evento podemos escribir:

window.addEventListener("storage", nombreFuncionRespuesta, false);

Donde nombreFuncionRespuesta es el nombre de la función que se invocará cuando se produzca el evento.

DIFERENCIAS ENTRE COOKIES Y STORAGE

Los objetos storage juegan un papel en alguna medida similar a las cookies, pero por otro lado hay diferencias importantes:

a) Las cookies están disponibles tanto en el servidor como en el navegador del usuario, los objetos storage sólo están disponibles en el navegador del usuario.

b) Las cookies se concibieron como pequeños paquetes de identificación, con una capacidad limitada (unos 4 Kb). Los objetos storage se han concebido para almacenar datos a mayor escala (pudiendo comprender cientos o miles de datos con un espacio de almacenamiento típicamente de varios Mb).

Tener en cuenta que de una forma u otra, ni las cookies ni los objetos storage están pensados para el almacenamiento de grandes volúmenes de información, sino para la gestión de los flujos de datos propios de la navegación web.

MÁS POSIBILIDADES DE JAVASCRIPT CON LOS NAVEGADORES MODERNOS

En este curso hemos hecho un recorrido por los fundamentos del lenguaje JavaScript y muchas de las tecnologías que hay en torno a él, pero extendernos en todas las posibilidades del lenguaje y extensiones al mismo requeriría alargar un curso y ese no era nuestro objetivo. Nuestro objetivo era dar una visión sintética del lenguaje y sus posibilidades.

A modo de referencia para aquellas personas que quieran profundizar en la programación JavaScript vamos a citar algunas extensiones o Apis de interés con las que no hemos trabajado:

Api forms: concebida para facilitar la validación de formularios definiendo eventos, estados, métodos, etc.

Api Indexed Database: concebida para almacenar grandes volúmenes de información estructurada, de forma análoga a una base de datos pero con ciertas particularidades.

Api File: destinadas a facilitar la operación con archivos y directorios.

Cross Document Messaging: concebida para comunicar entre sí diferentes ventanas.

Web sockets: concebida para envío y recepción de información del servidor en periodos cortos de tiempo, útil para aplicaciones de tiempo real (como una conversación a través de chat).

Web workers: concebida para hacer posible procesar múltiples tareas al mismo tiempo (multiprocesamiento).

History: concebida para permitir la navegación por páginas visitadas e incluso por estados dentro de la evolución de una web cuando no ha habido recarga explícita de la misma (por ejemplo debido al uso de Ajax).

Offline: concebida para detectar la falta de conexión a internet y permitir el trabajo sin conexión.

EJERCICIO

Si has seguido el curso hasta aquí darte nuestra enhorabuena. No vamos a plantear aquí un ejercicio, únicamente te propondremos que dejes un comentario en los foros aprenderaprogramar.com indicándonos qué te ha parecido el curso, qué es lo que te ha parecido lo mejor de él y cuáles serían los aspectos mejorables. Preparar este curso ha requerido muchas horas de dedicación, pensamos que pedirte unos minutos para que nos ayudes a mejorar y conocer tu opinión es razonable y será algo que te agradeceremos.

Próxima entrega: CU01199E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

APRENDERAPROGRAMAR.COM

FINAL DEL CURSO TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO. (CU01199E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº99 del Tutorial básico “JavaScript desde cero”.

Autor: César Krall

FINAL DEL CURSO TUTORIAL BÁSICO DEL PROGRAMADOR WEB: JAVASCRIPT DESDE CERO

Con esta entrega llegamos al final del curso. Esperamos que haya sido un curso útil y ameno para todas las personas que lo hayan seguido. Y como en todo final, cabe hacer algunas consideraciones especiales.

Gracias al equipo humano de aprenderaprogramar.com que ha hecho posible su publicación, y en especial a Enrique González, Manuel Sierra, Javier Roa, Manuel Tello y Walter Sagástegui.

Gracias a todas las personas que de una u otra forma han participado enviando propuestas de mejora, comentarios, avisos de erratas, etc. y a los alumnos que han seguido el curso en la modalidad tutorizada on-line.

A todos los que no han participado siguiendo el curso a través de la web, desde aprenderaprogramar.com les agradecemos que nos hagan llegar una opinión o propuesta de mejora sobre el mismo, bien a través de correo electrónico a contacto@aprenderaprogramar.com , bien a través de los foros. Todas las opiniones son bienvenidas y nos sirven para mejorar.

A quienes hayan seguido el curso de forma gratuita y piensen que los contenidos son de calidad y que merece dar un pequeño apoyo económico para que se puedan seguir ofreciendo más y mejores contenidos en este sitio web, les estaremos muy agradecidos si realizan una pequeña aportación económica en forma de donación pulsando sobre el enlace que aparece en la página principal de aprenderaprogramar.com. Ten en cuenta que detrás de esta web hay personas y que preparar y mantener este curso ha supuesto un gran esfuerzo. Agradeceremos de corazón tu apoyo.

A quienes tengan interés en proseguir formándose en el área de programación con aprenderaprogramar.com, les remitimos a consultar la oferta formativa en http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=64&Itemid=87

Quienes quieran consultar toda la oferta de cursos que se ofrecen en aprenderaprogramar.com pueden hacerlo en la siguiente URL:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=57&Itemid=86

Agradecemos que cualquier sugerencia, crítica, comentario o errata detectada que nos pueda permitir mejorar los contenidos para el futuro se nos haga llegar a la siguiente dirección de correo electrónico: contacto@aprenderaprogramar.com , o alternativamente a través de los foros aprenderaprogramar.com o del formulario de contacto que está a disposición de todos los usuarios en el portal web. A todos los que nos han leído y nos siguen, gracias. ¡Nos vemos en el próximo!

El equipo de aprenderaprogramar.com

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206