USERS

RESPONSIVE DESIGN PARA DIVERSOS EQUIPOS

SITIOS MULTIPLATAFORMA CON HTML5 + CSS3

- NUEVAS ETIQUETAS Y APIS
- JAVASCRIPT PARA HTML5
- FORMULARIOS AVANZADOS
- RESPONSIVE WEB DESIGN
- MEDIA QUERIES
- INTRODUCCIÓN A JQUERY Y MODERNIZR

por EUGENIA CASABONA / RICARDO CECI

DOMINE EL NUEVO PARADIGMA DE LA WEB

RU

SITIOS MULTIPLATAFORMA CON HTML5 + CSS3

por Eugenia Casabona y Ricardo Ceci

RedUSERS

TÍTULO:	Sitios multiplataforma con HTML5 + CSS3
AUTORES:	Eugenia Casabona y Ricardo Ceci
COLECCIÓN:	Manuales USERS
FORMATO:	24 x 17 cm
PÁGINAS:	352

ISBN 978-987-1949-45-8

Casabona, Eugenia
Sitios multiplataforma con HTML5 + CSS3 / Eugenia Casabona y Ricardo Ceci. - 1a ed. - Ciudad Autónoma de Buenos Aires: Fox Andina; Buenos Aires: Dalaga, 2014.
320 p.; 24x17 cm. - (Manual users; 259)

ISBN 978-987-1949-45-8

1. Informática. I. Ceci, Ricardo II. Título

CDD 005.3

VISITE NUESTRA WEB

EN NUESTRO SITIO PODRÁ ACCEDER A UNA PREVIEW DIGITAL DE CADA LIBRO Y TAMBIÉN OBTENER, DE MANERA GRATUITA, UN CAPÍTULO EN VERSIÓN PDF, EL SUMARIO COMPLETO E IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA.

RedUSERS
COMUNIDAD DE TECNOLOGÍA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios y todos los elementos necesarios para asegurar un aprendizaje exitoso.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com usershop@redusers.com + 54 (011) 4110-8700

Eugenia Casabona

Graduada en Diseño Gráfico por la UBA, su vida se divide en dos pasiones: el diseño gráfico y la docencia. Posee diez años de experiencia docente y más de catorce en desarrollo de aplicaciones y proyectos web. Ha trabajado en empresas nacionales y extranjeras, y ha brindado capacitaciones a empresas y organismos públicos. Además, ha dictado cursos y seminarios sobre diseño y desarrollo web en instituciones como la UBA y el C.C. San Martín. En la redacción de esta obra ha encontrado una oportunidad para extender mucho más allá el lugar donde transmitir sus saberes.

Dedicatoria

A mi esposo, Diego Camacho, por su infinita paciencia y su incondicional apoyo, y por ayudarme a cumplir todos y cada uno de mis sueños.

Ricardo Ceci

Es programador web, con más de diez años de experiencia. Dirige una empresa de desarrollo web y es docente de la carrera de Programador Web en un prestigioso instituto de capacitación en Buenos Aires. Además, es consultor en la implementación de nuevas tecnologías web en numerosas empresas y cámaras empresarias, y ha dictado cursos y seminarios de programación en múltiples puntos de Argentina y Colombia. Email: ricardoceci@gmail.com.

Dedicatoria y agradecimientos

En primer lugar, este libro está dedicado a dos angelitos: a un angelito que se fue hace muy poco, mi abuela Cecilia, cuyas enseñanzas y perseverancia recuerdo a cada minuto, y a otro angelito que no llegó a venir a este mundo, pero quien sé que nos está cuidando. También está dedicado a quienes me acompañan: a mi familia y, en especial, a la persona con la que comparto todos los días de mi vida, Silvana, quien soportó mis noches sin dormir durante la escritura del libro.

Son muchos los que, con su colaboración, me han hecho crecer como profesional. Quiero agradecer especialmente a Mariano, quien en el trabajo diario me hace incursionar en las novedades y avances de la programación en general, y a Maxi Firtman, mi principal referente en programación web mobile. También quiero agradecer a quienes me alentaron en los momentos más difíciles y a quienes confiaron en mí para pequeños o grandes proyectos.

Prólogo

Un gran cambio se aproxima en el mundo del desarrollo web. Las premisas que durante años rigieron el trabajo de diseñadores, maquetadores y desarrolladores están empezando a cambiar, y para no quedar fuera debemos estar preparados.

Imaginemos un futuro donde todas las interfaces gráficas, de cualquier dispositivo (pantallas táctiles de publicidades, pantallas de teléfonos IP, televisores, etcétera), pudieran utilizar un mismo lenguaje. Sumémosle a esto la posibilidad de programar dichas interfaces gráficas con un lenguaje potente y estandarizado: nos encontraríamos ante un universo de infinitas oportunidades.
En tal escenario, sería indispensable la confluencia de nuevas profesiones con las ya existentes, con el objetivo de que tales interfaces lleguen a los usuarios de la mejor manera posible.

HTML5, CSS3 y JavaScript nos abren este futuro lleno de posibilidades y lo trasladan al presente; ideas que parecían muy ambiciosas hoy son posibles, mediante el conocimiento profundo de los aspectos técnicos que conllevan.

Un requisito para la llegada de este futuro era la estandarización en la forma en que los navegadores interpretaban HTML y CSS, y su suficiente maduración para implementar la mayor parte de las especificaciones de HTML5 y CSS3. Este requerimiento, para nuestra suerte, finalmente llegó por medio de los navegadores más modernos, con lo cual las aplicaciones que desarrollemos pueden comenzar a utilizar las características presentadas en este libro.

Los dejo en manos de dos profesionales del diseño y el desarrollo web que, con la misma pasión que expresan en sus clases presenciales, aportarán todo su conocimiento y dedicación.

Que lo disfruten.

Edgardo Diego Martin Camacho
Arquitecto de Software

El libro de un vistazo

En esta obra desarrollaremos las principales novedades de la última gran revisión del lenguaje HTML, HTML5, así como también las nuevas capacidades de la versión 3 de CSS. De esta manera, analizaremos todos los aspectos que comprenden el trabajo con estas tecnologías.

*01 INTRODUCCIÓN A HTML5

El primer capítulo revisa la historia de HTML y las causas que llevaron a la evolución del lenguaje. Conoceremos sus características, nuevas capacidades y beneficios, y también las diferencias con versiones anteriores.

*02 JAVASCRIPT AVANZADO

En este capítulo haremos un repaso de JavaScript, mediante la descripción de sus características y la explicación de las técnicas avanzadas de programación en el lenguaje.

*03 LA REVOLUCIÓN DE HTML5

Aquí aprenderemos lo que implica el concepto de semántica en la web y comprenderemos los aportes de HTML5 en este aspecto. Veremos detalladamente tanto las nuevas etiquetas del lenguaje como las que han sido modificadas, y los nuevos atributos que pueden utilizar.

*04 MULTIMEDIA

Trabajaremos con los nuevos elementos de audio y video de HTML5. Aprenderemos a utilizar los reproductores nativos de los navegadores y a agregar subtítulos y controlarlos desde JavaScript.

*05 LA API DE DIBUJO CANVAS

En esta sección nos enfocaremos en la etiqueta <canvas> de HTML5, una nueva herramienta para gráficos vectoriales. Con JavaScript realizaremos, de un modo sencillo, tareas para las que antes requeríamos otras herramientas.

*06 FORMULARIOS

En este capítulo aprenderemos a trabajar con los nuevos campos para formularios que provee HTML5. Los validaremos de manera muy sencilla e incluso permitiremos completar los campos mediante reconocimiento de voz.

*07 SELECTORES AVANZADOS

En este capítulo repasaremos los selectores de uso frecuente y luego nos centraremos en los selectores avanzados de CSS3. Entenderemos su sintaxis y aprenderemos en qué casos hacer uso de cada uno.

*08 LAS NUEVAS CAPACIDADES DE CSS3

Aquí veremos las nuevas propiedades de CSS3, sus valores posibles y la compatibilidad con los distintos navegadores. También veremos cómo utilizarlas para crear interfaces visualmente enriquecidas pero de ágil renderización.

*09 INTERFAZ DE USUARIO AVANZADA

En este capítulo nos enfocaremos en el análisis de las características de CSS3, HTML5 y JavaScript que permiten crear interfaces enriquecidas y dinámicas que mejoran la experiencia del usuario.

*10 DISEÑO ADAPTABLE CON MEDIA QUERIES

Este capítulo presentará los fundamentos del diseño y la maquetación adaptable. Con el objetivo de lograr una interfaz que se adapte a las características de cada pantalla conoceremos las Media Queries.

*11 LAS APIS DE HTML5

Desarrollaremos las principales APIs de HTML5 para aprovechar el lenguaje en los diferentes navegadores. Utilizaremos la API de geolocalización, crearemos sitios capaces de funcionar sin conexión a internet, etc. Además, veremos una introducción a Device Motion API y Fullscreen API.

*Ap COMPATIBILIDAD ON WEB

Aquí aprenderemos a utilizar jQuery y Modernizr, para que nuestros sitios sean capaces de identificar las características del navegador donde se están ejecutando.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES


SITIOS WEB

Contenido

Sobre los autores 4
Prólogo 5
El libro de un vistazo 6
Información complementaria 7
Introducción 12

*01 Introducción a HTML5

HTML, un lenguaje con historia 14
Rumbo a la revolución de la Web 18
Nace HTML5 19
La caída del imperio de Flash 20
¿Qué es HTML5? 22
Características fundamentales de HTML5 25
Diferencias entre HTML 4.01, XHTML 1.0 y HTML5 26
Esa rara etiqueta llamada doctype 26
Tipos de doctype para HTML 4.01 27
Tipos de doctype para XHTML 1.0 28
Doctype para HTML5 30
El uso de la etiqueta <html> 30
Un sitio, un idioma 31
Cambios en la sintaxis general 33
Atributos: menos es más 38
Los elementos que pasaron a la historia 40
Etiquetas deprecadas 40

El mundo según CSS 41
Resumen 41
Actividades 42

*02 JavaScript avanzado

El lenguaje de programación de la Web 44
Conceptos generales 45
Variables y tipos de datos 46
La consola 48
Arrays 52
Trabajar con el DOM 57
Buscar elementos en el DOM 59
Modificar estilos y propiedades de los elementos del DOM 61
Crear elementos 63
Objetos con JavaScript 66
Agregar propiedades a nuestros objetos 67
JSON 67
Parsear JSON en JavaScript 69
Programación basada en prototipos 71
Excepción TypeError 72
Resumen 73
Actividades 74

*03 La revolución de HTML5

La Web semántica 76
¿Cómo nace? 76
¿Qué es la Web semántica? 76
Beneficios de la Web semántica 82
Semántica en HTML5 83
Anatomía de un sitio web 83
Nuevos elementos de secciones 86
Crear encabezados con <header> 87

Navegación más semántica con <nav> 89
Cierres con <footer> 91
La famosa etiqueta <article> 95
Creando secciones con <section> 97
La etiqueta <aside> 102
Adiós a la etiqueta <hgroup> 105
Nuevos elementos de agrupamiento en HTML5 106
La llegada de <main> 106
Crear figuras mediante <figure> 107
Nuevos elementos semánticos de texto 111
La etiqueta <time> 111
La nueva etiqueta <data> 115
La etiqueta <mark> 116
Las etiquetas <ruby>, <rt> y <rp> 117
La etiqueta <bdi> 120
La etiqueta <wbr> 121
Los elementos que fueron modificados 122
La etiqueta <a> 122
La etiqueta <address> 123
La etiqueta <cite> 124
La etiqueta <hr> 124
La etiqueta <i> 125
La etiqueta <b> 126
La etiqueta 126
La etiqueta 127
La etiqueta <small> 127
Atributos globales de HTML5 128
Resumen 131
Actividades 132

*04
Multimedia

Audio y video 134
Video en HTML5 134
Subtítulos en el video 138
Añadir un "poster" a nuestro video 141
Reproducción automática 142
Reproducción continua 143
Audio en HTML5 143
Audio y video avanzado con JavaScript 144
Reproductores de video de terceros 148
Resumen 149
Actividades 150

*05
La API de dibujo Canvas

Introducción al uso de Canvas 152
Trabajar con Canvas 153
Primeros dibujos 155
Círculos 159
Bordes redondeados 160
Curvas cuadráticas 161
Curvas de Bézier 162
Relleno con gradientes 163
Trabajar con imágenes 166
Trabajar con texto 168
Canvas avanzado, animaciones y frameworks 170
EaselJS 170
KineticJS 173
Resumen 175
Actividades 176

*06
Formularios

Trabajar con formularios 178
Elementos 178
Nuevos elementos en HTML5 187
Compatibilidad 187
Inputs 188
Trabajar con fechas 194
Nuevos atributos 198

Realizar validaciones....202
Resumen....205
Actividades....206

*07 Selectores avanzados

La magia de los selectores....208
Selectores básicos....208
Selectores avanzados....218
Pseudoclases y pseudoelementos....227
Pseudoclases y pseudoelementos de CSS3....229
Resumen....237
Actividades....238

*08 Las nuevas capacidades de CSS3

Un nuevo universo de posibilidades....240
Prefijos en etapa de desarrollo....240
Cuándo usar los prefijos....241
Nuevos modos de color....242
Color RGB....242
Color HSL....243
Colores translúcidos....243
Opacidad....244
Nuevas opciones para fondos....245
Background-size....245
Background-origin....246
Background-clip....246
Múltiples imágenes para los fondos....247
Fondos degradados....248
Bordes....252
Border-radius....252
Border-image....253
Sombras....256
Box-shadow....257
Text-shadow....258
Propiedades de texto....259
Word-wrap....259
Text-overflow....260
Word-break....261
Columnas de texto....262
Column-count....263
Column-width....263
Column-gap....263
Column-rule....263
Column-fill....264
Column-span....264
Columns....264
Propiedades de Interfaz de Usuario (UI)....265
Resize....265
Box-sizing....266
Outline-offset....266
Appearance....267
Resumen....267
Actividades....268

*09 Interfaz de usuario avanzada

Tipografías especiales....270
Implementación de tipografías....270
Uso de fuentes locales....272
Obtener tipografías....272
Transformaciones....275
Rotate....275
RotateX y rotateY....276
Scale, scalex y scaley....277
Skew, skewx y skewy....278
Translate, translatex y translatey....279
Sumar transformaciones....280
Transiciones....282
Animaciones....284
Usar la API para Drag&Drop....287

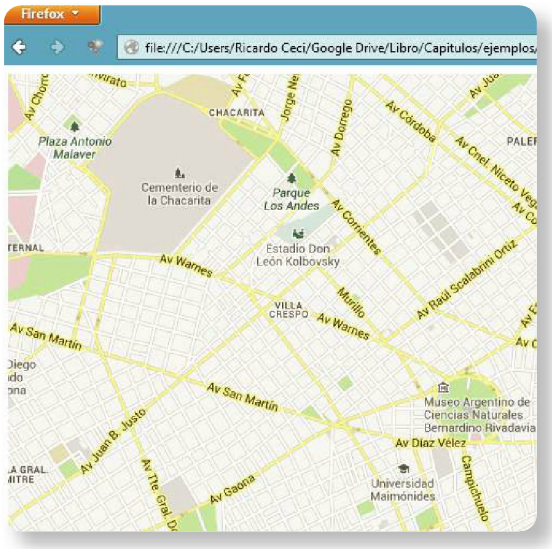
Antes de empezar....287
Eventos touch....297
Mover elementos....298
Resumen....301
Actividades....302

*10 Diseño adaptable con media queries

El diseño adaptable....304
El mundo de las media queries....305
Antepasados de las media queries....305
Anatomía de las media queries....306
Posibilidades de las media queries....307
El atributo viewport....311
Valores posibles de viewport....312
El futuro de la etiqueta meta viewport....314
Unidades de medida adaptables....315
El uso de em y rem....315
Resumen....317
Actividades....318

*11 Las APIs de HTML5

Dónde está el usuario....320
Manos a la obra....321
Más precisión....326
Trabajar sin conexión....327
Manos a la obra....328
¿Cómo funciona?....329
A tener en cuenta....330
Datos persistentes....331
Manos a la obra....332
Un poco de debugging....340
Las APIs avanzadas....341
Trabajar en segundo plano....341



Manos a la obra....342
Notificaciones de HTML5....347
Resumen....351
Actividades....352

*Ap Compatibilidad ON WEB

Pasar la prueba
¿Qué es un framework?
Introducción a jQuery
Buscar elementos en el DOM
Atravesar el DOM
Getters y Setters
Trabajar con estilos
Efectos
Sintaxis encadenada
Animaciones
Agregar y quitar elementos en el DOM
Eventos
Introducción a Modernizr
¿Cómo trabajar con Modernizr?
Modernizr.load()
Resumen
Actividades

Introducción

Este libro está orientado a todos aquellos diseñadores, maquetadores y desarrolladores web que tengan pasión por su trabajo y un gran interés en conocer en detalle las nuevas capacidades que proponen HTML5, CSS3 y JavaScript.

Para aprovechar al máximo los contenidos desarrollados en la obra recomendamos tener conocimientos básicos de los lenguajes abarcados. Los lectores de nivel principiante podrán encontrar en los distintos capítulos numerosos datos sobre las capacidades y novedades de HTML5, CSS3 y JavaScript, mientras que aquellos que cuenten con conocimientos más avanzados encontrarán una herramienta para profundizar lo que saben. Por otra parte, el libro servirá a ambos tipos de lectores para erradicar inquietudes que surgen del saber común o de la información errónea o ambigua que abunda en internet.

Mediante los temas seleccionados, nos enfocaremos en detalle en los beneficios y el potencial de trabajar con la nueva versión de cada uno de los lenguajes propuestos. Hemos intentado que los lectores puedan ir incorporando los conocimientos de manera paulatina, partiendo desde los temas más básicos e introductorios para finalizar con los más complejos. En este sentido, los usuarios más avanzados podrán pasar directamente a los temas de su interés, o aprovechar estas introducciones para refrescar sus saberes.

Sin duda, este libro ocupará su lugar dentro de los materiales de consulta de todo diseñador y/o desarrollador web, de modo tal que los lectores puedan recurrir a sus explicaciones detalladas ante cualquier duda que surja en el desarrollo de un proyecto.

*01 –

Introducción a HTML5

En este primer capítulo vamos a revisar algunos conceptos fundamentales del lenguaje HTML para, luego, conocer las principales características de la versión HTML5. Entenderemos por qué se considera esta versión como la revolución de la Web y cuáles son las características que la diferencian de las versiones anteriores.

- HTML, un lenguaje con historia 14
- Características fundamentales de HTML5 25
- Diferencias entre HTML 4.01, XHTML 1.0 y HTML5 26
- Los elementos que pasaron a la historia 40
- El mundo según CSS 41
- Resumen 41
- Actividades 42

HTML, un lenguaje con historia

No pretendemos aburrir a los lectores hablando sobre la historia y los inicios del lenguaje HTML, acerca del cual mucho habrán escuchado. Pero resulta importante repasar algunos momentos significativos de este recorrido para entender el contexto en el cual se da origen al proyecto HTML5 y cuáles son las principales razones que marcan el comienzo de esta revolución en la Web y nos muestran su futuro.

Lejos estamos de aquel documento inicial que fue presentado por Tim Berners-Lee, en el año 1991, especificando la primera –y rústica– versión de nuestro invaluable lenguaje HTML. Los curiosos y/o nostálgicos pueden aún consultar el documento en **www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html**.

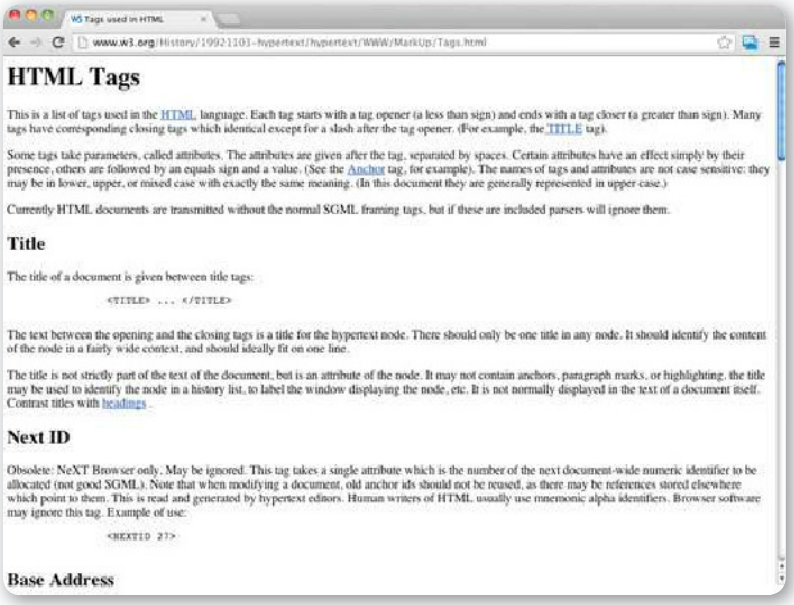


Figura 1. Primera versión de HTML publicada en el año 1991.

Esta primera versión del lenguaje fue llamada **Tags HTML** (etiquetas HTML) y contenía la especificación de 22 elementos, de los cuales 13 continúan existiendo. En esta primera edición, solo se contemplaron etiquetas para texto e hipervínculos, pero desde entonces el lenguaje HTML sigue en constante avance y evolución. Con el correr de los años, llegaron las etiquetas para tablas, imágenes y formularios, entre otras.

Pero no fue sino hasta el año 1995 cuando el lenguaje HTML logró su primera versión estándar: en septiembre de ese año se publicó **HTML 2.0** que, a pesar de su nombre, fue el primer estándar oficial del lenguaje.

Un año más tarde, en 1996, de la mano de Tim Berners-Lee, se crea la **W3C** (*World Wide Web Consortium*), organismo que se ocupa, desde entonces, de la especificación y la evolución del lenguaje HTML.

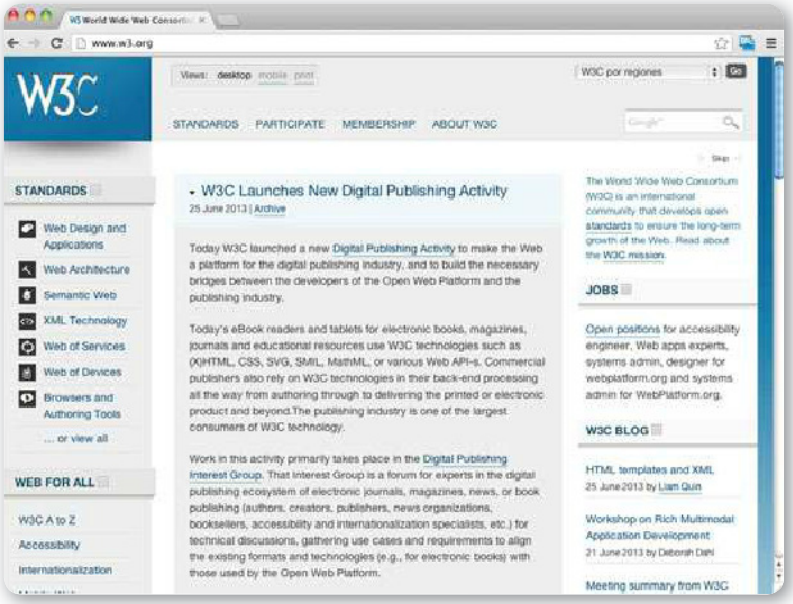


Figura 2. Sitio oficial de la **World Wide Web Consortium**, donde se puede encontrar la especificación de las distintas versiones de HTML.

En enero del año 1997, la W3C publica su primera recomendación oficial con la versión **HTML 3.2**, que presenta avances significativos respecto de las versiones anteriores, como, por ejemplo, la capacidad de insertar **applets** en Java (es decir, un componente de una aplicación que se ejecuta en el contexto de otro programa, como por ejemplo un navegador web).

Los avances cualitativos no se hicieron esperar mucho, ya que en el año 1998 se publica la versión **HTML4**, que se sigue utilizando hasta hoy. Esta versión incorpora mejoras sustanciales, como hojas de estilo (CSS), tablas complejas, mejoras en los formularios e incluso la posibilidad de incorporar scripts. La última revisión de esta versión del lenguaje se publicó a fines del año 1999 y se denominó **HTML 4.01**.

Desde entonces, los miembros de la W3C entendieron que debían ir un paso más allá y se dedicaron a una nueva y prometedora versión del lenguaje: **XHTML** (HTML de extensión), una versión avanzada de HTML basada en XML. Así fue como la evolución del HTML original fue dejada de lado, y los miembros de la W3C centraron todas sus energías en el desarrollo de XHTML.

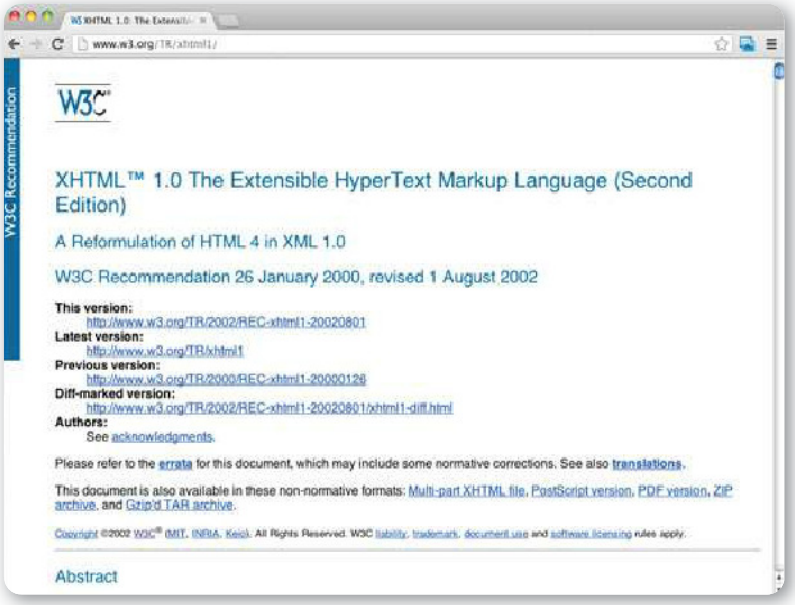


Figura 3. Especificación oficial de XHTML 1.0.

La primera especificación del lenguaje XHTML sale a luz en enero del año 2000, bajo la denominación **XHTML 1.0**, para ser revisada luego en el año 2002.

Ante esta postura adoptada por la W3C, miembros de empresas representativas en el desarrollo y avance de la Web –como Apple, Opera

y Mozilla– se mostraron preocupados por el abandono del proyecto HTML 4.01 por parte de la W3C y decidieron formar otro organismo estandarizador que continuara con la evolución natural del lenguaje a su versión 5. Este organismo se denomina **WHATWG** (*Web Hypertext Application Technology Working Group*), y podemos conocerlo más a través de su sitio oficial: **www.whatwg.org**.

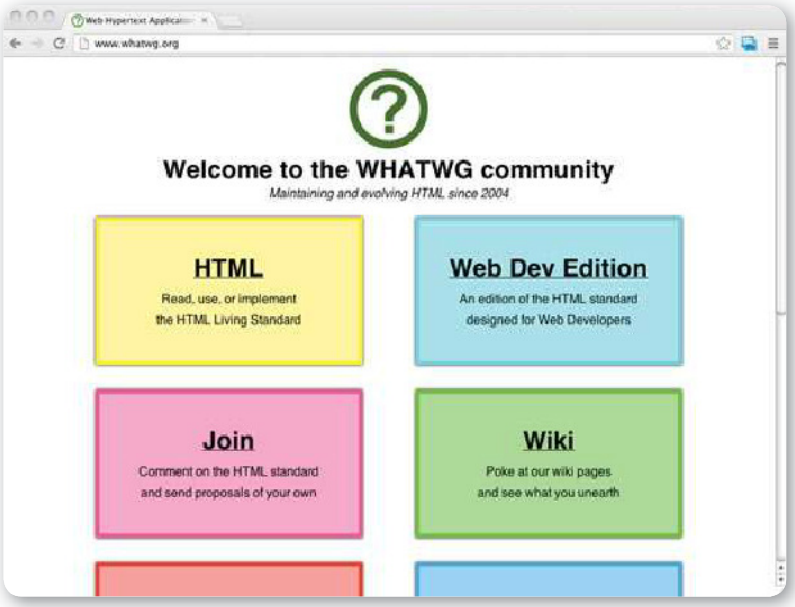


Figura 4. Sitio web oficial de la agrupación **WHATWG**.

Esta agrupación se centró exclusivamente en la especificación del primer borrador de HTML5, que fue publicado en enero del 2008. Ante la presión de este grupo y de las empresas que lo componen, la W3C decidió, en 2007, retomar el proyecto de especificación de HTML 4.01.

Durante algunos años, la W3C continuó trabajando de forma simultánea en dos proyectos de estandarización, HTML y XHTML, lo cual desbordó de preguntas a los desarrolladores: ¿cuál sería el rumbo que tomaría la W3C? ¿Por qué la misma institución trabaja sobre dos especificaciones similares? ¿Cuál sería el futuro de la Web? Los años siguientes traerían las respuestas a estas preguntas y, también, una gran cantidad de buenas noticias.

> ANTE LA PRESIÓN DE WHATWG, LA W3C RETOMÓ EL PROYECTO DE ESPECIFICACIÓN DE HTML 4.01

Rumbo a la revolución de la Web

De regreso, la W3C publica, en el año 2000, la especificación de **XHTML 1.0**, una adaptación de HTML 4.01 al lenguaje XML. Allí se prometía la gran revolución de la Web, que nunca llegó a producirse realmente, debido a que XHTML 1.0, a pesar de que posee una sintaxis más estricta y algunas mejoras semánticas, no difiere sustancialmente de su antecesor, HTML 4.01. Esta nueva versión del lenguaje mantiene casi todas las características y etiquetas de HTML 4.01, pero suma algunas restricciones y elementos propios de XML.

Finalmente, los cambios introducidos por XHTML no representan mejoras significativas para el desarrollo de sitios web. Por ello, el W3C presentó, en los años siguientes, dos nuevos borradores: **XHTML 1.1** y **XHTML 2.0**, dos nuevas especificaciones de XHTML con mejoras importantes respecto de la primera versión.

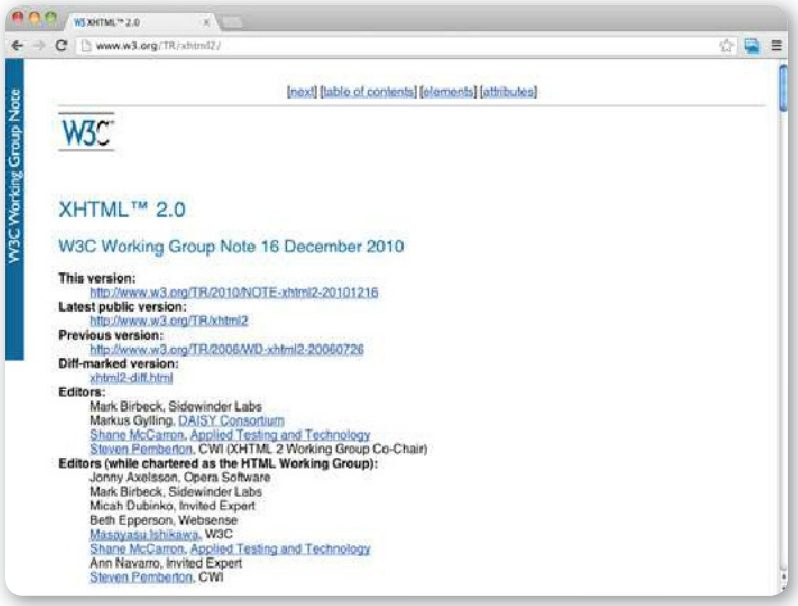


Figura 5. Especificación oficial de XHTML 2.0, publicada en el sitio de la W3C.

XHTML 2.0 promete un cambio realmente significativo para la Web. No obstante, vale recordar que al mismo tiempo existe un grupo de desarrolladores trabajando arduamente en otra especificación revolucionaria: HTML5. ¿Cuál será el rumbo que tomará esta historia? ¿Cuál será el ganador en la batalla para revolucionar la Web?

El proyecto HTML5 contaba con importante apoyo de las empresas que día a día empujan la Web, como Apple, Google, Firefox y Opera, entre otras. Esto pudo haber sido uno de los factores más importantes para que, finalmente, los miembros del W3C decidieran abandonar el proyecto de XHTML y concentrar sus recursos y energías en el desarrollo de HTML5.

Nace HTML5

Para comenzar con HTML5, deberíamos decir que es la quinta especificación relevante que sufre el lenguaje HTML. Si bien esta definición es correcta, resulta un poco pobre para definir los cambios que trae consigo, ya que cuando nos referimos a HTML5 estamos hablando de un cambio en el paradigma con el cual se venía trabajando desde hacía tiempo.

> HTML5 NO ES SOLO UNA ESPECIFICACIÓN, SINO UN CAMBIO EN EL PARADIGMA DE TRABAJO

Si bien la mayoría de los desarrolladores utilizamos, desde hace años, no solo HTML, sino también CSS y JavaScript para la construcción de proyectos web, a partir del nuevo paradigma de HTML5 estos tres lenguajes se volvieron inseparables.

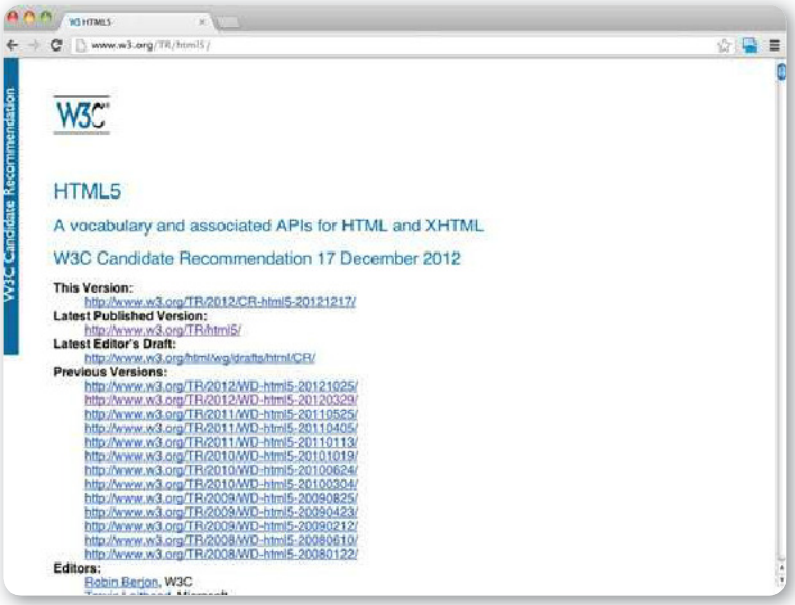


Figura 6. Especificación oficial de HTML5.

La caída del imperio de Flash

Durante mucho tiempo, **Flash** fue la herramiena favorita de la Web. Aunque no fue pensado originalmente para ser usado en internet, su condición de elemento multimedia hizo que se ganara rápidamente un lugar durante la década del noventa, ya que presentaba características invaluables para los diseñadores y desarrolladores de aquel tiempo.

Desarrollar un sitio de alto impacto visual no era tan simple hace algunos años: la maquetación con tablas y las insuficientes propiedades de CSS nos obligaban a limitar nuestra creatividad a diseños muy estructurados y básicos. Por otro lado, Flash nos seducía con una interfaz fácil de usar y herramientas gráficas que permitían explotar al máximo nuestro poder creativo.

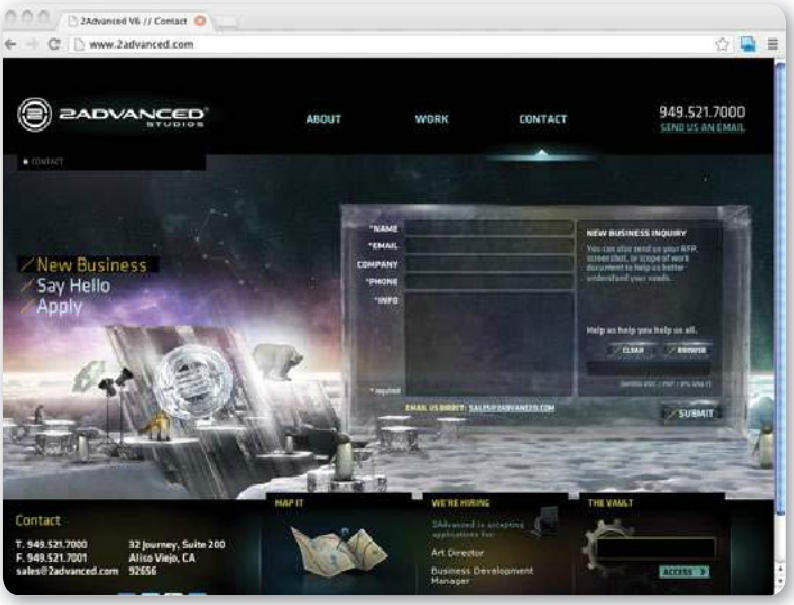


Figura 7. Página de inicio del estudio **2Advanced** (**www.2advanced.com**), reconocido por su desarrollo en Flash.

Durante varios años, los clientes estaban encantados con tener sus sitios desarrollados en Flash, pues incluían sonidos, video, animaciones y juegos que brindaban a los usuarios experiencias de navegación completamente diferentes. La mayoría de las empresas de primera línea eran dueñas de sitios realizados completamente en Flash, con un gran despliegue de recursos y efectos de animación.

Paralelamente a esto, el lenguaje **ActionScript** utilizado por Flash crecía a pasos agigantados, permitiendo desarrollar proyectos cada

vez más complejos y con mejores resultados. De este modo, gran parte de los sitios web estaban desarrollados 100 % en Flash, y se fueron dejando de lado aspectos importantes como la accesibilidad, el rendimiento de descarga y el posicionamiento, en función de la explosión visual de la interfaz de usuario.

¿Qué fue entonces lo que hizo que, desde hace algún tiempo, las empresas que habían invertido mucho dinero en desarrollos de Flash quisieran volver a invertir dinero en tener un nuevo sitio con lenguaje HTML? La respuesta a esta pregunta es bastante simple: llegó la **era de los dispositivos móviles**.

Como muchos habrán experimentado o escuchado por ahí, la mayoría de estos equipos (tablets y smatphones) no permiten visualizar archivos de Flash. Esta realidad nos lleva a deducir una respuesta simple a nuestro interrogante inicial: cada día son más y más los usuarios que acceden a la Web mediante dispositivos móviles. De hecho, las estadísticas realizadas en algunos países nos revelan que, en algunas ciudades del mundo, son más las conexiones a internet mediante dispositivos móviles que desde las clásicas computadoras de escritorio. Y aquí llegamos a una obvia conclusión: si nuestra empresa posee un sitio realizado en Flash, dejamos afuera a todos estos usuarios que a diario intentan acceder a nuestro sitio desde un dispositivo móvil.

Esta tendencia de los usuarios y del mercado nos lleva a una nueva era, la de HTML5. No es volver al pasado, a aquellos sitios duros y estáticos, sino entrar en un nuevo universo de posibilidades, donde la fuerza de HTML5, CSS3 y JavaScript combinados nos permite realizar desarrollos donde prácticamente no existen más límites que nuestra propia imaginación. La potencia de estos lenguajes es tan grande que

¿QUÉ EL VALIDADOR W3C?

En el sitio oficial de W3C, **www.w3.org**, podemos encontrar una valiosa herramienta de uso libre: el validador de sintaxis para HTML, al que podemos acceder desde **http://validator.w3.org**. Este recurso permite comprobar si nuestro código fuente tiene errores de sintaxis en cualquiera de las versiones de HTML. También es posible validar un archivo que está subido a un servidor, cargar un archivo desde nuestra máquina o bien validar entradas directas de código pegándolo en el validador.

podemos realizar sitios completamente interactivos, de alto impacto visual, excelente rendimiento y totalmente accesibles. El futuro del diseño y desarrollo web ya llegó: es hora de conocerlo en profundidad para comenzar a disfrutar de sus múltiples beneficios.

¿Qué es HTML5?

Tal como mencionamos al principio de este capítulo, HTML5 es la quinta revisión relevante que sufre el lenguaje HTML, y es más que importante resaltar que esta versión aún no es un estándar. El 17 de diciembre de 2012, la W3C anunció que la etapa de desarrollo de la especificación había llegado a su fin y que HTML5 era candidata a recomendación.

No obstante, aún no podemos "cantar victoria". Todavía quedan por delante dos años más de revisión de este lenguaje, período durante el cual no se agregarán nuevas capacidades a la especificación, pero sí es posible que algunos elementos que están actualmente sean removidos o modificados. En el mismo anuncio, la W3C señaló que la recomendación de HTML5 que será candidata para su estandarización no se publicará antes del 1 de septiembre del 2014, así que tenemos algún tiempo para seguir esperando.

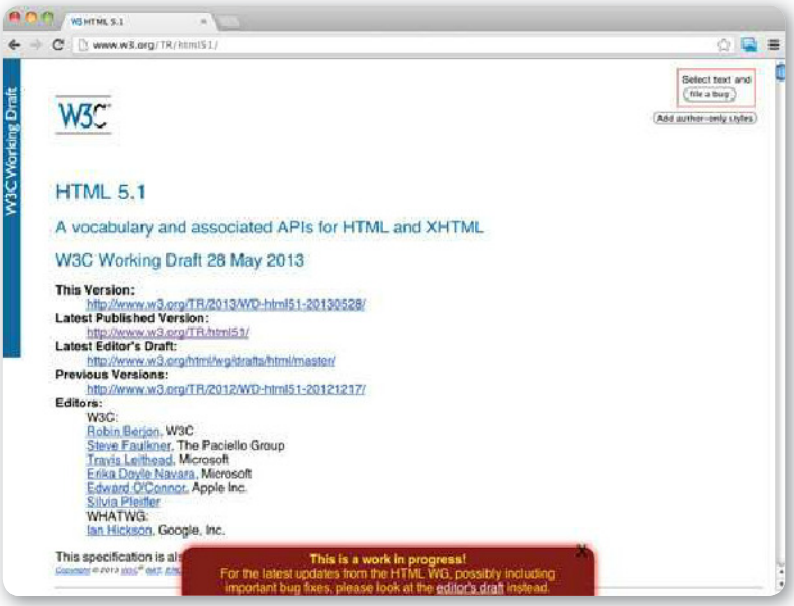


Figura 8. Sitio oficial de W3C con las especificaciones de la futura versión de HTML5.1.

A esta altura, tal vez se están preguntando **qué pasará luego de que HTML5 se vuelva un estándar**. ¿Será el final de este interesante viaje? No, claro que no. Ni bien se terminó el proceso de especificación de HTML5, un equipo de trabajo de la W3C inició el proyecto de especificación de **HTML5.1**. Sí, como lo leyeron: HTML5.1. El avance de este proyecto puede consultarse en **www.w3.org/TR/html51**.

> LA W3C YA SE ENCUENTRA TRABAJANDO EN LA ESPECIFICACIÓN DE HTML5.1

Hay que mencionar que trabajar con HTML5 tiene actualmente muchas alegrías y algunas tristezas. Hace algunos años, les hubiéramos dicho que eran puras tristezas y frustraciones, pero lejos estamos ya de esa etapa, pues actualmente contamos con navegadores modernos que tienen muy buen soporte para HTML5. No obstante, no existe un solo navegador que tenga soporte completo para HTML5, CSS3 y JavaScript, así que lamentablemente no podremos utilizar el 100 % del potencial de estos lenguajes. Todavía...

Por otro lado, tenemos usuarios que siguen utilizando navegadores antiguos como **Internet Explorer 8**, cuando la historia de este navegador con HTML5 comienza a partir de su versión número 9. Pero no se desanimen: a lo largo de los capítulos de este libro –especialmente, en el **Capítulo 12**–, aprenderemos técnicas para lograr que usuarios con navegadores viejos puedan también disfrutar de los sitios desarrollados con HTML5.

Algo que nos pasará a diario y, sobre todo, cuando arranquemos a trabajar con HTML5 y CSS3, es que nos costará recordar qué propiedad de CSS o qué capacidad de HTML5 tiene soporte en cada versión de navegador. Por suerte, en internet podremos encontrar herramientas

LA FUENTE DE HTML5

Si están interesados en conocer más, es recomendable visitar el sitio oficial de la W3C y recorrer la especificación de HTML5 en **www.w3.org/TR/html5**. Si bien internet es una fuente inagotable de conocimiento, la información que encontramos no siempre es correcta y precisa. Existen muchos sitios que ofrecen información incorrecta o desactualizada.

fantásticas y gratuitas que nos facilitarán esta tarea. Existen diversos sitios que nos proveen tablas de compatibilidad que indican qué capacidad es soportada por cada versión de navegador.

Uno de los sitios de cabecera en este aspecto es **Can I use...** (**www.caniuse.com**). Posee una completísima tabla que nos muestra tanto capacidades de HTML5 como propiedades de CSS y JavaScript. Esta página incorpora mejoras periódicamente y abarca los nuevos navegadores para dispositivos móviles que van saliendo al mercado.

Figura 9. Caniuse.com ofrece una completa tabla de compatibilidad para HTML5, CSS y JavaScript.

Es frecuente la pregunta sobre si es recomendable, hoy en día, usar HTML5, o si es mejor continuar usando XHTML. La respuesta es: **no hay una sola razón por la cual no debiéramos usar HTML5.**

DISEÑO ADAPTABLE

Es una técnica de diseño y maquetación mediante la cual detectamos la resolución de pantalla del navegador del usuario y, mediante el uso de distintas hojas de estilo, aplicamos un diseño de pantalla distinto para cada dispositivo, adaptándolo para que se vea mejor en cada resolución. Podemos ver un excelente ejemplo de sitio adaptable en **Smashing Magazine**: **www.smashingmagazine.com**.

Características fundamentales de HTML5

El concepto fundamental de HTML5 es la **simplicidad**. En tanto vayamos profundizando los temas de este libro, notaremos que esta premisa está siempre presente. Desarrollar un sitio usando HTML5 es mucho más simple que hacerlo con las versiones anteriores. Siempre se trata de hacer más simple y más accesible el código de nuestros sitios y aplicaciones web. En HTML5 veremos siempre presente el antiguo lema del arquitecto Ludwig Mies van der Rohe: menos es más.

Otro de los preceptos fundamentales de HTML5 fue reducir al máximo la cantidad de plugins o agregados que el usuario debe tener instalados en su máquina, como el reproductor de Flash. Por esta razón, en el **Capítulo 4** y en el **Capítulo 9** veremos cómo trabajar con etiquetas que permiten insertar archivos de audio y video y, también, cómo crear animaciones nativas que no requieren que el usuario tenga en su máquina algo más que su navegador.

Un precepto más que importante para HTML5 es la independencia del dispositivo: permite desarrollar sitios y aplicaciones compatibles con distintos tipos de dispositivos. En el **Capítulo 10**, veremos que no solo podemos crear un código compatible con todos los equipos, sino que también lograremos que su diseño visual se adapte a cada uno de ellos, usando la metodología de trabajo **responsive** (diseño adaptable).

Por último, pero no menos importante, HTML5 introduce más y mejores elementos de marcación, que permiten lograr una mejor estructura semántica y, por lo tanto, hacer más accesibles los sitios. En el **Capítulo 3**, conoceremos en profundidad cada uno de los nuevos elementos y su uso específico para el modelado del documento.

CÓMO SABER SI UN SITIO WEB ES RESPONSIVE

Para saber si un sitio web posee diseño adaptable, no hace falta tener una computadora, una tablet y un smartphone. Si redimensionamos la ventana del navegador, podremos ver las distintas interfaces que verá el usuario según el dispositivo con que ingrese. Esto lo logramos haciendo un doble clic sobre la barra de título del navegador y cambiando el tamaño de la ventana.

Diferencias entre HTML 4.01, XHTML 1.0 y HTML5

Es hora de poner manos a la obra y comenzar a trabajar con HTML5. Y qué mejor idea que arrancar entendiendo las diferencias estructurales y de sintaxis entre las tres versiones del lenguaje acerca de las cuales estamos hablando desde el comienzo de este capítulo.

Esa rara etiqueta llamada doctype

Cuando comenzamos a codificar un documento HTML, sin importar con cuál de las versiones estemos trabajando, debemos especificar la etiqueta **doctype**, esa etiqueta misteriosa que muchas veces no entendemos para qué sirve pero que sabemos que debe estar.

La etiqueta **<!DOCTYPE>** se utiliza para que el navegador sepa con qué versión de HTML está codificado el documento y pueda renderizarlo de manera correcta. Esta etiqueta solo puede ser usada una vez, al principio del documento, y puede ser escrita en mayúscula o minúscula, debido a que se encuentra por fuera de la etiqueta **<HTML>** (por lo tanto, el modo en el cual está escrita es irrelevante).

Los navegadores tienen dos modos de trabajo: **estándar** y **quirks**. Dependiendo de si el doctype está o no definido en nuestro archivo, el navegador funcionará en un modo u otro.

En el modo **estándar**, el navegador renderizará el documento HTML según el estándar determinado por el **doctype** del archivo. En el modo **quirks** no hay garantías de cómo se mostrará el archivo, ya que el navegador no puede determinar a qué versión de HTML corresponde. Por esta razón, siempre debemos utilizar esta etiqueta para garantizar que el navegador renderice la página en su modalidad estándar.

Como probablemente sepan, cada una de las versiones cuenta con sus propios tipos de **doctype**, que determinan el tipo de versión de la especificación. Tanto XHTML 1.0 como HTML4 poseen más de un **doctype** que permite determinar el tipo de especificación. Generalmente vamos a encontrar una **versión de transición**, más flexible a la validación, y una **versión estricta**.

Repasemos los tipos de **doctype** para cada una de las versiones de HTML y veamos qué ha cambiado en HTML5.

Tipos de doctype para HTML 4.01

HTML4 cuenta con tres tipos de **doctype**. Veamos cómo se escriben y cuál es la función específica de cada uno de ellos.

Estricto

Esta es la versión estricta del **doctype** para HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML4.01//EN" "http://www.
w3.org/TR/html4/strict.dtd">
```

El **doctype** estricto no admite el uso de ningún tipo de atributo considerado deprecado para esta versión de HTML. En caso de ser utilizado, el archivo presentará errores de validación cuando realicemos su comprobación.

Transitional

Esta es la versión de transición del **doctype** para HTML4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

En el **doctype** de transición se admite el uso de algunos atributos a los cuales se los considera obsoletos en HTML. La idea de este **doctype** es permitir a los desarrolladores migrar de forma más rápida sus sitios desarrollados en versiones anteriores a HTML4. Por esta razón, el **doctype transitional** admite el uso de algunos atributos que son considerados obsoletos para esta versión y, aun así, el archivo parará la validación de W3C.

Frameset

Esta es la versión del **doctype** para HTML 4.01 que permite el uso de las etiquetas **frame** y **frameset**. Este doctype casi no tiene uso, ya que prácticamente nadie usa etiquetas **frame** hoy en día.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

Tipos de doctype para XHTML 1.0

XHTML 1.0 cuenta con tres tipos de **doctype**. Veamos cómo se escriben y cuál es la función específica de cada uno de ellos.

Estricto

Esta es la versión estricta del **doctype** para XHTML 1.0:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Transitional

Esta es la versión de transición del **doctype** para XHTML 1.0:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

La diferencia entre el **doctype** estricto y el transicional radica en los atributos que cada uno va a admitir para que el documento HTML sea válido. En particular, en el caso del **doctype** estricto, no se admite ningún tipo de atributo que tenga que ver con la representación visual. Por ejemplo:

```
<p align="center"> Texto del párrafo </p>
```

Todos los aspectos relativos a la representación visual deben estar definidos desde el archivo de estilo CSS. Por otro lado, el **doctype** de transición es más flexible y admite el uso de algunos atributos relativos a la representación visual, de modo tal que la siguiente

porción de código resulta válida para el **doctype** transicional, pero no así para el **doctype** estricto.

```
<h1 align="center"> Texto del titulo</h1>
```

Frameset

Esta es la versión del **doctype** para XHTML 1.0 que permite el uso de las etiquetas **frame** y **frameset**. Al igual que en HTML4, el uso de este tipo de **doctype** es casi nulo, dado que nadie usaría actualmente etiquetas obsoletas como los frames.

El uso de tecnologías como AJAX ha reemplazado la utilización de frames, permitiendo crear documentos con mejor rendimiento y accesibilidad.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

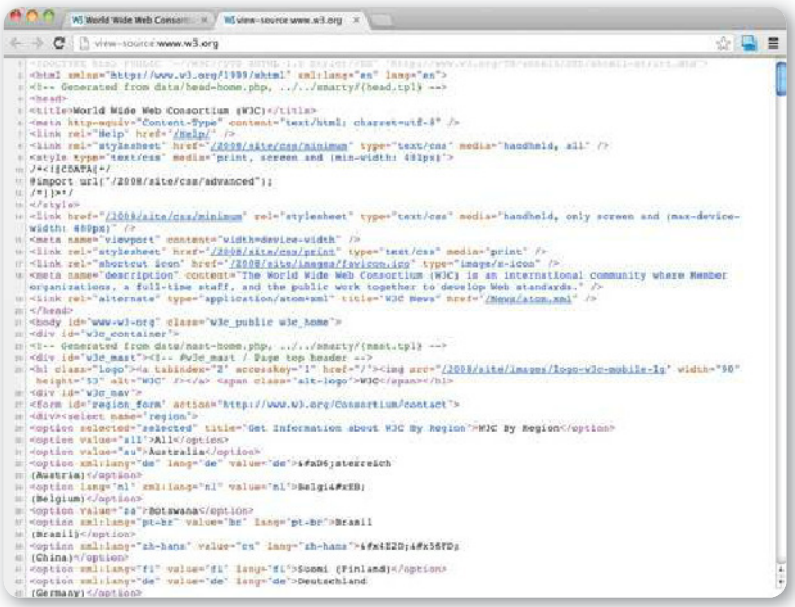


Figura 10. Captura del código fuente del sitio web de W3C, donde podemos ver que se utiliza el **doctype** estricto para XHTML 1.0.

Doctype para HTML5

HTML5 cuenta con un único tipo de **doctype**. Veamos cómo es la sintaxis de esta etiqueta:

```
<!DOCTYPE html>
```

A diferencia de sus antecesores, HTML5 cuenta con un único tipo de **doctype**, mucho más corto y simple; como vimos, este lenguaje se basa en la simplicidad. Si nunca pudieron recordar cómo se escribe correctamente la etiqueta **doctype**, ahora pueden descansar tranquilos, pues el **doctype** de HTML5 es muy fácil de recordar. La ventaja de este **doctype** no es solo el hecho de ser más simple de escribir y recordar, sino que también permite que, debajo de él, podamos emplear tanto la sintaxis de HTML4 como la de XHTML y, de este modo, hacer más fácil la migración a HTML5. Hablaremos de este tema con mayor profundidad en este mismo capítulo.

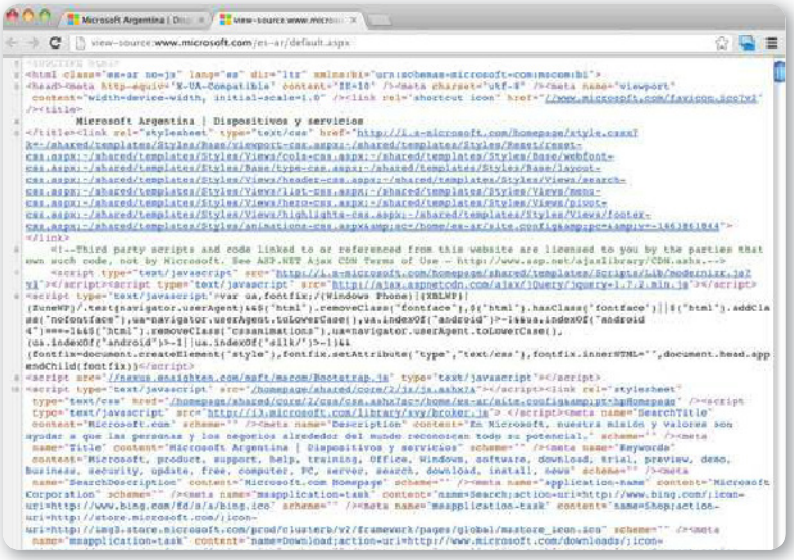


Figura 11. Captura del código fuente del sitio oficial de **Microsoft**, donde podemos ver que se está usando el **doctype** de HTML5.

El uso de la etiqueta <html>

Un punto muy importante a destacar entre los cambios en XHTML y HTML5 es el uso de la etiqueta **<html>**. En XHTML, habitualmente vamos a ver escrita la etiqueta **<html>** de la siguiente forma:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

En XHTML, es obligatorio el uso del atributo **xmlns**, es decir, el **namespace**. Un **namespace** es una forma de agrupar un conjunto de etiquetas para evitar ambigüedades de nombre. Este atributo (**xmlns**) define el espacio de nombres que va a ser utilizado en el documento. El link a espacio de nombres en el sitio de W3C se utiliza sólo como identificador. Este atributo permite a los navegadores interpretar correctamente si esas etiquetas o atributos pertenecen a la versión de HTML establecida en el **doctype.**

En HTML5 ya no es necesario determinar el **namespace**; por ende, la etiqueta **<html>** se simplifica y vuelve a ser la que usamos años atrás, desde las primeras versiones de HTML.

```
<html>

...

</html>
```

Un sitio, un idioma

Un punto importante a tener en cuenta, que muchas veces tomamos por trivial o pasamos por alto directamente, es la especificación de idioma del sitio web en el elemento raíz mediante el atributo **lang**, es decir, en la etiqueta **<html>**, tal como se muestra en el ejemplo:

¿QUÉ ES AJAX?

AJAX es el acrónimo de **Asynchronous JavaScript And XML** (JavaScript Asincrónico y XML) y consiste en una metodología que combina varios lenguajes, como HTML, JavaScript, CSS, XML, etcétera. Permite crear aplicaciones interactivas que se ejecutan en el navegador del usuario, mientras se mantiene una comunicación en segundo plano con el servidor. Así, se pueden realizar modificaciones sobre las páginas sin necesidad de recargarlas por completo.

```
<!DOCTYPE HTML>
<html lang="en">
        <head>
                <title>Titulo de mi web site</title>
        </head>
        <body>
                <p>Contenido de mi sitio web</p>
        </body>
</html>
```

En este caso, el idioma del documento está establecido en inglés. Esta simple intervención en el código mejora notablemente la experiencia del usuario, ya que permite a los programas de accesibilidad saber cuál es el idioma del documento y pronunciar las palabras de forma acorde. También permite a las herramientas de traducción determinar qué reglas debe usar. Y, por último, provoca que Google se "ponga contento" con nuestro sitio web, ya que este simple punto puede influir en el posicionamiento orgánico de nuestro proyecto en los resultados de búsqueda.

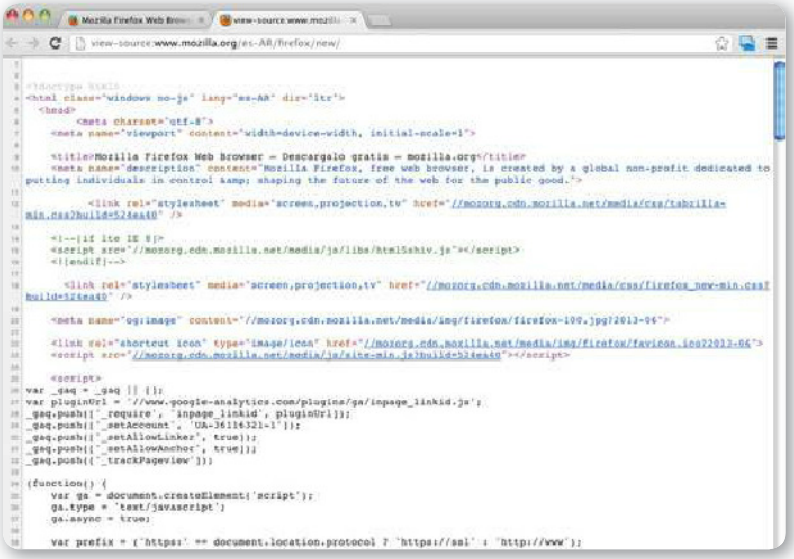


Figura 12. Captura del código fuente del sitio web de **Mozilla Firefox Argentina**, donde podemos ver claramente la declaración del idioma del documento.

Cambios en la sintaxis general

Un punto que se presta bastante a la confusión es la sintaxis que debemos utilizar en HTML5. Por esta razón, nos detendremos para aclararlo. Una de las diferencias más significativas entre HTML4 y XHTML1 es la forma de escribir las etiquetas a las cuales llamamos **etiquetas vacías** (aquellas que no admiten dentro otras etiquetas), como, por ejemplo, la etiqueta para las imágenes, los campos de texto o los saltos de línea.

A continuación, podemos ver algunos ejemplos de etiquetas vacías:

```
<br>
<img>
<input>
<hr>
<link>
```

En XHTML, esta sintaxis no está permitida, dado que XHTML toma de XML un precepto muy importante: **no pueden existir etiquetas sin cerrar**. Por lo tanto, toda etiqueta debe tener, obligatoriamente, una etiqueta de cierre. Por esta razón, muchas etiquetas tuvieron que modificar su sintaxis, incorporando una barra de cierre y un espacio en la propia etiqueta de apertura, como vemos en el ejemplo siguiente:

```
<br />
<img />
<input />
<hr />
<link />
```

De esta manera, las etiquetas quedan correctamente cerradas y son totalmente válidas para el estándar de XHTML. Ante estas dos posibilidades, tenemos que preguntarnos cuál es la sintaxis correcta que se debe usar para HTML5 en las etiquetas vacías. Entonces, debemos recordar que HTML5 es la continuación de HTML4. No obstante, las dos formas de escritura son válidas bajo el **doctype** de HTML5. En líneas generales, la sintaxis de HTML5 es más flexible y es

correcto usar tanto la sintaxis que ya conocimos de HTML4 como la que aprendimos de la mano de XHTML 1.0.

Lo recomendable es utilizar la sintaxis de HTML4 para las etiquetas vacías cuando desarrollemos un proyecto nuevo en HTML5, ya que éstas aportan menos caracteres y, por ende, el resultado final de archivo será más liviano. En cambio, si estamos migrando un sitio de XHTML a HTML5, podemos dejar sin problemas las etiquetas vacías con sus correspondientes barras de cierre.

Un caso similar se da en el uso de las comillas para encerrar los valores de los atributos: en HTML4, su uso de no es obligatorio, pero sí lo es en XHTML. Veamos, a continuación, un ejemplo concreto.

Esta sintaxis es válida para HTML4, pero no lo es para XHTML:

```
<img src=foto.jpg>
```

En cambio, la sintaxis del siguiente ejemplo es totalmente válida para XHTML y también lo es para HTML4:

```
<img src="foto.jpg">
```

Para el caso de HTML5, ambos ejemplos son válidos. Podemos usar o no comillas para los valores de los atributos, aunque la mayoría de los autores recomienda no perder la buena práctica del uso de las comillas que adoptamos de XHTML.

Otra diferencia relevante entre HTML4 y XHTML es la manera de anidar los elementos. En XHTML debemos ser mucho más cuidadosos con el orden de estos.

VISIÓN REDUCIDA

Si bien la definición de idioma de la página es importante por varios aspectos, lo es mucho más aún para aquellos usuarios con visión reducida que acceden con un lector de pantalla como **Jaws**. Esto se debe a que permitirá al programa conocer el idioma del sitio web y, por lo tanto, usar su pronunciación y sus reglas cuando haga la lectura del sitio, brindando, así, una mejor experiencia de usuario.

Veamos un par de ejemplos para entender mejor este punto. Esta forma de anidar las etiquetas es completamente válida para HTML4, pero no es correcta para XHTML:

```
<em><strong>Contenido</em></strong>
```

En el ejemplo siguiente, se encuentra la forma correcta de anidar los elementos para XHTML:

```
<em><strong>Contenido</strong></em>
```

Este orden de anidamiento es válido para HTML4 y –lógicamente– también lo es para HTML5. Igualmente, es aconsejable no perder las buenas prácticas que aprendimos con XHTML.

El uso de las mayúsculas y las minúsculas para la escritura de los elementos también es un punto donde HTML y XHTML van a tener diferencias de sintaxis.

En HTML4, es válido escribir los elementos y los atributos tanto en letra mayúscula como en minúscula, mientras que en XHTML es obligatorio hacerlo exclusivamente en minúscula.

De los siguientes ejemplos, ambos son válidos para HTML4, pero solo el segundo es correcto para el estándar de XHTML:

```
<A HREF=»http://www.unlink.com»>Ver masinfo</A>

<a href="http://www.unlink.com">Ver masinfo</a>
```

Una vez más, HTML5 admite ambos ejemplos, pero es preferible seguir los lineamientos de XHTML y escribir en minúsculas tanto elementos como atributos. En cuanto a los atributos booleanos, en XHTML no está permitido reducirlos. Es obligatorio escribir su valor, como podemos ver en ejemplo siguiente:

```
<textareareadonly=»readonly»></textarea>
```

El ejemplo que sigue es válido para HTML4 y para HTML5. De hecho, su uso es más que recomendable, ya que acorta bastante la cantidad de caracteres que usamos, hace más simple la escritura del código y, por supuesto, ayuda a reducir el tiempo de descarga del archivo:

```
<textareareadonly></textarea>
```

Otro cambio que podemos marcar es el uso de texto directamente en el cuerpo del documento. En HTML4 podemos incluir texto directamente en la etiqueta **<body>**, como se muestra en el siguiente ejemplo:

```
<body>Contenido de cuerpo de la página</body>
```

En el **doctype** estricto de HTML nos vemos obligados a incluir los textos dentro de elementos de texto:

```
<body><p>Contenido de cuerpo de la página</p></body>
```

Si bien en HTML5 no es obligatorio este segundo caso, es altamente recomendable realizarlo, para aportarle una mejor estructura semántica a nuestro documento.

También vale destacar otra pauta más a la hora de anidar elementos: en XHTML, los elementos de línea nunca pueden contener elementos de bloque.

ETIQUETAS DE LÍNEA Y DE BLOQUE

Todas las etiquetas de HTML pueden ser separadas en dos grandes grupos de etiquetas: de bloque y de línea. Las primeras siempre ocupan todo el ancho del elemento que las contiene; por ello, no puede haber una etiqueta de bloque junto a otra. Las de línea ocupan el mínimo ancho posible y, por esta razón, se disponen una junto a otra, siempre y cuando el contenedor tenga espacio suficiente para ellas. En términos generales, una etiqueta de línea no puede contener nunca una de bloque.

No es correcto anidar una etiqueta de título dentro de una etiqueta de énfasis, ya que la primera es de línea y la segunda de bloque:

```
<em><h1>Texto para el título principal</h1></em>
```

No obstante, es totalmente válido y correcto anidar una etiqueta de énfasis dentro de una de título, dado que la primera es de bloque y la segunda es de línea:

```
<h1><em>Texto para el título principal</em></h1>
```

Si bien este lineamiento se mantiene en HTML5, en el **Capítulo 3** veremos con mayor nivel de detalle este tema.

Un último punto muy importante a tener en cuenta es el uso de la etiqueta **<font>**. Si bien en HTML4 ya no se recomendaba su uso, en XHTML esta etiqueta no forma parte de los elementos válidos. En su lugar, podemos usar el atributo **style**:

```
<span style="color: #0000FF;">Texto del color indicado en el style</span>
```

Esta normativa es también válida para HTML5. De hecho, en esta especificación no es válido utilizar ningún atributo que tenga que ver con la representación visual: **todo estilo gráfico debe ser especificado desde el archivo de estilo CSS**.

Como hemos visto en estos puntos, las diferencias más relevantes entre las tres versiones del lenguaje las podemos notar en que HTML5 toma lo mejor de HTML4 y XHTML. Por esta razón, podemos decir que su sintaxis es mucho más flexible y, por supuesto, menos estricta. Esta ambigüedad está basada en la intención de que los desarrolladores puedan migrar los sitios escritos en HTML4 o XHTML a HTML5, de una manera rápida.

Por esta razón, debajo del **doctype** de HTML5 es correcto emplear la sintaxis de ambas versiones del lenguaje. Por cuestiones de prolijidad en el código, se aconseja ser detallistas y no mezclar una y otra sintaxis.

Esta flexibilidad en la sintaxis de HTML5 es particularmente útil cuando estamos migrando un sitio de gran envergadura. En este caso, nos llevaría mucho esfuerzo modificar la sintaxis de cada una de las etiquetas, pero, como ya hemos visto, no es necesario. Solo basta con cambiar el **doctype** y... ¡listo! Fácil, ¿no?

Por último, debemos aclarar que modificar solamente el **doctype** de un archivo no lo transforma en un sitio HTML5, mágicamente, sino que permite que empecemos, paulatinamente, a utilizar las nuevas etiquetas y capacidades de HTML5 en nuestros sitios.

Atributos: menos es más

Con el objetivo de simplificar y minimizar el código de nuestros proyectos web, en HTML5 es posible eliminar algunos atributos que en XHTML resultan obligatorios. Veamos algunos ejemplos:

Etiqueta link

En la etiqueta **link**, que utilizamos para enlazar el archivo HTML con su correspondiente archivo de estilos, podemos omitir el atributo **type**. En XHTML, en cambio, estamos obligados a utilizar el atributo **type.**

```
<link rel="stylesheet" href="estilo.css" type="text/css" />
```

En HTML5, este atributo no es necesario, así que es recomendable que lo eliminemos para minimizar la cantidad de caracteres utilizados en el documento:

```
<link rel="stylesheet" href="estilo.css">
```

Etiqueta script

Podemos ver un caso muy similar con la etiqueta **script**, mediante la cual enlazamos un documento HTML con un archivo JavaScript. En XHTML, estamos obligados a utilizar el atributo **type.**

```
<script src="link-fixup.js" type="text/javascript"></script>
```

En HTML5, el atributo **type** puede ser omitido:

```
<script src="link-fixup.js"></script>
```

Etiqueta meta

En el caso de las etiquetas **meta**, también vamos a ver que algunos atributos ya no son necesarios. De esta forma, el código se hace más simple y compacto, reduciendo el peso del archivo y el tiempo de descarga del documento.

Si bien existe una gran cantidad de etiquetas meta, la que siempre debería estar en nuestros documentos es la que permite establecer el juego de caracteres que este usará.

En XHTML, la etiqueta meta para el **charset** (set de caracteres) se escribe de la siguiente manera:

> AL OMITIR ATRIBUTOS, SE REDUCEN EL PESO DEL ARCHIVO Y EL TIEMPO DE DESCARGA

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Mientras que, en HTML5, podemos escribirla de forma mucho más compacta, sin utilizar ningún tipo de atributo adicional:

```
<meta charset=»UTF-8»>
```

LAS FAMOSAS ETIQUETAS META

Es importante recordar que existe una gran diversidad de etiquetas meta. Todas ellas se incluyen en el encabezado del documento y poseen distintos fines. A algunas de ellas hoy ya se las considera obsoletas, pero otras siguen vigentes; por ejemplo, las que se utilizan para incluir metadata para los buscadores, para determinar la codificación de caracteres del archivo, para mencionar al autor del sitio, etcétera.

Los elementos que pasaron a la historia

En la especificación de HTML5 se incorporaron varias etiquetas nuevas, así como también otras tantas sufrieron modificaciones en su descripción y algunas otras fueron eliminadas para siempre del mundo de HTML.

Es importante entender que las etiquetas que fueron eliminadas no fueron elegidas por capricho, sino porque ya no tenían nada que hacer. Muchas están relacionadas con las etiquetas no semánticas que se utilizaban para dar al documento características gráficas. Hoy debemos manejar estas características exclusivamente desde el archivo de estilo y solo utilizar etiquetas referidas al contenido del documento.

Etiquetas deprecadas

Veamos cuáles son las etiquetas que ya no forman parte de la especificación de HTML5.

ETIQUETAS DEPRECADAS EN HTML5

ETIQUETA	DESCRIPCIÓN
<acronym>	Se utiliza para insertar acrónimos.
<applet>	Permite insertar applets.
<basefont>	Determina el tamaño de base de la tipografía.
<big>	Permite indicar que la tipografía es de mayor tamaño.
	Se utiliza para centrar contenidos.
<dir>	Se utiliza para indicar un directorio.
<font>	Permite controlar los atributos de las fuentes.
<frame>	Permite crear un marco.
<frameset>	Determina el conjunto de marcos.
<isindex>	Permite crear una campo de línea única.
<nonframe>	Permite añadir contenido alternativo para un marco.
<s>	Se utiliza para indicar texto que fue eliminado.

ETIQUETAS DEPRECADAS EN HTML5	
<strike>	Se utiliza para poner texto tachado.
<tt>	Representa texto de máquina de escribir.
<xmp>	Determina un bloque donde los elementos de HTML son ignorados.

Tabla 1. Etiquetas deprecadas en HTML5.

El mundo según CSS

Desde hace ya varios años es un estándar utilizar en el desarrollo de un sitio el lenguaje **CSS** (*Cascading Style Sheets*, es decir, Hoja de Estilos en Cascada) para controlar la representación visual de los elementos del documento HTML. Este lenguaje incluye aproximadamente 200 propiedades que permiten crear interfaces de usuario enriquecidas con mucha facilidad. De esta manera, se crea un archivo aparte del HTML con las propiedades que tendrán todos los objetos de nuestra página web, como dimensiones, color, tipografía, etcétera. El navegador se encargará de asignarle las características a cada elemento de la página web sobre la base de las propiedades definidas en la hoja de estilos.

CSS cuenta con tres versiones vigentes. **CSS3** es la más reciente, aunque ya se está trabajando en el desarrollo de la versión 4, que sumará nuevas propiedades y selectores. A lo largo de este libro iremos viendo las nuevas capacidades de CSS3 y las posibilidades que ofrece.

RESUMEN

En este capítulo hemos iniciado un viaje que nos conducirá por el conocimiento del lenguaje HTML5, CSS3 y JavaScript, y nos permitirá conocer en profundidad los beneficios de trabajar con estos lenguajes. En este primer bloque de contenido, hemos repasado la historia y algunas características básicas del lenguaje HTML. A continuación, hemos descubierto las diferencias entre las distintas especificaciones: HTML4, XHTML y HTML5. Pero aún nos queda un largo camino por recorrer.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione cuántos tipos de doctype existen para HTML5.
2. Explique cuál es la forma correcta de escribir la etiqueta para insertar imágenes en HTML5.
3. En HTML5, ¿es obligatorio el uso de comillas para encerrar los valores de los atributos?
4. ¿Es correcto escribir las etiquetas en letra mayúscula en HTML5?
5. ¿Es posible omitir el atributo **type** en la etiqueta **link**, en un documento HTML5?
6. Explique cómo se determina el idioma de archivo en HTML5.
7. Indique cuál fue el cambio sufrido en HTML5 para la etiqueta HTML.
8. Indique el año de finalización estimado para HTML5.

EJERCICIOS PRÁCTICOS

1. Descargar el archivo **tp1.zip**.
2. Descomprimir la carpeta para obtener los archivos fuentes.
3. Modificar el código HTML del archivo **index.html** y migrar este archivo a HTML5. Tratar de respetar todas las recomendaciones que vimos en este capítulo (por ejemplo, no escribir los elementos ni los atributos en mayúsculas).
4. Comprobar la sintaxis del archivo usando el validador de W3C.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*02 –

JavaScript avanzado

En este capítulo repasaremos los conceptos básicos de JavaScript, trabajaremos con el DOM, aprenderemos a utilizar la consola de desarrolladores y nos introduciremos en la programación con objetos y sus principales características. Además, descubriremos cómo utilizar JavaScript en todo su potencial y de forma eficiente, y aprenderemos cómo aplicarlo para poder trabajar de manera correcta en los capítulos subsiguientes.

- El lenguaje de programación de la Web 44
- Trabajar con el DOM 57
- JSON 67
- Resumen 73
- Actividades 74

El lenguaje de programación de la Web

Durante su evolución, JavaScript se fue consolidando como el lenguaje de programación de la Web. ¿Por qué decimos esto? Muy simple: es el único lenguaje de programación web que viene integrado y es soportado por todos los **browsers**, lo cual también lo convierte en uno de los lenguajes más populares de programación al momento de escribir este libro.

Si bien existen otros lenguajes de programación web a nivel cliente, ninguno, hasta el momento, viene directamente integrado para ser soportado por todos los navegadores. Por eso, podemos decir que **quien desee ser un programador web, deberá saber JavaScript**.

Una de las ventajas de JavaScript es que con poco conocimiento del lenguaje ya podemos trabajar y resolver diferentes planteos y necesidades para tener nuestro proyecto funcionando.

Muchas personas se introducen en el uso del lenguaje "inspirándose" o "descargando plugins" de sitios web de ejemplo, sin saber en realidad qué es lo que están haciendo, pero resolviendo su inconveniente y planteando una solución a sus problemas.

En la práctica profesional, es común encontrarse con gente que realizó una galería de fotos, efectos y animaciones en sus sitios web con JavaScript. El problema se presenta al preguntarles cómo lo hicieron o qué recursos utilizaron. La respuesta es simple: "Me bajé un script en JavaScript de internet". No está mal, pero a lo largo de este capítulo descubriremos por qué tenemos que conocer el lenguaje y cómo podemos hacer bien las cosas.

UN POCO DE HISTORIA

JavaScript es el lenguaje de programación web soportado por todos los navegadores web disponibles en el mercado. Su primera "demo" fue escrita en solamente diez días por **Brendan Eich** para Netscape, quien luego ayudó a la formación de la Fundación Mozilla. Actualmente, la mayoría de los sitios web utiliza este lenguaje de programación para "darles vida" y mejorar la experiencia del usuario.

El objetivo consiste en que no nos bajemos un script en JavaScript de internet solamente, sino que sepamos qué es ese plugin o herramienta que descargamos, para qué sirve y de qué manera extenderlo, mejorarlo, implementarlo y, por qué no, desarrollar uno propio.

Para eso, necesitamos repasar y revisar algunos conceptos generales, conocer un poco la historia del lenguaje, saber con qué herramientas contamos para desarrollar y adquirir conocimientos avanzados. Nos adentraremos, de este modo, en la teoría y en diversos temas relacionados con la programación avanzada con JavaScript.

Conceptos generales

El estándar que define JavaScript es la tercera edición de la publicación **ECMAScript Language Specification**. En este documento encontraremos todas las características del lenguaje y sus diferentes usos y conceptos.

> LA PRIMERA VERSIÓN DE JAVASCRIPT SE PUBLICÓ EN 1995, BAJO EL NOMBRE DE MOCHA

La primera versión de JavaScript fue publicada por **Brendan Eich** en 1995, y su primer nombre fue **Mocha**. Fue creado dentro de las oficinas de **Netscape**, tratando de cumplir con lo que sus directivos necesitaban: "un lenguaje de scripting que se viera como Java". La primera demo de este lenguaje fue desarrollada en diez días.

Netscape cambió el nombre de Mocha por **LiveScript** para hacerlo coincidir con otro producto que estaban lanzando, que sería similar a lo que hoy es PHP del lado del servidor, y que luego fracasó. A fines de 1995, Netscape y **Sun Microsystems** (ahora **Oracle**) se decidieron por un nombre comercial y cambiaron LiveScript por **JavaScript**. Este nombre fue el que finalmente se registró y es como lo conocemos actualmente.

Al ser una marca registrada, JavaScript debe ser utilizado bajo licencia. En la actualidad, los productos sacados por la Fundación Mozilla tienen autorización para utilizarlo. Microsoft también creó un lenguaje de scripting similar a JavaScript y lo lanzó con la versión 3 de su explorador, denominándolo **JScript**. Este lenguaje tiene escasa relación con el estándar ECMAScript, pero es un

lenguaje que evoluciona y se mantiene. Es por esto que tenemos que tener especial cuidado cuando trabajamos con ciertas propiedades o métodos, para hacerlos compatibles entre los navegadores.

Vista y conocida la historia de este lenguaje de programación, vamos a hacer un repaso por los conceptos básicos para ir introduciendo los nuevos y más avanzados.

Variables y tipos de datos

JavaScript no es un lenguaje tipado, es decir, no debemos indicar qué tipo de valores almacena una variable, a diferencia de otros lenguajes como Java, que sí lo requieren. Las variables se definen utilizando la palabra reservada **var.**

Antes de comenzar, es importante destacar que, en los ejemplos, utilizaremos comentarios. Estos estarán delimitados por **/*** y ***/**, o comenzarán con **//**, dependiendo de si son de una o varias líneas:

```
/*

Este es un comentario de múltiples líneas, no será interpretado por
        el navegador

*/

//Este es un comentario de 1 línea

//Creamos una variable
var capitulo;
```

EN EL SERVIDOR Y EN EL CLIENTE

Cuando hablamos de **lenguaje del lado del servidor** nos estamos refiriendo a un lenguaje de programación que se ejecuta en el servidor y no en nuestro navegador. Un ejemplo de esto es **PHP**. En cambio, los lenguajes de programación que se ejecutan en nuestro navegador son llamados **lenguajes del lado del cliente**, como es el caso de **JavaScript**.

```
/* Creamos una variable numérica y le asignamos un valor */
var capitulo = 2;

/* Creamos una variable de texto y le asignamos un valor */

var titulo = "JavaScript Avanzado";
```

Nótese que **podemos declarar las variables sin especificar de qué tipo son**: con solo asignarles el valor, el lenguaje ya reconoce de qué tipo son. Los nombres de las variables no deben ser palabras reservadas del lenguaje, ni deben comenzar con números.

TIPOS DE DATOS PRIMITIVOS

TIPO DE DATO	EJEMPLOS
String	"Esta es una cadena de texto"
Number	1, 1.0, -1, 32
Boolean	true, false
Null	
Undefined	var x;

Tabla 1. Tipos de datos primitivos en JavaScript.

JavaScript tiene **un solo tipo numérico de datos**, a diferencia de otros lenguajes de programación. Por ejemplo, no hay un tipo de datos separado para los enteros y otro para los que tienen decimales. Esto nos ahorra algunos dolores de cabeza al momento de trabajar con números.

El valor **NaN**, por ejemplo, es un **valor del tipo numérico**, que indica el resultado de una operación que no puede devolver un resultado normal. **NaN no es igual a ningún valor.**

Todo lo que no es un tipo de dato primitivo, en JavaScript es un **objeto**. Por ejemplo, los **arrays** son objetos. Más adelante, en este mismo capítulo, abarcaremos este tipo de datos en particular.

Antes de eso, debemos conocer una de las herramientas con la que vamos a pasar la mayor cantidad de tiempo revisando, corrigiendo y mejorando nuestro código: la **consola.**

La consola

A los programadores web nos interesa y nos gusta explorar las herramientas para desarrollar, y queremos que esta tarea sea lo más amena y placentera posible. Para ello, utilizaremos la herramienta conocida como **consola**.

La consola es el lugar donde el sistema produce **salidas** para hacer un seguimiento de todo nuestro programa sin que el usuario final lo note directamente, pues este no verá las salidas en pantalla ni en ningún lado donde interactúe.

Generalmente, la consola es utilizada al momento de desarrollar para conocer qué valor toman las variables o qué salida retorna alguna función durante la ejecución de nuestro script. Entre otras cosas, nos permitirá detectar errores y advertencias que se producen en tiempo de ejecución.

La consola está disponible en la mayoría de las versiones actuales de los navegadores web.

En Google Chrome e Internet Explorer 8 en adelante se activa presionando **F12** dentro del navegador y yendo a la solapa **Consola.** En otros sistemas operativos o navegadores que no pueden activarse mediante esta tecla, presionamos clic derecho y pinchamos **Inspeccionar elemento**, o elegimos la solapa **Consola.**

Para Firefox es recomendable instalar el complemento **FireBug**, el cual se puede descargar gratuitamente desde **www.getfirebug.com**. Nos agregará la consola a nuestro Firefox con un set de herramientas que nos permitirá trabajar con CSS y otros lenguajes relacionados con la programación web.

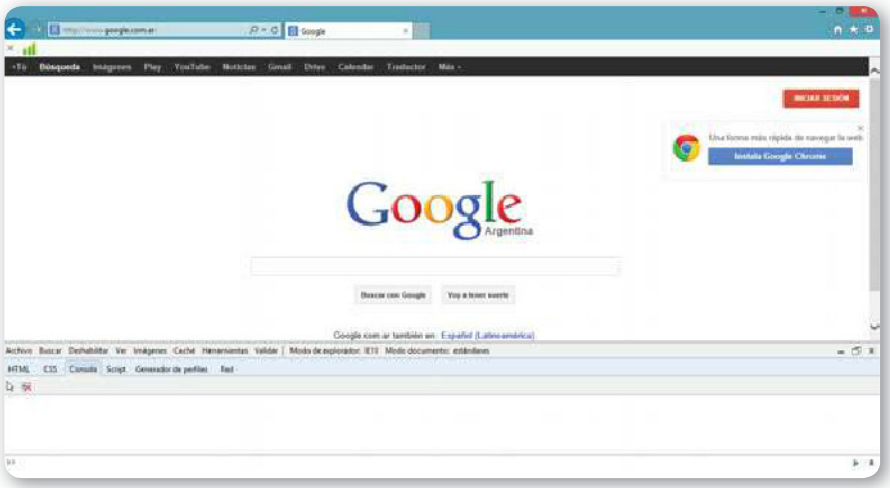


Figura 1. La consola en **Internet Explorer 10**.

Figura 2. La consola en **Google Chrome** (y en todos los navegadores con motor **WebKit**).

Figura 3. La consola en **Firefox**, utilizando el plugin **FireBug**.

Para enviar salidas a la consola con JavaScript, utilizaremos el comando **console.log()**. Este comando lleva como parámetro lo que queramos que salga por consola.

OTRAS HERRAMIENTAS

Junto con la consola, generalmente vienen otras herramientas muy útiles para el desarrollador. En Internet Explorer, por ejemplo, podemos cambiar la versión del navegador a ediciones anteriores para previsualizar el sitio sin necesidad de instalar otras versiones del navegador.

Veamos un ejemplo: creemos un nuevo archivo HTML llamado **consola.html** con el siguiente código:

```
<!doctype html>
<html>
<head>
<title>Probando la consola</title>
</head>
<body>
        <hl>Esta es una prueba de la consola, el usuario solo vera el ultimo valor
            de la variable <b>equipo</b>
</body>
<script>
console.log('Probando la consola con Users');
        //Muestra 'Probando la consola con Users'

        //Creamos la variable equipo
        var equipo;

        console.log(equipo);
        //Muestra por consola undefined

        //Le asignamos un valor
         equipo = 'Barcelona FC';

        console.log(equipo);
        //Muestra por consola "Barcelona FC"

        //Cambiemos el valor de la variable
        equipo = 'Mi equipo favorito es Bayern Munich';

        //Mostremos al usuario el contenido de la variable
        alert(equipo);

        </script>

        </html>
```

Debemos poner las etiquetas **<script></script>** al final del documento para que se muestre la alerta una vez mostrado el contenido del HTML. En caso contrario, se mostraría primero la alerta y luego el **<h1>**.

Al abrir el archivo **consola.html**, veremos que el navegador disparará una alerta con el mensaje "Mi equipo favorito es Bayern Munich". Nunca nos mostrará el valor "Barcelona FC", ya que este es solo mostrado por consola.

Figura 4. En esta imagen podemos observar el resultado que ve el usuario final de nuestro sitio.

Como ya vimos, para ver el comportamiento, los valores de la variable y los mensajes que indicamos que salgan por consola, deberemos mostrarla presionando **F12**.

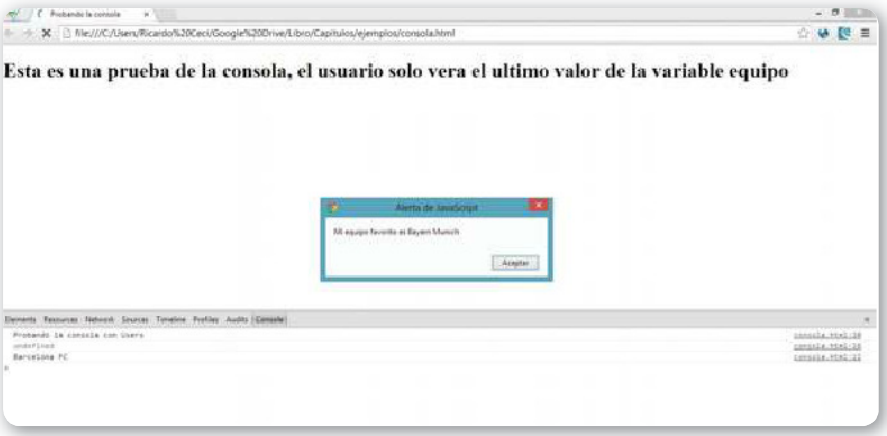


Figura 5. El resultado tal como lo vemos los programadores utilizando la consola.

Como podemos observar, la consola es una herramienta indispensable para el programador y muy útil al momento de poner en producción nuestro sitio.

Una vez terminado, debemos eliminar todo tipo de salidas por consola que hayamos puesto, ver que nuestro sitio no tenga errores en tiempo de ejecución y dejarlo listo para que la gente lo consuma.

Así como nosotros accedemos a la consola de los archivos, cualquier otro programador puede acceder a la consola de nuestro sitio en cualquier momento y ver lo que hayamos dejado ahí, lo cual no es muy profesional.

Muchos de los ejemplos que daremos en este capítulo y en los subsiguientes tendrán salidas por consola, que, gracias a esta sección, ya somos capaces de generar y visualizar.

Arrays

Supongamos que, dentro de una única variable, queremos guardar todos los meses del año y luego acceder a cada uno de manera individual sin traer el resto. Para poder realizar esta operación, necesitamos los **arrays.**

Un **array** (matriz o vector), técnicamente, es un contenedor en memoria para elementos generalmente homogéneos (los meses, por ejemplo), los cuales están ordenados en posiciones que nos permiten acceder individualmente a cada uno de ellos.

Ejemplos de arrays: meses, animales, frutas.

```
var meses = ["Enero","Febrero","Marzo"];
var frutas = ["Manzana","Naranja","Peras"];
var animales = ["Perro","Gato","Hipopótamo"];
```

Para trabajar con arrays en JavaScript, lo hacemos de la siguiente manera:

```
//Creación

var miArray = [1,2,3];
```

```
//Recuperamos un valor
console.log(miArray);
//1

//Modificamos un valor
miArray[2] = 'Esta es la última posición
n';

console.log(miArray);

//[1,2,"Esta es la última posición"];
```

Nótese que para declarar arrays también tenemos la **forma literal**: utilizamos los corchetes **[]** y, dentro, escribimos los valores separados por comas.

Para recuperar un valor, simplemente escribimos el nombre del array y, entre corchetes, indicamos la posición a la que queremos acceder, siendo la primera posición **0**. En nuestro ejemplo anterior, la posición **0** del array **miArray** tenía como valor el entero **1**, valor que será mostrado en la consola luego de ejecutar el comando **console.log(miArray[0]);**.

Reemplazar un valor en una posición del array

Como mencionamos al principio de este capítulo, los arrays también son **objetos** en JavaScript. Para comprobarlo, agregamos las siguientes dos líneas a nuestro script anterior:

```
typeof miArray;
//object
```

Esto quiere decir que, al crear un **array**, automáticamente ya contamos con las propiedades y métodos que el objeto **Array** de JavaScript nos provee.

Una propiedad es, por ejemplo, **length**, que nos permite saber la cantidad de elementos que posee un **array**.

```
var frutas = ["Naranjas","Manzanas","Peras"];

console.log(frutas.length); // 3
```

A continuación, veremos algunos métodos utilizados.

.toString()

Convierte el **array** en una cadena de texto.

```
var frutas = ["Naranjas","Manzanas","Peras"];

console.log(frutas.toString());
//Naranjas,Manzanas,Peras
```

.toLocaleString()

Convierte el **array** en una cadena de texto y permite ver cómo se mostrará, de acuerdo al idioma del navegador del usuario. La detección de la configuración de idioma del navegador se realiza automáticamente.

```
var frutas = ["Naranjas","Manzanas","Peras"];

console.log(frutas.toString()); // Naranjas,Manzanas,Peras
```

.join(x)

Une los valores del **array** y devuelve una cadena de texto unida por el separador **x**.

MÁS MÉTODOS Y EJEMPLOS

En este capítulo revisamos los métodos de JavaScript usados más frecuentemente en todo tipo de proyectos, pero también existen otros que pueden resultarnos útiles en proyectos puntuales. Podemos consultarlos en **ww.w3schools.com/jsref/jsref_obj_array.asp**, y probar las funcionalidades de cada uno.

```
var frutas = ["Naranjas","Manzanas","Peras"];

console.log(frutas.join('|')); // Naranjas|Manzanas|Peras
```

.push(x)

Inserta el elemento **x** al final del **array** y devuelve la nueva cantidad de elementos del **array.**

```
var frutas = ["Naranjas","Manzanas","Peras"];

frutas.push('Sandías');
//4

console.log(frutas);
//["Naranjas","Manzanas","Peras","Sandías"]
```

.pop(x)

Quita el último elemento del **array** y lo devuelve.

```
var frutas = ["Naranjas","Manzanas","Peras"];

frutas.pop();
//Sandías

console.log(frutas);
//["Naranjas","Manzanas","Peras"]
```

.shift()

Extrae el primer elemento del **array** y lo devuelve.

```
var frutas = ["Naranjas","Manzanas","Peras"];

frutas.shift();
//Naranjas
```

```
console.log(frutas);
//["Manzanas","Peras"];
```

.pop() y .shift()

Utilizados en conjunto, pueden ayudarnos a crear y manejar colas. Por ejemplo:

```
var listaDeEspera = ["Juan","Daiana","Cecilia"];

//Ingresa un nuevo cliente

listaDeEspera.push("Antonio");
// 4

//Llamamos al primero
alert(listaDeEspera.shift());
//"Juan"

console.log(listaDeEspera);
//["Daiana","Cecilia","Antonio"];
```

En la **Tabla 2**, podemos ver un resumen de los métodos y funciones. Para ejecutar cualquier método, basta con escribir **array.metodo();**.

MÉTODOS DEL OBJETO ARRAY DE JAVASCRIPT

MÉTODO	DESCRIPCIÓN
join ("separador")	Une todos los elementos del array devolviendo un string separado por el separador.
toString()/ toLocaleString()	Devuelven el array en formato de string y de string de acuerdo al idioma del navegador, respectivamente.
lastIndexOf(x)	Busca dentro del array el elemento X, comenzando desde el final del array, y devuelve su posición.

MÉTODOS DEL OBJETO ARRAY DE JAVASCRIPT	
pop()	Remueve un elemento del final del array y lo devuelve.
push(x)	Inserta el elemento X al final del array.
shift()	Remueve un elemento del array.

Tabla 2. Métodos del objeto **array**.

Trabajar con el DOM

El **DOM** es técnicamente una **API** (*Interfaz de Programación de Aplicaciones*, en inglés) que nos proporciona una serie de objetos para representar los elementos de los documentos HTML y XML. Al ser una API, esta puede ser accedida y utilizada por medio de JavaScript. A continuación explicaremos cómo utilizarla y combinarla con JavaScript.

Comencemos con un ejemplo:

```
<!doctype html>
<html>
<head>
<title>Trabajando con el DOM</title>
<style>
#titulares {
        font-weight:bold;
}
```

CONOCER MÁS SOBRE JAVASCRIPT

Las revisiones de la especificación del estándar **ECMAScript-262** podemos encontrarlas y descargarlas en **www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf**. De esta manera, encontraremos siempre las últimas novedades (ya que es la publicación oficial) y nos aseguraremos de la compatibilidad de nuestro sitio web con las futuras versiones de JavaScript.

```
.hombre {
        color:#367889;
}
.mujer {
        color:#CC3366;
}
</style>
</head>
<body>
        <div id="listados">
                <ul id="titulares">
                        <li  lass="mujer">Sara
                        <li  lass="mujer">Carolina
                        <li  lass="hombre">Ricardo
                </ul>
                <ul id="suplentes">
                        <li  lass="hombre">Nicolás
                        <liclass="hombre">Matias
                        <liclass="hombre">Roberto
                </ul>
</body>
</html>
```

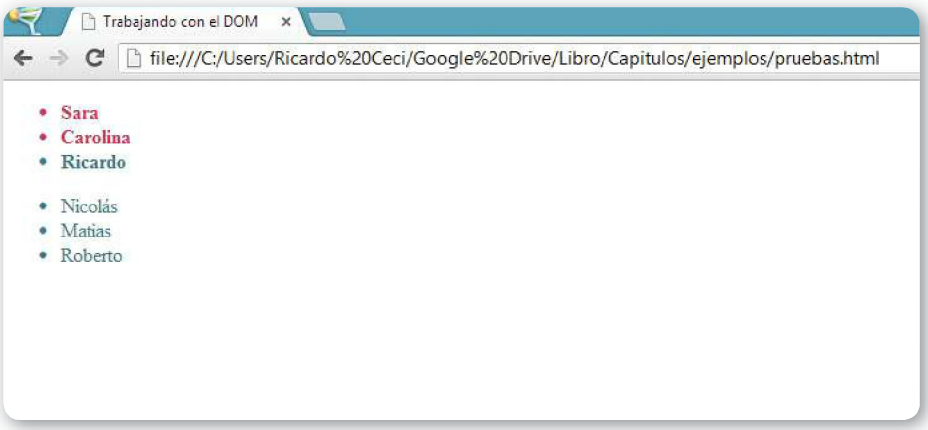


Figura 6. Salida en pantalla del documento HTML generado.

El DOM respeta una estructura jerárquica, tal como se puede observar en el diagrama siguiente:

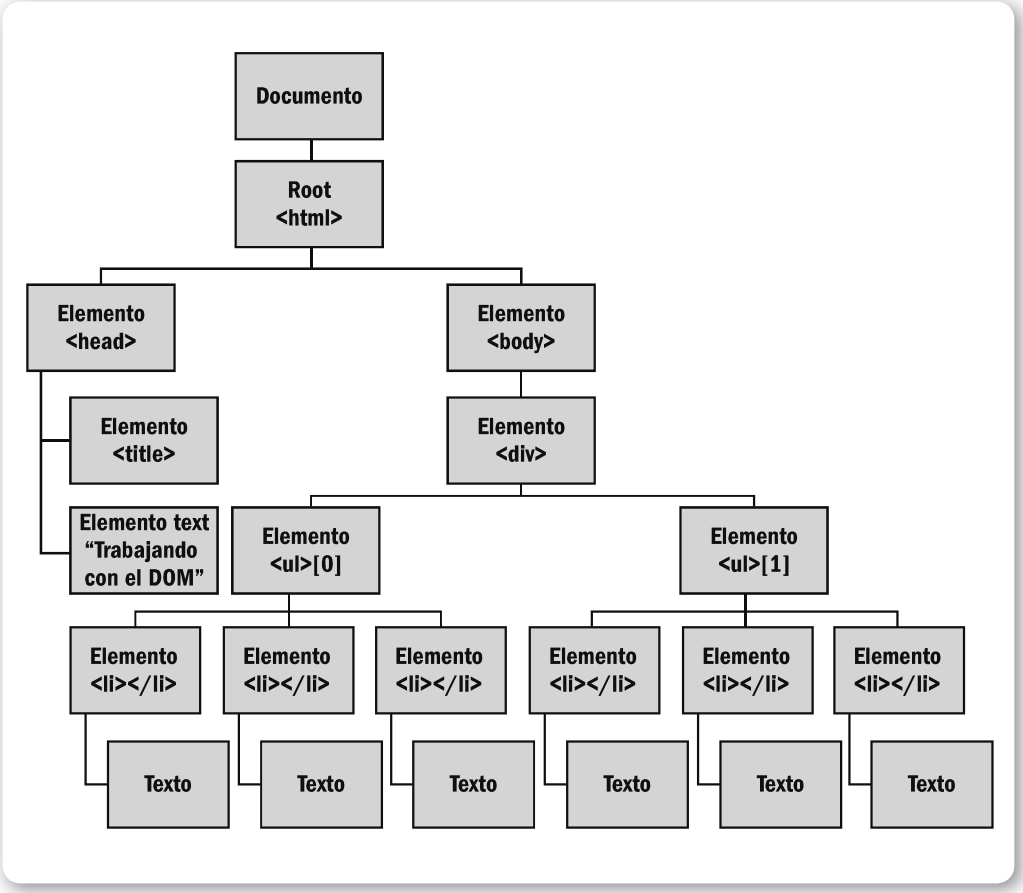


Figura 7. Estructura jerárquica del DOM.

Buscar elementos en el DOM

Para buscar elementos en el DOM, JavaScript nos provee de tres funciones, que veremos a continuación:

document.getElementById('id');

Esta función busca el primer elemento dentro del DOM con el atributo **ID** pasado como parámetro:

```
//Obtengamos la lista de titulares
var titulares = document.getElementById('titulares');

//<ul id="titulares"><li...
```

document.getElementsByTagName('tag');

Devuelve un **array** con todos los elementos que sean del tipo **tag.**

```
//Obtengamos todos los ítems de las listas
var jugadores = document.getElementsByTagName('li');

//[<li...,<li...,<li...
```

document.getElementsByClassName('clase');

Devuelve un **array** con todos los elementos que tengan cierta **clase.**

```
//Obtengamos todos los ítems que tengan la clase mujer
var jugadores = document.getElementsByClassName('mujer');

//[<li...,<li...,<li...
```

JavaScript ha añadido, además, dos nuevas formas de acceder a elementos utilizando los conocidos selectores de CSS: **document.querySelector** y **document.querySelectorAll.**

document.querySelector('#miID');

Buscará el primer elemento con el ID **miID.**

```
//Vamos a traer el listado de titulares

var titulares = document.querySelector('#titulares');
```

document.querySelectorAll('p');

Devolverá un **array** de elementos del tipo **p.**

```
//Vamos a traer los elementos del tipo ul
```

```
document.querySelectorAll('ul');

//[<ul id="titulares>...</ul>,<ul id="suplentes">...</ul>
```

document.querySelectorAll('.miClase');

Devolverá un **array** de elementos con la clase **miClase.**

document.querySelectorAll('p.miClase');

Devolverá un **array** de elementos del tipo **p** con la clase **miClase.**

Modificar estilos y propiedades de los elementos del DOM

JavaScript nos permite, de manera muy sencilla, cambiar el contenido del atributo **class** de un elemento, como así también el atributo **style**. Para hacerlo, tendremos que buscar el elemento, obtenerlo y luego modificar las propiedades **.style** o **.className.**

Veamos un ejemplo, tomando como referencia el HTML con el listado de titulares y suplentes que ya utilizamos en este capítulo. En este caso, les cambiaremos la clase a los diversos elementos de la lista.

> CON JAVASCRIPT PODREMOS MODIFICAR ELEMENTOS DINÁMICAMENTE

```
//Obtengamos todos los ítems de las listas
var jugadores = document.getElementsByTagName('li');

//Vamos a ponerle a todos los elementos, la clase "mujer"

for(i in jugadores){
        jugadores[i].className='mujer';
}
```

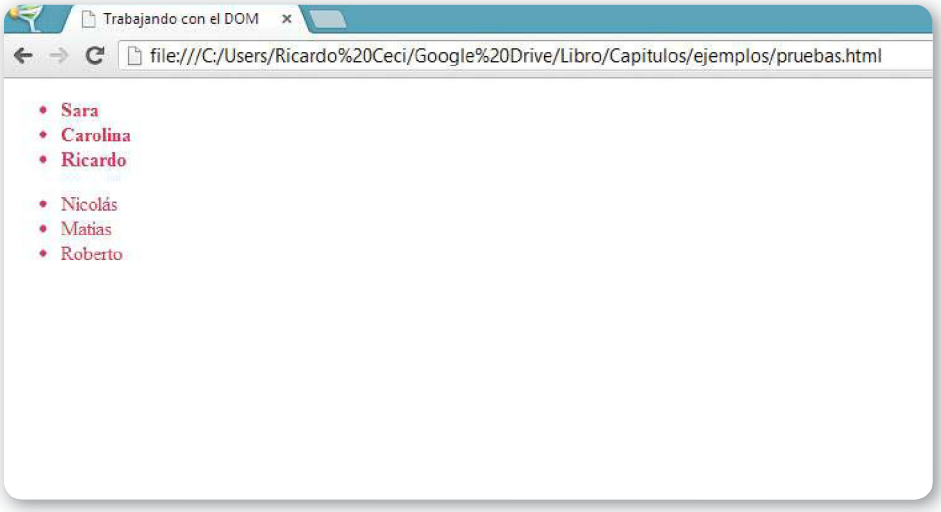


Figura 8. Todos los elementos de las listas pasaron a tener la clase **`mujeres`**.

```
//Obtengamos la lista de titulares
var titulares = document.getElementById('titulares');

//Vamos a ponerle un fondo a la lista de titulares
 titulares.style.backgroundColor = '#FFFF66';
```

Como vemos, modificar el atributo **style** es muy sencillo. Hay que tener en cuenta que, cuando queremos modificar alguna propiedad de estilos, tenemos que utilizar las mismas que utilizamos en CSS, pero con una diferencia: el guión (–) no está permitido en JavaScript para nombres de propiedades ni de métodos. Por ello, se utiliza un método llamado **camelCase** de palabras: se escribe el nombre de la primera palabra en letra minúscula y la siguiente palabra debe

UNIFICAR SELECTORES

Mediante la evolución y la aparición de los diversos lenguajes de programación web, se busca simplificarle la tarea al programador. Ahora, con el método **querySelector** del objeto **document**, podemos obtener elementos a través de los selectores de CSS.

comenzar con mayúscula. A continuación, veamos una tabla con algunos ejemplos de propiedades y su equivalente:

EQUIVALENCIAS ENTRE CSS Y JAVASCRIPT

CSS	JAVASCRIPT
background-color	backgroundColor
font-size	fontSize
font-weight	fontWeight
border-right	borderRight

Tabla 3. Propiedades de CSS y su equivalente en JavaScript.

Crear elementos

JavaScript nos permite trabajar y crear elementos que luego, dinámicamente, insertaremos en el documento. Partiendo del ejemplo de los titulares y suplentes, agregaremos nuevos titulares a la lista cuando el usuario complete un formulario. Para ello, repetimos el código y agregamos nuevos elementos para trabajar.

```
<!doctype html>
<html>
<head>
<title>Trabajando con el DOM</title>
<style>
#titulares {
        font-weight:bold;
}
.hombre {
color:#367889;
}
.mujer {
        color:#CC3366;
}
</style>
```

```
</head>
<body>
	<div id="listados">
		<ul id="titulares">
			<li  lass="mujer">Sara
			<li  lass="mujer">Carolina
			<li  lass="hombre">Ricardo
		</ul>
		<ul id="suplentes">
			<li  lass="hombre">Nicolás
			<li  lass="hombre">Matias
			<li  lass="hombre">Roberto
		</ul>
	<form>
		<label for="nombre">Nombre:</label>
<input type="text" name="nombre" id="txtNombre"/>
	<label for="genero">Genero:</label>
		<select name="genero" id="selGenero">
			<option>hombre</option>
			<option>mujer</option>
		</select>
	<label for="tipo">Tipo:</label>
<select name="tipo" id="selTipo">
		<option>titulares</option>
			<option>suplentes</option>
</select>

	<input type="button" onClick="agregar()" value="Agregar"/>
	</form>
</body>
</html>
```

Veamos la función **agregar()**, a la cual llamamos desde el botón al momento de presionarlo:

```
function agregar(){
```

```
//Busco el valor que ingresó el usuario en el campo nombre

var nombreIngresado = document.getElementById('txtNombre').value;

//Verificamos que el nombre no esté vacío
if(nombreIngresado !== ''){

//Creamos un elemento del tipo "li"
var item = document.createElement('li');
//Insertamos dentro de la etiqueta li el nombre ingresado
item.innerHTML = nombreIngresado;

//Le ponemos la clase de acuerdo al sexo seleccionado
var selGenero = document.getElementById('selGenero');
item.className = selGenero.options[selGenero.selectedIndex].text;

//Lo insertamos en la lista

var selTipo = document.getElementById('selTipo');
var listaSeleccionada = document.getElementById(selTipo).options[selTipo.select
        edIndex].text;

document.getElementById(listaSeleccionada).appendChild(item);
}
else {
        //Mostramos mensaje indicando el error
        alert("Por favor ingrese un nombre");
}

}
```

Para crear un elemento en JavaScript, utilizamos la función **document.createElement** y almacenamos lo creado en una variable.

Luego, esta variable tiene todas las propiedades de un elemento HTML capturado desde JavaScript. En este caso, por ejemplo, le ponemos una clase de acuerdo a lo que se haya seleccionado en la lista desplegable de género.

Mediante el atributo **.options[elemento.selectedIndex].text**, obtenemos el texto que está entre **<option>** y **</option>**.

Objetos con JavaScript

Ahora vamos a focalizarnos en este tipo de dato. Un **objeto**, propiamente dicho, es un contenedor de propiedades y métodos, donde cada propiedad tiene un nombre y un valor que puede ser cualquier tipo de dato de JavaScript, a excepción de **undefined**.

```
//Creamos un objeto vacío
var objeto = {};

//Creamos un objeto con propiedades

var jugador = {
        'nombre':'Martín',
        'apellido':'Palermo',
        'precio-venta':1000000
}

//Recuperamos datos de un objeto

//Martín
console.log(jugador.nombre);

//Palermo
console.log(jugador.apellido);

//undefined
console.log(jugador.edad);
```

Para crear un objeto, podemos hacerlo de varias maneras; en este caso, utilizamos las llaves **{}**. Esta forma de declarar objetos en JavaScript es denominada **literal**.

Dentro de estas llaves, escribimos las propiedades y los valores que contendrá dicho objeto, de la siguiente forma: **propiedad:valor**,

separadas cada una por una coma. Los nombres de las propiedades pueden ser cualquier palabra, a excepción de las reservadas. Las comillas para definir el nombre son obligatorias solamente si el nombre no es válido; por ejemplo, serán obligatorias para **'precio-venta'**, pero no para **precio_venta**, ya que el primero no es un nombre válido en JavaScript y el segundo sí lo es.

Cuando tratamos de acceder a una propiedad de un objeto que no declaramos previamente, esta toma el tipo de dato **undefined.**

Agregar propiedades a nuestros objetos

En JavaScript es muy sencillo agregar propiedades o métodos a nuestros objetos:

```
var persona = {};

persona.nombre = 'Silvana';
persona.sexo = 'Femenino';
persona.saludar = function() {
        return "Hola a todos";
}
```

Directamente tenemos que escribir la variable, un punto y el operador de asignación; de esta manera, estaremos agregándole a nuestro objeto nuevas propiedades y métodos.

JSON

Como hemos visto anteriormente, podemos declarar de forma literal tanto los **arrays** como los **objetos** propiamente dichos; es más, también podemos juntarlos.

A esta notación la llamamos **JSON** (*JavaScript Object Notation*). Para poder pronunciar bien "JSON", debemos acordarnos del personaje

asesino de las películas de terror **Viernes 13**: Jason. Se pronuncia de la misma manera. Veamos un ejemplo de su uso:

```
var jugador = {
        'nombre':'Martin',
        'apellido':'Palermo',
        'precio-venta':10000000,
        'clubes':['Boca Juniors','Estudiantes','Villareal'],
datos_deportivos: {
                debut:1992,
                goles:306
}
}
```

JSON es texto plano, fácil de leer por humanos y, a la vez, fácil de interpretar por los lenguajes de programación. Por esta razón y por ser tan liviano, se ha convertido en un formato para intercambiar información no solo a través de JavaScript, sino también a través de varios lenguajes de programación, como PHP, C, C++, etcétera.

Existen miles de servicios web y de servicios de terceros que proveen los datos no solo en formato XML, sino también en JSON. Por ejemplo, Facebook tiene un servicio de datos en formato JSON. Si ingresamos a **www.facebook.com/redusers** podemos ver, en formato web, la Fan Page de **RedUSERS**.

Supongamos, ahora, que queremos ver los datos en formato JSON para luego interpretarlos y consumirlos con JavaScript. Ingresamos a **https://graph.facebook.com/redusers** y obtendremos la información en formato JSON, lista para ser utilizada.

JSON EN FACEBOOK

Facebook nos muestra toda esta información en formato **JSON** gracias a la **Graph API**, una herramienta que pone esta red social a disposición de los desarrolladores, para que puedan obtener información y trabajar con los perfiles de Facebook de los usuarios y empresas. Para más información sobre este tema, recomendamos visitar el sitio de desarrolladores de Facebook: **https://developers.facebook.com**.

Al ser tan liviano, JSON es el preferido por los desarrolladores de aplicaciones para celulares, para enviar y recibir información de los servidores.

Nosotros utilizaremos esta notación, de aquí en adelante, para declarar y trabajar con objetos y arrays.

Por ejemplo, será utilizado para obtener datos de proveedores de terceros, cuando implementemos geolocalización integrada con **Google Maps**. Los datos los recibiremos en este formato, los interpretaremos y sabremos de qué se trata cada uno de los ítems y elementos que recibamos.

Parsear JSON en JavaScript

Cuando en programación hablamos de **parsear**, estamos hablando de analizar e interpretar –de acuerdo a determinadas reglas– una entrada y generar una representación de lo interpretado (salida) en forma de estructura de datos.

Al ser JSON parte de la notación literal de objetos en JavaScript, podemos utilizarlo en nuestro desarrollo sin mayores inconvenientes, tipeando directamente el texto y guardándolo dentro de una variable.

Hay dos formas para convertir un **string** de JSON en un **objeto** de JavaScript: utilizando el comando **eval()** o, en los navegadores que lo soporten, usando el **parser** nativo de JSON, mediante la función **JSON.parse.**

El comando **eval()** se encarga de llamar al compilador de JavaScript. Es muy rápido y puede compilar y ejecutar cualquier código JavaScript que le sea pasado como parámetro. Por ello, debemos tener especial cuidado al utilizar esta función, ya que no siempre el código que se le pasa puede tener buenas intenciones y producir el efecto esperado.

DEBUGGING JAVASCRIPT

Aprender a usar la consola y sus herramientas nos hará más profesionales y nos ayudará a realizar un debugging profesional de JavaScript, permitiéndonos agregar breakpoints, cambiar el User Agent, la versión del explorador y editar código en tiempo de ejecución. Para probar las novedades, debemos usar la versión Canary de Chrome: **www.google.com/intl/es/chrome/browser/canary.html**.

```
//Creamos un objeto

var persona = {
        'nombre': 'Juan',
        'apellido': 'López',
        'edad': 36
}

var objPersona = eval(persona);

console.log(objPersona.saludar("Ricardo"));
```

Es probable que muchas veces se produzca un error al intentar pasar un **string** de JSON por la función **eval**, ya que probablemente alguna propiedad o valor entre en conflicto con JavaScript. Para solucionar este problema, si el texto en JSON está bien armado, lo que debemos hacer es pasar el texto de la siguiente manera:

```
var miObjeto = eval('('+textoJSON+')');
```

Utilizamos los paréntesis concatenándolos junto con el texto en JSON para evitar errores en el compilador.

Otra forma consiste en utilizar el **parser** de JSON nativo del browser **JSON.parse(textoJSON)**, que está presente desde la versión 8 de Internet Explorer, 21 de Firefox, 26 de Chrome, 5.1 de Safari, 15 de Opera, 4.0 de Safari para iOS, 5.0 de Opera Mini, 2.1 del Android Browser y 7.0 del Blackberry Browser. Para versiones anteriores, debemos recurrir a la función **eval()**. Ejemplo:

```
//Creamos un objeto

var persona = {
        'nombre': 'Juan',
        'apellido': 'López',
        'edad': 36
```

```
}
//Parseamos el texto en JSON

if(JSON && JSON.parse) {

//Browsers compatibles
var objPersona = JSON.parse(persona);
}
else {
var objPersona = eval(persona);
}
console.log(objPersona.saludar("Ricardo"));
```

El uso de **JSON.parse** nos evita ejecutar código JavaScript malicioso, ya que chequea los tipos de datos recibidos y ejecuta una serie de validaciones antes de pasar al compilador los datos.

Programación basada en prototipos

En JavaScript no existe el concepto de "clase" como sí ocurre en otros lenguajes de programación que trabajan con el paradigma de Orientación a Objetos (POO). En JavaScript, un objeto es un contenedor de propiedades, métodos y, también, en este caso, de otros objetos, a los cuales vamos a poder acceder a través de una clave específica.

Todos los objetos en JavaScript están vinculados a un **prototipo**; es decir, cada vez que creamos un objeto, este automáticamente está vinculado a **Object.prototype**. Este prototipo viene por defecto en el lenguaje JavaScript. Ahora bien, **¿qué es un prototipo?** Se trata de un objeto del cual otros objetos heredan sus propiedades.

> JAVASCRIPT NO IMPLEMENTA POR DEFECTO LA POO, SINO QUE UTILIZA PROTOTIPOS

¿Cómo funciona este vínculo? Cada vez que queremos acceder a una propiedad de un objeto y no existe, JavaScript la busca en su prototipo. Si tampoco existe ahí, va a buscar en el prototipo del prototipo y así continuará con la cadena hasta que encuentre un valor del tipo **null**, y retornará **undefined.**

Excepción TypeError

Veremos este tipo de excepción, que es la que suele ocurrir con mayor frecuencia cuando trabajamos con el modelo de prototipos en JavaScript. Veamos un ejemplo:

```
//Creamos un objeto

var jugador = {
        'nombre':'Martín',
        'apellido':'Palermo',
        'precio-venta':1000000,
        posicion:'Director Técnico',
        datos_deportivos: {
                debut:1992,
                goles:306,
                ultimo_partido: {
                        fecha:'18-06-2011',
                duracion:90,
                rival: 'Gimnasia y Esgrima de La Plata',
                resultado: '2-2'
   }
}
}
//Recuperemos datos de un objeto

//1992
console.log(jugador.datos_deportivos.debut);

//90
console.log(jugador.datos_deportivos.ultimo_partido.duracion);

//undefined
console.log(jugador.datos_academicos);

//excepción TypeError
console.log(jugador.datos_academicos.universidad);
```

En este caso, creamos un objeto con datos del jugador y, dentro de este objeto, tenemos otro objeto con datos deportivos de este jugador, que a la vez tiene otro objeto dentro.

Cuando queremos acceder a los datos académicos del jugador, nos devuelve **undefined**. Hasta aquí, la ejecución de nuestro código continúa con total normalidad, ya que estamos intentando acceder al valor de una propiedad aún no definida. Pero cuando queremos acceder a una propiedad de una propiedad no definida, nos aparece una **excepción** que indica: **no se puede acceder a la propiedad 'universidad' de 'undefined'.** Esto detiene la ejecución de nuestro código, y no podremos continuar ejecutándolo hasta tanto no solucionemos este problema. Esta excepción puede visualizarse desde la consola.

Para evitar este tipo de errores, primero debemos chequear que la propiedad esté definida previamente, por ejemplo, de la siguiente manera:

```
if(jugador.datos_academicos){
        console.log(jugador.datos_academicos.universidad);
        }
```

Aquí, chequeamos previamente que la propiedad **datos_academicos** exista y, luego, tratamos de acceder a la propiedad **universidad** dentro de **datos_academicos**. Tenemos que prestar especial atención cuando queremos acceder a los datos dentro de un objeto, ya que, si la propiedad retorna **undefined** y tratamos de consultar una propiedad de la primera, se arrojará una excepción del tipo **TypeError**. Las excepciones de este tipo son arrojadas **cuando el valor recibido es distinto del esperado**.

RESUMEN

JavaScript es "el lenguaje de la Web", tiene mucho potencial y es sencillo de aprender, pero a la vez es complejo de entender. En este capítulo vimos en detalle herramientas que utilizaremos a lo largo del libro para trabajar con las APIs de HTML5 de manera profesional. Además, aprendimos a revisar y usar herramientas que nos ayudarán en nuestras tareas diarias y nos introdujimos en la programación con objetos y herencias de prototipos en JavaScript para comprender cuestiones avanzadas de este lenguaje.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione dos tipos de datos primitivos de JavaScript.
2. ¿Cómo se declara una variable?
3. ¿Cómo se inicializa un array?
4. Mencione dos formas de buscar elementos en el DOM.
5. ¿Cuál es el concepto de Objeto?
6. ¿Qué es JSON?
7. ¿Podemos utilizar JSON en otros lenguajes de programación?
8. ¿Para qué sirve la consola?

EJERCICIOS PRÁCTICOS

1. Realice un formulario en HTML con tres campos y cámbiele el fondo a rojo si no está completo algún campo.
2. Envíe a la consola las evaluaciones parciales de la validación del formulario del punto anterior (por ejemplo: el nombre está vacío).
3. Cree un objeto en JavaScript para gestionar una colección de libros.
4. Realice un sistema de turnos en JavaScript armando una pantalla para solicitar turnos con dos colas (Ventas y Soporte Técnico, por ejemplo) y con dos botones para llamar a los que estén primero.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*03 –

La revolución de HTML5

En este apartado, nos adentraremos en la Web semántica, para entender su importancia y conocer los nuevos aportes de HTML5 en este aspecto. Veremos en detalle cada una de las nuevas etiquetas y atributos de HTML5, así como también revisaremos aquellas etiquetas cuya descripción fue modificada para esta versión del lenguaje.

- La Web semántica 76
- Anatomía de un sitio web 83
- Nuevos elementos de secciones 86
- Nuevos elementos de agrupamiento en HTML5 106
- Nuevos elementos semánticos de texto 111
- Los elementos que fueron modificados 122
- Atributos globales de HTML5 128
- Resumen 131
- Actividades 132

La Web semántica

Seguramente, el término **Web semántica** nos resulte familiar. De hecho, es probable que lo hayamos repetido una y otra vez, aunque tal vez no terminemos de entender por completo lo que implica realmente la Web semántica y cuál es, en efecto, su potencial. He aquí una perfecta oportunidad para repasar brevemente algunos aspectos conceptuales de la Web semántica.

¿Cómo nace?

Su origen se remota al origen mismo de internet. El precursor de esta idea fue el propio **Tim Berners–Lee**, a quien conocemos como el creador de la World Wide Web (WWW). Ya desde la primera versión de HTML, Tim Berners–Lee había intentado incluir información semántica en los elementos del lenguaje pero, por alguna razón, las condiciones no estaban dadas aún por aquellos años para que la Web semántica pudiese mostrar su brillo y enamorara a cientos de desarrolladores.

Tuvieron que pasar varios años para que se iniciara el proceso de evolución en la Web y, con esto, la forma de acceso a la información.

¿Qué es la Web semántica?

Se trata de una nueva generación de la Web, dotada de mayor significado; es decir, es una versión extendida de la Web que posee mayor información accesible a todos los usuarios. Podríamos resumirlo en lo siguiente: llamar a las cosas por lo que las cosas son realmente.

Estamos acostumbrados a acceder a la Web por medio de la vista, nuestro principal y más valioso sentido para acceder a la información. Muchas veces olvidamos que existen otros usuarios que acceden a diario a la Web por medio de otros sentidos, como, por ejemplo, el oído. Estamos hablando, por un lado, de personas ciegas, pero también de cientos de robots que acceden a diario a nuestros sitios web en busca de información específica para las búsquedas de los usuarios. Para todos ellos, la Web semántica es la respuesta a sus necesidades.

Antes de avanzar, veamos un ejemplo básico de semántica para dejar más claro de qué estamos hablando. Supongamos que nos

piden desarrollar un portal de noticias donde se publicarán, a diario, contenidos editoriales. Cada noticia deberá contener los siguientes elementos:

- Título
- Volanta
- Copete o bajada
- Texto principal

Visualmente, se deberá ver tal como observamos en la siguiente figura:

Figura 1. Diseño editorial del formato de noticias que deberemos maquetar para nuestro sitio web.

Entonces, podríamos pensar en realizar un código HTML como el que podemos ver a continuación:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Portal de noticias de deporte</title>
</head>
```

```
<body>
        <p class="volanta">EL CIRCUITO ATP</p>
        <p class="titulo">Andy Murray descansa con su novia en Bahamas, tras
                ganar Wimbledon</p>
        <p class="copete">Luego de ser el primer británico en 77 años en ganar el
                certamen, el campeón se relaja.</p>
        <p class="texto">A los 26 años, Andy Murray dio uno de los golpes más
                grandes al ser el primer británico en ganar Wimbledon, luego de 77
                años.
</p>
</body>
</html>
```

Este archivo HTML está asociado a un archivo de estilo mediante el cual se controla la representación visual del documento, y que posee las siguientes líneas de código:

```
@charset "UTF-8";
/* CSS Document */

.titulo {
font-family:Verdana;
font-size:24px;
font-weight:bold;
color:#333;
}

.volanta {
font-family:Verdana;
font-size:14px;
color:#ccc;
}

.copete {
font-family:Verdana;
```

```
font-size:18px;
color:#333;
}

.texto {
font-family:Verdana;
font-size:12px;
color:#333;
}
```

Entonces hagamos un pequeño análisis: el código que vimos recién, ¿es correcto? Si hablamos en términos de sintaxis, podríamos decir que sí: si lo probamos mediante el validador de W3C, este nos dirá que no presenta ningún error de validación. ¿Podemos hacerlo mejor? La respuesta es, también, sí. Aunque nuestro código HTML es sintácticamente correcto, no posee ninguna riqueza semántica, ya que se están usando etiquetas de párrafo para todos los contenidos de texto y clases para controlar su representación visual.

Veamos ahora cómo trabajar el código de forma semántica:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Portal de noticias de deporte</title>
</head>
<body>
        <h3>EL CIRCUITO ATP</h3>
        <h1>Andy Murray descansa con su novia
                en Bahamas, tras ganar Wimbledon</h1>
        <h2>Luego de ser el primer británico en 77 años en ganar el
                certamen, el campeón se relaja.</h2>
        <p>A los 26 años, Andy Murray dio uno de los golpes más grandes al ser
                el primer británico en ganar Wimbledon, luego de 77 años.<p>
```

```
</body>
</html>
```

En esta versión del código HTML estamos usando etiquetas de títulos y párrafos respectivamente.

Veamos cómo quedaría ahora el archivo de estilo:

```
@charset "UTF-8";
/* CSS Document */

h1 {
font-family:Verdana;
font-size:24px;
font-weight:bold;
color:#333;
}


h3 {
font-family:Verdana;
font-size:14px;
color:#ccc;
}

h2 {
font-family:Verdana;
font-size:18px;
color:#333;
}

p {
font-family:Verdana;
font-size:12px;
color:#333;
}
```

Veamos cuál es la diferencia entre los dos archivos HTML que desarrollamos: desde el punto de vista de la representación visual, notamos que no hay ninguna diferencia entre ellos, ya que, como podemos apreciar en los documentos de estilo, ambos poseen los mismos atributos de CSS y, por ende, se ven igual. El ser humano está acostumbrado a regirse por convenciones, aquellas que aprendimos desde pequeños en la escuela, donde conocimos cómo decodificar información editorial y entendimos que el título es una parte relevante del contenido, que siempre se encuentra arriba del texto y con características visuales de mayor peso tipográfico que el resto de los elementos de la noticia. Todos sabemos esto porque está incorporado en nuestro inconsciente, pero a un robot no le pasa lo mismo.

> GOOGLE PUEDE DISTINGUIR LOS NIVELES DE TÍTULOS TAL COMO HACEMOS NOSOTROS

Si Google accede a nuestro sitio para buscar información, entenderá que los dos archivos que le presentamos son completamente diferentes. En el primero verá **solo párrafos del mismo valor semántico**; por lo tanto, no podrá decodificar cuál de ellos es más importante que el resto. En cambio, en el segundo código HTML que le presentamos verá algo completamente distinto y **podrá entender claramente qué significado posee cada una de las partes del contenido**. Podrá distinguir los niveles de títulos, exactamente igual que lo hacemos nosotros mediante el color y los tamaños.

Entonces podemos arribar a la siguiente conclusión: la diferencia que existe entre estos documentos es que **ambos son sintácticamente correctos, pero el segundo, además, es semánticamente correcto**, mientras que el primero no lo es. El segundo documento podrá ser entendido del mismo modo por todos los usuarios que accedan a él, ya sea un ser humano o un robot; en cambio, el primero tendrá una lectura totalmente diferente para una persona que accede por la vista y para un robot o un lector de pantalla.

¿Se entiende ahora a qué se refiere la semántica? El uso a conciencia de los elementos de HTML es solo una arista de la Web semántica. A lo largo de este capítulo, iremos viendo que no solo se refiere a etiquetas sino que también incluye una serie de atributos no visibles para la vista humana, pero de suma importancia para los robots.

Beneficios de la Web semántica

Gracias a que la Web semántica está dotada de mayor significado, todos los usuarios podrán acceder de forma mucho más rápida y sencilla a la información que están buscando, con un mayor grado de precisión; no solo facilita el acceso por parte de los robots.

Es común que cuando realizamos búsquedas, la información que nos devuelven los motores de búsqueda no es exactamente lo que estábamos buscando. La Web semántica permite ponerle fin a estos problemas mediante el uso de **una infraestructura común con la cual es posible procesar y compartir información de forma más sencilla usando un mismo lenguaje**. Además, brinda una mejor experiencia, ya que mejora el acceso a la información para todos los posibles tipos de usuarios.

> LA WEB SEMÁNTICA PERMITE PROCESAR Y COMPARTIR INFORMACIÓN DE FORMA MÁS SENCILLA

Internet ha cambiado para siempre la forma en la cual concebimos la vida cotidiana en la actualidad. Para muchos, el día no se inicia en tanto no se hayan chequeado las distintas cuentas de e-mail, visitado las redes sociales y revisado los titulares de los principales diarios digitales. La forma en la cual nos comunicamos y relacionamos está cada día más integrada a lo que pasa en la Web. Día a día son más las compras y transacciones que podemos realizar por internet. La frontera entre el mundo físico y virtual es cada vez más difusa.

Supongamos que algunos de nosotros –seguramente, muchos– usamos el celular como despertador. El equipo podría estar conectado permanentemente a internet y acceder a toda la información que en ella se genera. Haciendo uso de la Web semántica, podría detectar algún problema relacionado con los medios de transporte y entender que debo levantarme más temprano para llegar a tiempo a mi trabajo.

La Web semántica permite delegar tareas y problemas cotidianos al software, haciendo un uso inteligente de los recursos y la información disponible en la red. Los sistemas pueden entender su contenido, procesar y combinar la información para finalmente hacer deducciones y tomar decisiones que ayuden a los usuarios a resolver problemas cotidianos de manera automática.

La Web semántica mejora la experiencia de los usuarios y, en consecuencia, su calidad de vida.

Semántica en HTML5

Después de tanto hablar de semántica, resulta casi obvia la pregunta: ¿qué impacto tiene HTML5 en la Web semántica? La respuesta es más que razonable: ¡impacto total!

Una de las principales mejoras introducidas por HTML5 es que posee una **mejor estructura semántica**. Introduce una serie de nuevas etiquetas que aportan mejoras a la semántica del documento. Además, algunas de las etiquetas que ya existían fueron especificadas nuevamente con el objetivo de mejorar aún más la semántica.

Anatomía de un sitio web

Cuando realizamos el diseño de la interfaz de un sitio web, sabemos que este debe tener una estructura determinada que se encuentra asociada al tipo de contenido que vamos a mostrar en el sitio web y también a la forma con la cual los usuarios navegarán dicha pieza. Veamos, a continuación, algunos ejemplos de sitios web conocidos por todos nosotros y observemos brevemente su estructura.

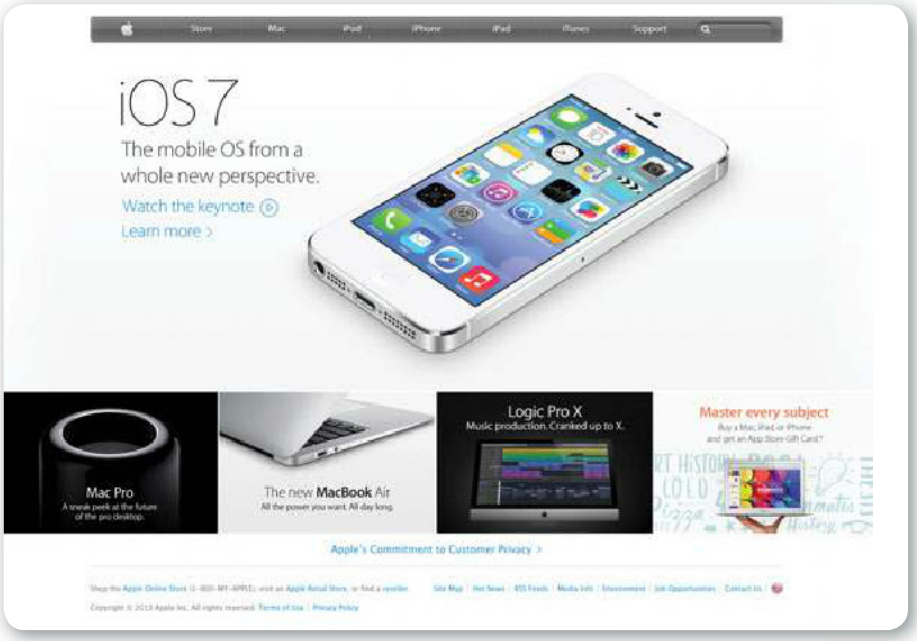


Figura 2. En la página principal (home) del sitio web de Apple se puede ver la estructura típica de un sitio web.

Figura 3. En la página de inicio del sitio web de Windows 8 apreciamos una estructura idéntica.

Podríamos estar todo el día analizando distintos sitios web y llegaríamos a la conclusión de que la mayor parte de ellos poseen una estructura similar, fácilmente reconocible, donde podemos distinguir cuatro bloques principales:

- Área del encabezado
- Área del menú de navegación

¿QUÉ ES LA ACCESIBILIDAD?

Entendemos por **accesibilidad en la Web** a la capacidad de un sitio web de ser accedido por todo tipo de usuarios, especialmente aquellos que poseen alguna discapacidad física, como ceguera o disminución motriz. Para ello, la W3C ha lanzado, hace varios años, una iniciativa mediante la cual se establece una serie de pautas a seguir para realizar un sitio accesible a distintos tipos de usuarios. Podemos encontrar más información sobre este tema en **www.w3c.es/Traducciones/es/WAI/intro/accessibility**.

- Área del contenido
- Área del pie de página

Sabiendo esto, a la hora de diseñar un nuevo sitio web, partimos siempre de una estructura básica que contempla contenedores para estos bloques principales en que se divide la página.

Figura 4. Estructura de base de un sitio web, en la cual se contemplan los bloques de contenido básico de un documento.

Este sería el código HTML que desarrollaríamos:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Título de mi sitio web</title>
</head>
<body>
        <div id="header"></div>
        <div id="nav"></div>
        <div id="main"></div>
        <div id="footer"></div>
</body>
</html>
```

¿Es correcto este código HTML? Claro que sí. Pero, a pesar de ser un código totalmente válido y bien estructurado, resulta obsoleto cuando hablamos de HTML5 y, sobre todo, de semántica. Si todos los documentos HTML poseen esta estructura básica, es necesario tener etiquetas específicas para cada uno de estos bloques de contenido.

La etiqueta ** ** significa división. Si bien es una de las etiquetas más utilizadas en XHTML, su uso es casi nulo en HTML5, ya que **no posee ningún valor semántico**. Lo único que nos dice es en cuántas partes se segmenta nuestro documento, pero no nos brinda ninguna información sobre el contenido específico que hay dentro de este.

En HTML5 encontraremos una serie de etiquetas que usaremos donde antes usábamos las etiquetas ** **. Maquetar un sitio en HTML5 puede parecer un poco más complejo al principio, ya que tenemos muchas más opciones que antes. Pero cuando nos acostumbremos a trabajar con ellas, notaremos que maquetar en HTML5 es más simple, más prolijo y optimiza el trabajo de los desarrolladores web.

Nuevos elementos de secciones

Es hora de poner manos a la obra. Por ello, conoceremos en profundidad cada una de las nuevas etiquetas de HTML5 y, luego, desarrollaremos una maquetación haciendo un uso apropiado de ellas. Las etiquetas que veremos a continuación pertenecen al grupo de las **etiquetas de secciones**, que se utilizan básicamente para **seccionar el documento** en partes, según su semántica.

TABLAS DE COMPATIBILIDAD

Existen muchos sitios web que contienen tablas de compatibilidad, en las cuales podemos verificar si las capacidades de HTML5 o atributos de CSS3 poseen o no compatibilidad en cada uno de los navegadores y sus distintas versiones. Entre los más conocidos, encontramos: **www.caniuse.com**, **www.html5please.com**, **www.css3please.com**, **www.html5demos.com** y **www.html5test.com**.

A continuación, hablaremos acerca de la forma de uso de cada una de las nuevas etiquetas, así como de la compatibilidad de cada una de ellas con los diferentes navegadores. Es recomendable verificar siempre que esta información sea precisa, mediante alguna tabla de compatibilidad –como podría ser **www.caniuse.com**–, dado que el soporte de los navegadores para HTML5 progresa día a día.

Crear encabezados con <header>

La etiqueta **<header>** es una etiqueta semántica que se utiliza para contener información relativa al encabezado de documento o bien de otras secciones.

Pertenece al grupo de las etiquetas de bloque y debe ser usada dentro de **<body>** o de otras etiquetas como **<section>** o **<article>**, que conoceremos más adelante en este mismo capítulo. Esta etiqueta puede ser usada tantas veces como sea necesario, pero se recomienda no abusar de su utilización.

Comúnmente vamos a encontrar, dentro de la etiqueta **<header>**, etiquetas de títulos (**<h1>**/**<h6>**), aunque su uso no es obligatorio. Veamos un ejemplo de cómo podemos utilizarla:

```
<header>
      <h1> Blog de HTML5 </h1>
      <h2> Toda la información sobre HTML5 </h2>
</header>
```

La etiqueta **<header>** también puede ser usada para contener la tabla de contenido de una sección, un formulario de búsqueda o bien logos relevantes. Veamos un ejemplo de su uso en el sitio web de W3C:

```
<header>
<h1>Scalable Vector Graphics (SVG) 1.2</h1>
<p>W3C Working Draft 27 October 2004</p>
<dl>
<dt>Esta versión:</dt>
```

```
<dd><a href="http://www.w3.org/TR/2004/WD-SVG12-20041027/">http:
    //www.w3.org/TR/2004/WD-SVG12-20041027/</a></dd>
<dt>Versión anterior:</dt>
<dd><a href="http://www.w3.org/TR/2004/WD-SVG12-20040510/">http:
    //www.w3.org/TR/2004/WD-SVG12-20040510/</a></dd>
<dt>Última versión:</dt>
<dd><a href="http://www.w3.org/TR/SVG12/">http://www.w3.org/TR/SVG12/
    </a></dd>
<dt>Última recomendación para svg:</dt>
<dd><a href="http://www.w3.org/TR/SVG/">http://www.w3.org/TR/SVG/
    </a></dd>
<dt>Editor:</dt>
<dd>Dean Jackson, W3C, <a href="mailto:dean@w3.org">dean@w3.org</
    a></dd>
<dt>Autores:</dt>
<dd>Ver <a href="#authors">Lista del autor</a></dd>
</dl>
</header>
```

Es importante destacar que la etiqueta **<header>** no introduce una nueva sección dentro del documento, dado que no pertenece a este grupo de etiquetas, como sí los son, por ejemplo, las etiquetas de título que van desde el **<h1>** hasta el **<h6>.**

La etiqueta **<header> es el encabezado de una sección ya existente**: si se encuentra dentro de la etiqueta **<body>**, entonces estamos haciendo referencia al encabezado del elemento raíz del documento; en el caso opuesto, lo será de su elemento padre. Veamos un ejemplo sobre las secciones y los encabezados:

```
<body>
<header>
<h1>Estudio de diseño Nueva Era</h1>
<!-- Acá comienza la primera subsección -->
<nav>
<ul>
<li><a href="servicios.html">Servicio</a>
```

```
<li><a href="portfolio.html">Portfolio</a>
<li><a href="filosofia.html">Filosofía</a>
</ul>
</nav>
<h2>Noticias</h2>
<!-- Acá comienza una nueva subsección -->
<p>Creamos una nueva unidad de negocios dedicada a SEO.</p>
<h2>Proyectos recientes</h2>
<!-- Acá comienza la tercera subsección -->
</header>
<p>Desarrollo de logo y web site</p>
<!-- Este contenido es de la subsección Proyectos recientes-->
```

El soporte de la etiqueta <header> está presente en todos los navegadores: Chrome, desde la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

> LA ETIQUETA HEADER SIRVE PARA INDICAR EL ENCABEZADO DE UNA SECCIÓN DEL SITIO WEB

Navegación más semántica con <nav>

La etiqueta **<nav>** se utiliza para contener la navegación principal de la página. Representa una sección que siempre debe contener enlaces dentro de la misma página o hacia otras páginas.

Veamos el siguiente ejemplo de código:

```
<nav>
    <h1>Navegación</h1>
    <ul>
    <li><a href="clientes.html">Clientes</a></li>
    <li><a href="productos.html">Productos</a></li>
    <li><a href="contacto.html">Contacto</a></li></ul>
</nav>
```

Figura 5. El código anterior se vería de esta manera en el navegador.

Si bien su uso no es obligatorio, dentro de la etiqueta **<nav>** podemos usar una lista, tal como habitualmente lo hacemos cuando maquetamos las barras de navegación. Esta etiqueta puede contener directamente los elementos de hipervínculo u otras etiquetas de estructuración, como se ejemplifica a continuación:

```
<nav>
<hl>Diseño web</hl>
    <p>El <a href="disenioweb.html">diseño web</a> es una actividad que consiste en la <a href="planificacion.html">planificación</a>, diseño e implementación de sitios web. No es simplemente una aplicación de diseño convencional, ya que requiere tener en cuenta la navegabilidad, interactividad, <a href="usabilidad.html">usabilidad</a>, <a href="ai.html">arquitectura de la información </a>y la <a href="interaccion.html">interacción</a> de medios como el audio, texto, imagen, enlaces y video.</p>
</nav>
```

MENÚS DE NAVEGACIÓN

Los sitios web son cada día más parecidos a las aplicaciones que tenemos en el escritorio de nuestras computadoras. Como muestra de esa tendencia podemos mencionar a las barras de navegación que aparecen en algunos sitios y tienen menús desplegables, donde se agrupan otras opciones a modo de atajo. Estos tipos de barras y menús se pueden obtener a partir del uso de CSS3, tal como aprenderemos hacia el final de este libro.

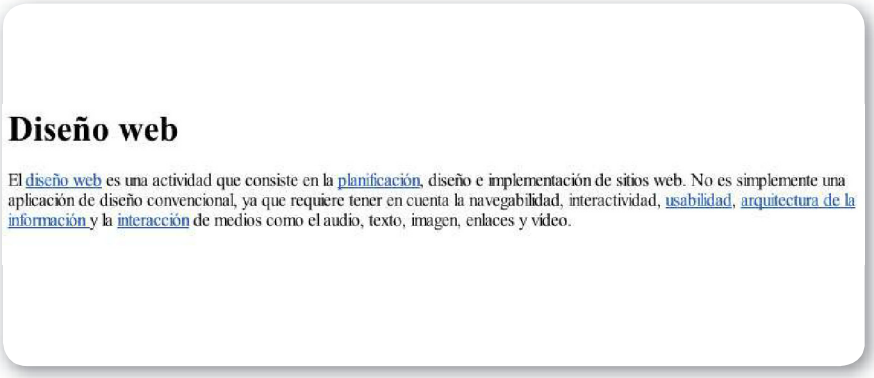


Figura 6. Así veríamos en el navegador el código que acabamos de probar.

Es importante que la etiqueta **<nav>** **sea usada solo para la navegación principal de la página**. Esto significa que no todos los grupos de enlaces deben estar contenidos en una etiqueta **<nav>.**

Es común que los sitios web tengan más de una barra de navegación. Un ejemplo típico es un listado de vínculos en el pie de página. En estos casos, no se recomienda contener los enlaces en una etiqueta **<nav>**, ya que con la etiqueta del pie de página resulta suficiente.

Si bien es perfectamente válido el uso de más de una etiqueta **<nav>** en cada documento, su utilización es poco recomendable, especialmente para usuarios que acceden mediante un lector de pantalla o para los motores de búsqueda. El uso de un único elemento **<nav>** les permitirá comprender mejor la navegación del documento y, de esta manera, brindar una mejor experiencia a los usuarios.

La etiqueta **<nav>** puede ser contenida dentro de una etiqueta **<header>**; de hecho, es una práctica bastante común.

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

Cierres con <footer>

La etiqueta **<footer>** se utiliza para contener la información relativa al pie de página o a una sección. **Siempre es relativa a su padre**, es decir, a la etiqueta en la cual se encuentra anidada.

Entre la información que suele contener, podemos mencionar: datos de contacto de la empresa, legales, mapa del sitio, datos del autor, enlaces a redes sociales, etcétera. Veamos un ejemplo de uso:

```
<footer>
<nav>
<p><a href="politica.html">Política de privacidad</a> —
<a href="terminos.html">Términos de uso</a> —
<a href="blog.html">Blog </a></p>
</nav>
<p>Copyright © 2013</p>
</footer>
```

La etiqueta **<footer>** puede ser usada tantas veces como se considere necesario. Si se encuentra dentro de la etiqueta **<body>**, se refiere al pie de página general del documento; mientras que, si está dentro de otra etiqueta –como por ejemplo **<section>** o **<article>**–, se refiere al pie de página de esa sección.

Un punto importante a destacar sobre la etiqueta **<footer>** es que no necesariamente debe aparecer al final de la sección que la contiene, aunque es bastante común su uso de esta manera.

Veamos un ejemplo donde se usa más de una etiqueta **<footer>** dentro de un mismo contenedor:

```
<body>
<h1>Preguntas Frecuentes</h1>
<footer><a href="#">Volver al inicio</a></footer>
<div>
<h2>Cómo realizar una compra</h2>
<p>Theipsum of all lorems</p>
</div>
```

HTML5 OUTLINER

Una interesante herramienta online es **HTML5 Outliner**, que nos permite cargar un archivo o una URL de los cuales extraerá el outline, es decir, la jerarquía de los títulos, para que podamos comprender la forma en la cual el archivo o URL serán interpretados por los robots. También nos permite introducir directamente el código HTML para conocer su outline. Se puede usar en: **http://gsnedders.html5.org/outliner**.

```
    <p>A dolor sit amet, consecteturadipisicingelit, sed do eiusmodtemporincididun-
tutlabore et dolore magna aliqua. Utenim ad minim  veniam, quisnostrud exercitation
ullamco labor<p>
    <footer><a href="#">Volver al inicio</a></footer>
    </body>
```

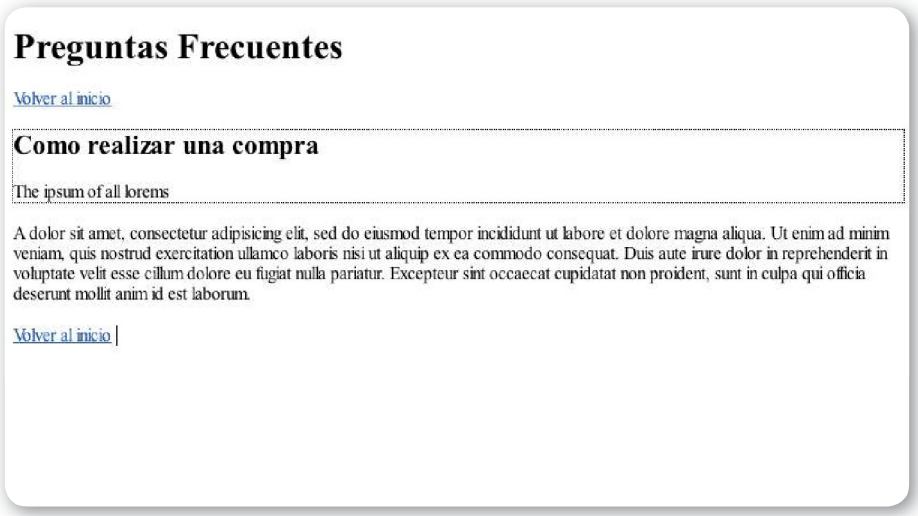


Figura 7. De esta manera veríamos en el navegador el código que acabamos de probar.

Al igual que la etiqueta **<header>**, la etiqueta **<footer> no introduce una nueva sección en el documento**.

En algunos sitios web, podemos encontrar lo que se llama comúnmente **fatfooter**. En estos casos, el pie de página puede contener todo tipo de contenidos, desde el mapa del sitio hasta la inclusión de plugins de redes sociales, productos destacados,

HEADER, FOOTER Y NAV

La etiqueta **<nav>** suele utilizarse dentro de la etiqueta **<header>** en algunos proyectos que requieren que la barra de navegación se encuentre presente en todas las páginas de un sitio web (como por ejemplo la Web de una empresa o incluso un simple blog). El **<footer>** también suele repetirse, pero nunca debe ir anidado dentro de un **<header>**.

formularios de contacto, etcétera. La etiqueta **<footer>** puede ser usada perfectamente para estos casos, ya que en su interior podemos anidar todo tipo de etiquetas.

Figura 8. En este sitio web podemos ver un ejemplo visual de **fatfooter**.

Veamos un ejemplo de código para un **fatfooter**:

```
<footer>
<nav>
<section>
<hl> Destacados del mes</hl>
<p><imgsrc="img/destacadosmes.jpeg" alt="destacado 1">LoremIpsum es simplemente el texto de relleno de las imprentas y archivos de texto.
<a href="articulos/destacadol.html">Parte 1</a> · <a href="articles/somersaults/2">Parte 2</a></p>
<p><imgsrc="img/destacadosmes.jpeg" alt="destacado 2">LoremIpsum es simplemente el texto de relleno de las imprentas y archivos de texto. LoremIpsum ha sido el texto de relleno. <a href="articles/kindplus/1">Leer más...</a></p>
<p><imgsrc="img/destacadosmes.jpeg" alt="destacado 3">The chips are down, now all that's left is a potato. What can you do with it? <a href="articles/crisps/1">Leer más...</a></p>
</section>
<ul>
<li><a href="nosotros.html">Sobre nosotros</a>
```

```
<li><a href="http://facebook.com/webl">Seguinos en facebook!</a>
<li><a href="sitemap.html">Mapa del sitio</a>
</ul>
</nav>
<p><small>Copyright © 2013
<a href="terminos.html">Términos de uso</a></small></p>
</footer>
</body>
```

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

La famosa etiqueta <article>

La etiqueta **<article>** es una de las nuevas y más importantes etiquetas introducidas por HTML5. Se utiliza para **crear un sección de contenido independiente**, es decir, un contenido que posee su propia autonomía y que puede ser reutilizado en otro contexto o mediante sindicación.

La etiqueta **<article>** puede contener un artículo de texto de una revista, un post de un foro, un artículo de un diario, una entrada de un blog, un widget o cualquier otro contenido independiente.

Veamos un ejemplo:

```
<article>
<hl>Diseño Adaptable</hl>
<p>El diseño adaptable es un técnica de .....</p>
</article>
```

La etiqueta **<article>** generalmente contiene etiquetas de títulos (**<h1>**/**<h6>**) y puede contener un **<header>** y un **<footer>** propios.

```
<article>
      <header>
```

```
        <hl>Diseño Adaptable</hl>
                <p>El diseño adaptable es un técnica<p>
        </header>
                <p>Componentes del diseño adaptable</p>
                <p> .... </p>
        <p> .... </p>
        <footer>
                <p>Este artículo fue escrito por Nic</p>
        </footer>
</article>
```

La etiqueta **<article>** puede estar anidada dentro de otra etiqueta **<article>**. En este caso, su contenido debe estar relacionado con el contenido de la etiqueta padre. Este es un caso que se utiliza mucho en las entradas de los blogs.

Veamos un ejemplo de uso:

```
<article>
        <header>
                <hl>Diseño Adaptable</hl>
                <p>El diseño adaptable es un técnica<p>
           </header>
                <p>Componentes del diseño adaptable</p>
                <p> .... </p>
                <p> .... </p>
           <article>
                <p> .... </p>
                <p> .... </p>
           </article>

           <article>
                <p> .... </p>
                <p> .... </p>
           </article>

           <article>
```

```
                <p> .... </p>
                <p> .... </p>
            </article>
</article>
```

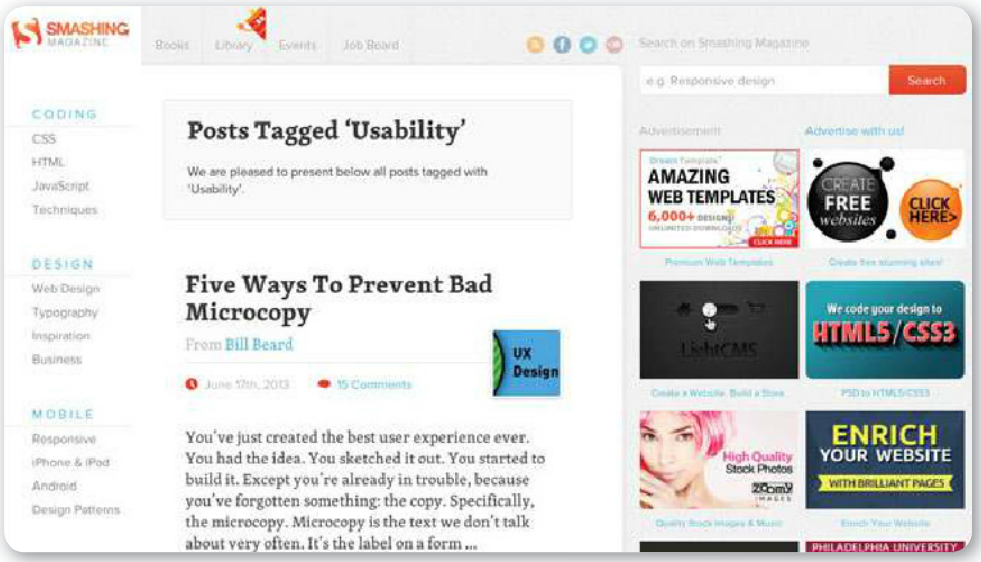


Figura 9. En este sitio podemos ver un caso de uso típico de la etiqueta **<article>**.

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

Creando secciones con <section>

La etiqueta **<section>** se utiliza para delimitar una sección de contenido propio de nuestro documento HTML. Esta etiqueta **debe contener un único tema**, generalmente posee un **<header>** y **siempre debe tener un título** (**<h1>**/**<h6>**). Veamos un ejemplo:

```
<section>
<h1>Diseño Adaptable</h1>
<p>El diseño adaptable es un técnica...</p>
</section>
```

Podemos usar la etiqueta **<section>** para secciones estáticas como el "Quiénes somos", un capítulo, una noticia o la sección de contacto, entre otras. Las páginas principales (home) de un sitio web son ejemplos clarísimos para el uso de la etiqueta **<section>**.

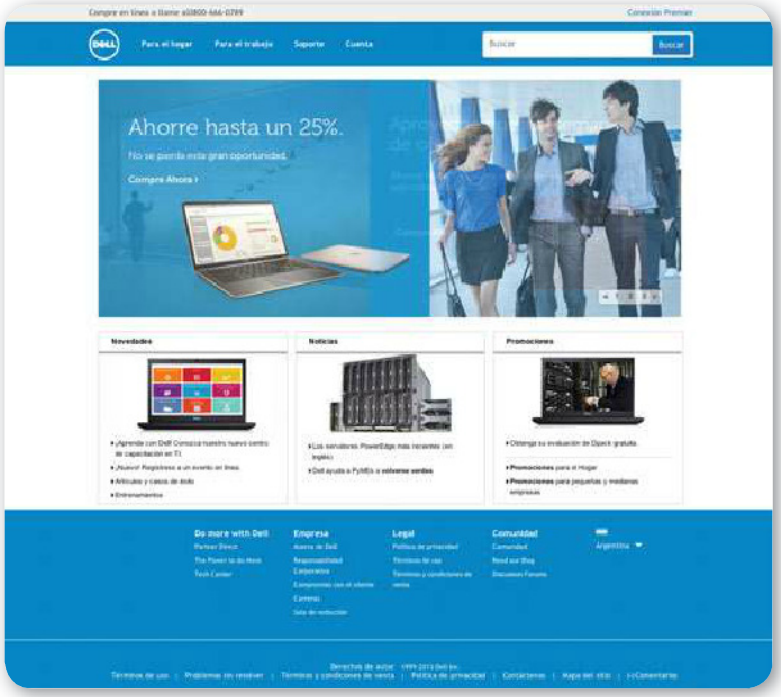


Figura 10. En la home del sitio oficial de Dell podemos ver cómo los contenidos pueden ser subdivididos en secciones.

Es importante resaltar que la etiqueta **<section>** no debe ser usada como un contenedor genérico con propósitos de diseño o de maquetación. Veamos la figura de la siguiente página para entender esta cuestión:

CITAS DE OTROS SITIOS WEB

Entre las nuevas etiquetas de HTML5 se encuentra la etiqueta **<blockquote>**, que se utiliza para destacar contenido que fue extraído de otro sitio web. Si bien no la utilizaremos en todos nuestros proyectos, es importante tenerla en cuenta a la hora de setear la hoja de estilos, si estamos diseñando un sitio web informativo o un blog.

Figura 11. En la maquetación de este proyecto debemos usar la etiqueta **<section>** solo para los bloques que poseen título.

En la figura podemos ver claramente dónde debemos usar la etiqueta **<section>** y en qué casos no debe ser usada. Los bloques que poseen su propio título deben ser maquetados usando etiquetas **<section>**, pero, según la referencia de esta imagen, los tres bloques con títulos están anidados dentro de un bloque general que los contiene visualmente.

Ese contenedor debe ser agregado a los efectos del diseño, y esa es la razón por la cual nunca debemos usar una etiqueta **<section>** para esto, pues la etiqueta **<section>** describe una sección de contenido y, por ende, debe agrupar contenidos propios en lugar de incluir otros contenidos. Veamos este ejemplo en el código HTML:

```
<section id=»main»>

        <section id="cont1"></section>
        <section id="cont2"></section>
<section id="cont3"></section>

</section>
```

En este ejemplo, la etiqueta **<section>** está siendo usada de forma incorrecta. Entonces, ¿cómo deberíamos maquetar este proyecto para que sea semánticamente correcto?

Cuando necesitemos incluir un contenedor que nos sirva para el diseño o la diagramación, deberíamos hacer uso de la etiqueta ** .** Entonces, el código debería quedar de esta manera:

```
<div id="main">

        <section id="cont1"></section>
        <section id="cont2"></section>
<section id="cont3"></section>

</div>
```

Más adelante en este capítulo, veremos que, en este caso, podríamos encontrar una mejor solución, utilizando la etiqueta **<main>**. Sin embargo, el uso de esta etiqueta por ahora está un poco limitado por la compatibilidad de los navegadores.

```
<main>

        <section id="cont1"></section>
        <section id="cont2"></section>
<section id="cont3"></section>

</main>
```

La etiqueta **<section>** puede estar anidada dentro de una etiqueta **<article>**; en este caso, estamos dividiendo el artículo en dos secciones. Veamos un ejemplo de esto último:

```
<header>
<h1>Mariposas</h1>
<p>Pequeños y coloridos insectos voladores</p>
</header>
<p>Existen muchas clases de mariposas en el mundo</p>
<section>

<article>
```

```
<h1>Mariposa Monarca</h1>
<p>LoremIpsum es simplemente el texto de relleno</p>
</section>
<section>
<h1>Mariposa china</h1>
<p>LoremIpsum es simplemente el texto de relleno </p>
</section>
</article>
```

Es importante resaltar que, cuando anidamos una etiqueta **<section>** dentro de una **<article>**, quedarán subordinadas a su etiqueta padre, pues estamos creando una sección del artículo.

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1 y Safari, desde la 5.

<section> vs. <article>

Es muy común que los desarrolladores se confundan a la hora de implementar las etiquetas **<section>** y **<article>**. Es importante entender que **la diferencia entre ellas no es jerárquica:** ninguna debe estar obligatoriamente anidada dentro de la otra. Cada una de ella **posee una semántica diferente y claramente marcada**, que debemos respetar a la hora de realizar nuestra maquetación.

Por esta razón, resulta muy importante tener en claro cuáles son los contenidos que se van a incluir en el documento antes de comenzar el proceso de maquetación.

Cuando maquetábamos los sitios usando ** ** no había gran diferencia, ya que para todos los bloques de contenido usábamos la misma etiqueta. Ahora contamos con varias alternativas, lo cual implica que debamos planificar de antemano el contenido del proyecto.

Uso de los títulos en las secciones

Un punto muy importante a tener en cuenta es el uso de las etiquetas **<h1>/<h6>** dentro de las secciones del documento.
Si volvemos a ver el ejemplo anterior:

```
<article>
<header>
<h1>Mariposas</h1>
<p>Pequeños y coloridos insectos voladores</p>
</header>
<p>Existen muchas clases de mariposas en el mundo</p>
<section>
<h1>Mariposa Monarca</h1>
<p>LoremIpsum es simplemente el texto de relleno de las imprentas</p>
</section>
<section>
<h1>Mariposa china</h1>
<p>LoremIpsum es simplemente el texto de relleno de las imprentas</p>
</section>
</article>
```

Podemos notar aquí cómo se están usando las etiquetas **<h1>.** Cada **<section>** puede usar perfectamente una etiqueta **<h1>** sin tener en cuenta que esa etiqueta de título fue usada en el nivel anterior.

En el estándar de HTML5, es perfectamente válido que cada sección tenga su propia jerarquización de títulos comenzando desde el primer nivel (**h1**).

La etiqueta <aside>

La etiqueta **<aside>** representa una sección dentro de nuestro documento, cuyo contenido está relacionado con el resto del contenido del documento, pero que, al mismo tiempo, es independiente y puede ser omitido sin que esto altere la lectura del resto. Es decir que **se trata de contenido suplementario** que completa el contenido de la página, pero que, si lo eliminamos, el resto del documento no debe perder sentido alguno.

ASIDE REPRESENTA UNA SECCIÓN CON CONTENIDO INDEPENDIENTE

Frecuentemente podemos encontrar dentro de la etiqueta **<aside>** contenidos como banners publicitarios, barras laterales,

navegación suplementaria como en los blogs, aclaración sobre una cita, etcétera.

El significado del **<aside>** va depender de la etiqueta en donde se encuentre anidado. Si se encuentra dentro de un **<article>**, el contenido del **<aside>** debe estar relacionado al contenido del **<article>**, mientras que cuando se encuentre dentro del **<body>**, entonces su contenido deberá estar relacionado al documento.

Veamos un ejemplo gráfico para entender mejor el uso de la etiqueta **<aside>**:

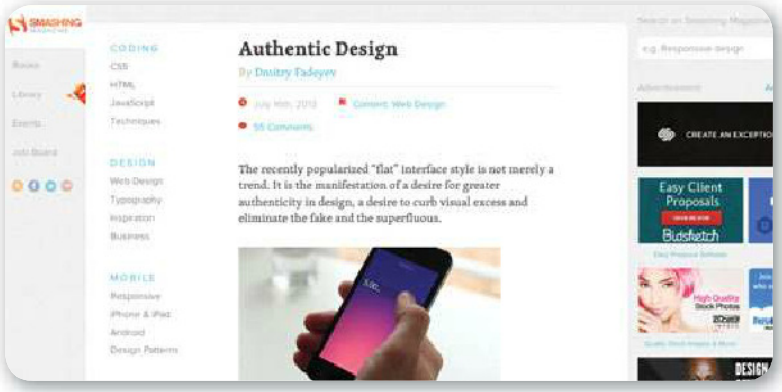


Figura 12. En la página de inicio de Smashing Magazine, podemos ver un ejemplo claro del uso de la etiqueta **<aside>**.

Ahora veamos un ejemplo de código:

```
<aside>
<h1>Diseño Adaptable</h1>
<p>LoremIpsum es simplemente el texto de relleno de las imprentas y archivos
      de texto. LoremIpsum ha sido el texto de relleno estándar de las industrias
      desde el año 1500 </p>
</aside>
```

En el siguiente ejemplo, podemos ver cómo se utiliza la etiqueta **<aside>** para remarcar una cita del texto principal del articulo:

```
<body>
<article>
```

```
        <p>Respecto de este tema Gandhi siempre fue muy claro en su pensamiento
<q>Siento que el progreso espiritual nos demanda que dejemos de matar y comer
        a nuestros hermanos, criaturas de Dios, y solo para satisfacer nuestros
        pervertidos y sensuales apetitos. La supremacía del hombre sobre el animal
        debería demostrarse no solo avergonzándonos de la bárbara costumbre de
        matarlos y devorarlos sino cuidándolos, protegiéndolos y amándolos. No
        comer carne constituye sin la menor duda una gran ayuda para la evolución
        y paz de nuestro espíritu.</q></p>

        <aside>
                <q> La supremacía del hombre sobre el animal debería
                demostrarse no solo avergonzándonos de la bárbara
                costumbre de matarlos y devorarlos, sino cuidándolos </q>
        </aside>
<p>Continua el texto de la nota</p>

</article>
</body>
```

Un dato importante a tener en cuenta es que nunca debemos usar la etiqueta **<aside>** para contener los textos que se colocan entre paréntesis a modo de aclaración. La razón es que este contenido forma parte del flujo del documento HTML y, al colocarlo en una etiqueta **<aside>**, este flujo se rompería. El contenido de un **<aside>** **nunca debe ser el contenido principal** de un documento. De hecho, es un práctica bastante común, cuando hacemos un sitio "responsive", que ocultemos las etiquetas **<aside>** para el layout de las pantallas más pequeñas.

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

USAR O NO USAR CONTENIDO SEMÁNTICO

En este capitulo se enumeran varias etiquetas nuevas de HTML5. Pero solo hay que usarlas si sabemos para qué sirven; si no, estaremos otorgando una semántica errónea a los componentes de nuestro sitio. Si desconocemos su uso o necesitamos bloques para maquetar, es recomendable implementar los ** **.

Adiós a la etiqueta <hgroup>

Una de las nuevas etiquetas semánticas introducidas por HTML5 es la etiqueta **<hgroup>**, que se utiliza como elemento agrupador para títulos **<h1>**/**<h6>**, como podemos ver en el ejemplo siguiente:

```
<hgroup>
<h1>Curso de HTML5</h1>
<h2>Introducción al nuevo estándar de la Web</h2>
</hgroup>
```

No nos detendremos mucho explicando esta etiqueta, ya que **fue eliminada de la versión HTML5.1** en diciembre del año 2012. En ese mismo momento, se anunció que algunas etiquetas –entre las cuales se encontraba **<hgroup>**– corrían el riesgo de ser eliminadas de la especificación oficial. Efectivamente, en el primer borrador presentado de la versión HTML5.1, la etiqueta **<hgroup>** ya no existe.

Seguramente se estarán preguntando qué ocurre con los sitios que venimos desarrollando en HTML5 desde hace algún tiempo, donde usamos la etiqueta **<hgroup>**. Simple: esos sitios tienen, desde entonces, errores de validación que antes no tenían. El validador de W3C nos dirá que estamos usando un código obsoleto. Son los riesgos de trabajar con un lenguaje que aún no cuenta con una versión estándar.

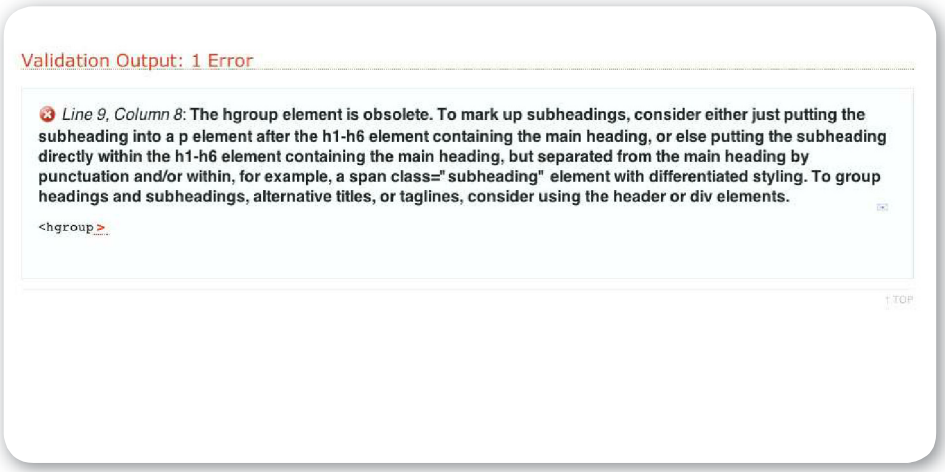


Figura 13. En la figura podemos ver claramente un error de validación que presenta un sitio web que utiliza actualmente la etiqueta **<hgroup>**.

Nuevos elementos de agrupamiento en HTML5

Para continuar con nuestro aprendizaje, en las próximas páginas veremos novedades relativas al agrupamiento.

La llegada de <main>

La etiqueta **<main>** es un elemento nuevo de la especificación HTML5.1. Esta etiqueta representa el contenido principal dentro del **<body>** del documento. No representa una sección, ya que no afecta el **outline** o esquema del documento, sino que es una etiqueta cuya función es **agrupar otras etiquetas** que pueden ser o no de secciones. Encierra contenido que es único en el documento, es decir, que no se repite en los distintos archivos de nuestro sitio, como sí podría ser el **<nav>** o bien el **<footer>** general de la página. No debe estar anidada dentro de etiquetas como **<article>**, **<aside>**, **<footer>**, **<header>** o **<nav>**. Veamos un ejemplo:

```
<main>

<h1>Diseño Adaptable</h1>
<p>El diseño web es una técnica de maquetación</p>

<article>
<h2>Componentes de diseño adapatable</h2>
<p>LoremIpsum is simply dummy text of the printing and typesetting industry.</p>
<p>LoremIpsum is simply dummy text of the printing and typesetting industry.</p>
</article>

<article>
<p>LoremIpsum is simply dummy text of the printing and typesetting industry.</p>
<p>LoremIpsum is simply dummy text of the printing and typesetting industry.</p>
</article>

</main>
```

Se recomienda usar ARIA en la etiqueta **<main>**, como lo hacíamos antes con ** **, dado que la mayor parte de los navegadores aún no reconoce el elemento **<main>**. Veamos un ejemplo:

```
<main role="main">
<h1>Diseño Adaptable</h1>
</main>
```

El atributo **role="main"** les indica a los navegadores que **este elemento es el que se establece como principal dentro del documento**. Puede ser usado una única vez en el archivo, del mismo modo que la etiqueta **<main>.**

Hasta el momento, esta etiqueta solo es soportada por los navegadores Chrome, a partir de la versión 25, y Opera, desde la edición 15. De todas maneras, es importante revisar la tabla de compatibilidades para verificar si el resto de los navegadores añadieron soporte para **<main>.**

Crear figuras mediante <figure>

La etiqueta **<figure>** se utiliza para representar figuras dentro de un contenido de texto. **Su contenido debe estar relacionado al del elemento padre, pero al mismo tiempo debe ser independiente** de este, de manera tal que si quitáramos la etiqueta **<figure>**, el contenido del elemento padre no debería perder sentido.

La etiqueta **<figure>** puede ser usada para contener imágenes, ilustraciones, gráficos, tablas, videos, un bloque de código, etcétera.

Veamos un ejemplo básico de su uso:

WAI-ARIA

Así se denomina al conjunto de atributos de HTML que **permiten agregar información que mejora la accesibilidad** de los documentos HTML. Estos atributos, invisibles a simple vista, brindan información adicional para los robots y motores de búsqueda, mejorando así la experiencia de los usuarios y los resultados de las búsquedas.

```
<figure>

<imgsrc="grafico-barras.png" alt="Grafico de barras" width="500"
        height="250">

</figure>
```

La etiqueta **<figure>** también puede ser utilizada para contener ejemplos de código, tal como podemos ver en el ejemplo que aparece a continuación:

```
<figure id="ejemplo1">

<pre><code>h1 {
        font-family:Verdana;
        font-size:24px;
        font-weight:bold;
        color:#333;
        }</code></pre>

</figure>
```

Podemos observar un caso de uso típico, y además muy fácil de entender, en los sitios de noticias, donde se suelen incluir imágenes ilustrativas dentro de los contenidos de texto, tal como nos muestra la figura de la página siguiente.

MÁS DE UNA IMAGEN

Otra de las posibilidades que nos ofrece la etiqueta **<figure>** consiste en incluir varias imágenes dentro de un mismo bloque y asignarle un único **<figurecaption>**. De esta forma, se puede aplicar un mismo epígrafe a varias fotos, dibujos o diagramas relacionadas con un mismo tema. Además, al utilizar esta etiqueta estaremos ahorrando código en la creación de nuestro sitio web.


EL CIRCUITO ATP

Andy Murray descansa con su novia en Bahamas, tras ganar Wimbledon

Luego de ser el primer británico en 77 años en ganar el certamen, el campeón se relaja.

A los 26 años, Andy Murray dio uno de los golpes más grandes al ser el primer británico en ganar Wimbledon, luego de 77 años. La gesta la logró hace 9 días en Londres y por estas horas el campeón disfruta y descansa en Bahamas junto a su novia, Kim Sears. Sin tenis, el hombre se divierte con la paleta.

La pareja descansa en el caribe

A los 26 años, Andy Murray dio uno de los golpes más grandes al ser el primer británico en ganar Wimbledon, luego de 77 años. La gesta la logró hace 9 días en Londres y por estas horas el campeón disfruta y descansa en Bahamas junto a su novia, Kim Sears. Sin tenis, el hombre se divierte con la paleta.

Figura 14. Podemos usar la etiqueta **<figure>** para contener la imagen ilustrativa y el epígrafe de esta nota.

La figura puede tener o no un título. En caso de que vayamos a incluir un título para la figura, debemos hacerlo mediante la etiqueta **<figurecaption>** que veremos a continuación en este capítulo.

Veamos cómo quedaría el código HTML para maquetar el ejemplo de la figura anterior:

```
<figure>

    <figurecaption> La pareja descansa en el caribe </figurecaption>
<imgsrc="foto-pareja.jpg" width="500" height="250">

</figure>
```

Si bien es bastante común usar la etiqueta **<figure>** para contener imágenes, es importante aclarar que no toda imagen debe estar contenida dentro de una etiqueta **<figure>**, sino solamente cuando queremos significar que alguna imagen forma parte de un figura.

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5. La etiqueta **<figurecaption>** representa el título de una figura y solo puede ser usada dentro de una etiqueta **<figure>**, aunque su uso no es estrictamente obligatorio.

Observemos nuevamente el ejemplo que vimos antes, pero agregando un título para la figura:

```
<figure id="ejemplo1">
<figcaption> Declaración de reglas de estilo </figcaption>
<pre><code>h1 {
        font-family:Verdana;
        font-size:24px;
        font-weight:bold;
        color:#333;
        }</code></pre>
</figure>
```

No es obligatorio que la etiqueta **<figurecaption>** sea el primer hijo dentro de la etiqueta **<figure>**, sino que también podemos usar esta etiqueta a modo de epígrafe para una imagen o un gráfico.

Veamos un ejemplo utilizando la etiqueta **<figurecaption>** a modo de epígrafe:

```
<figure>

<imgsrc="foto-pareja.jpg" width="500" height="250">
        <figurecaption> La pareja descansa en el caribe </figurecaption>

</figure>
```

ACCESIBILIDAD EN LA ETIQUETA FIGURE

En HTML5, el atributo **alt** para las imágenes no es obligatorio, pero sí es muy importante que sea usado, a menos que se trate de una etiqueta **<figure>** en la cual la etiqueta **<figcaption>** cumpla una función similar en cuanto a la accesibilidad. En este caso, no se recomienda usar el atributo **alt** para la imagen, ya que los lectores de pantalla leerían dos veces el mismo texto.

En maquetaciones más complejas, la etiqueta **<figure>** también puede anidar a otras etiquetas **<figure>**. Veamos un ejemplo:

```
<figure>
<figcaption> Vanguardias artísticas </figcaption>
<figure>
<figcaption> Impresionismo </figcaption>
<imgsrc="impresionismo.jpeg" alt="Impresionismo">
</figure>

<figure>
<figcaption> Expresionismo </figcaption>
<imgsrc="expresionismo.jpeg" alt="Expresionismo">
</figure>

<figure>
<figcaption> Fauvismo </figcaption>
<imgsrc="fauvismo.jpeg" alt="Fauvismo">
</figure>
</figure>
```

Nuevos elementos semánticos de texto

HTML5 ofrece nuevas etiquetas para añadir significado a los bloques de texto, permitiendo una mejor lectura de los buscadores y de los agentes de usuario. A continuación, repasamos los más importantes.

La etiqueta <time>

Sirve para contener **información sobre fechas y horarios**, usando un formato estandarizado que pueda ser comprensible por los robots.

```
<time> 2013-10-06 </time>
```

El formato en el cual escribimos la fecha **debe responder a un formato de fecha estabilizado**. De lo contario, el robot no podrá comprenderlo, a menos que utilicemos el atributo **datetime** (nuevo en HTML5). En este caso, podemos usar un formato de escritura para los robots y otro distinto para los seres humanos, por ejemplo:

```
<time datetime="2013-10-06"> 6 de octubre</time>
```

De esta manera, incluimos un texto más atractivo para los seres humanos y un formato estandarizado para los robots. Es importante mencionar que, si la etiqueta **<time>** no posee el atributo **datetime**, no debería entonces tener etiquetas anidadas.

A continación, enumeramos los formatos soportados por la etiqueta **<time>**:

Formato válido de mes:

```
<time>2013-10</time>
```

Formato válido de fecha:

```
<time>2013-10-06</time>
```

Formato válido sin establecer el año:

```
<time>10-06</time>
```

Formatos válidos de horario:

```
<time>13:45:50</time>

<time>13:45:50.679</time>

<time>13:45</time>
```

Formatos válidos de fecha y horario local:

```
<time>2013-10-06T13:45</time>

<time>2013-10-06T13:45:50</time>

<time>2013-10-06T13:45:50.678</time>

<time>2013-10-06 13:45:50</time>
```

Formatos válidos para determinar la zona horaria:

```
<time>Z</time>

<time>+0000</time>

<time>+00:00</time>

<time>-0300</time>

<time>-03:00</time>
```

Formatos válidos aplicando la zona horaria:

```
<time>2013-10-06T13:45Z</time>
<time>2013-10-06T13:45:39Z</time>
<time>2013-10-06T13:45:39.929Z</time>

<time>2013-10-06T13:45+0000</time>
<time>2013-10-06T13:45:39+0000</time>

<time>2013-10-06T13:45:39.929+0000</time>

<time>2013-10-06T13:45+00:00</time>
<time>2013-10-06T13:45:39+00:00</time>
<time>2013-10-06T13:45:39.929+00:00</time>
```

```
<time>2013-10-06T06:54-0800</time>
<time>2013-10-06T06:54:39-0800</time>
<time>2013-10-06T06:54:39.929-0800</time>

<time>2013-10-06T06:54-08:00</time>
<time>2013-10-06T06:54:39-08:00</time>
<time>2013-10-06T06:54:39.929-08:00</time>
```

Formato válido para la semana:

```
<time>2011-W46</time>
```

Formato válido para indicar un intervalo de tiempo:

```
<time>PT4H18M3S</time>

<time>4h 18m 3s</time>
```

Veamos un caso de uso concreto en el cual indicamos los datos de un evento:

```
<article>
<a href="http://www.waf.com/"> http://www.waf.com/</a>
<span>Web Argentine Festival</span>:
<time datetime="2013-10-05T15:00:00"> 5 de octubre, 15h</time>
- en el Paseo La Plaza, Buenos Aires</span>
</article>
```

La etiqueta **<time>** también puede se usada para "encodear" fechas usando microformatos.

Veamos algunos ejemplos más para entender el uso específico de esta etiqueta:

```
<p>Nuestro casamiento fue <time datetime="2013-10-06">un domingo </
    time>.</p>

<p> Terminamos el trabajo <time datetime="2013-09-24T07:00:00">a las
    7am del otro día</time>.</p>
```

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

La nueva etiqueta <data>

La etiqueta **<data>** pertenece al estándar de HTML5.1. Este elemento está muy relacionado con la etiqueta **<time>** que vimos recién, pero su uso es mucho mas genérico. Mientras **<time>** tiene un uso exclusivo para fechas y horarios, la etiqueta **<data>** permite incluir todo tipo de contenidos en un formato estandarizado para los robots y una representación diferente para los seres humanos. Esta etiqueta utiliza el atributo **value**, su uso es obligatorio. Veamos un ejemplo:

```
<data value="8">0cho</data>
<data value="93">80+13 hexágonos (93) total</data>
```

El atributo **value** provee a los robots un formato estandarizado para comprender el contenido de la etiqueta usando microformatos o microdata.

¿QUÉ ES LA MICRODATA?

En la especificación de HTML5, aparece el concepto de **microdata** o **microdatos**. La microdata nos ayuda a darle información semántica a una estructura de datos. Estos datos extra, contenidos en las etiquetas de nuestros documentos, brindan a los motores de búsqueda información adicional para realizar búsquedas mas precisas. Los microdatos utilizan, principalmente, los atributos **itemscope** e **itemprop.**

El contenido del atributo **value** puede ser utilizado también por los scripts que contenga el documento, almacenando la información para ser usada en otro momento. En estos casos, el formato dependerá solamente de las necesidades del script. Esto se verá con mayor detalle en el **Capítulo 11**.

La etiqueta **<data>** puede ser utilizada para cualquier tipo de contenido. Sin embargo, no es recomendable usarla cuando nos referimos a una fecha y horario, ya que, en ese caso, sería más adecuado usar la etiqueta **<time>.**

Esta etiqueta tiene por ahora un soporte muy bajo: solo es aceptada por Chrome, a partir de la versión 25, y Opera, desde la 15.

La etiqueta <mark>

La etiqueta **<mark>** se utiliza para hacer un llamado de atención al usuario. Veamos un ejemplo de uso de esta etiqueta:

```
<p>En este capítulo vamos a hablar sobre <mark>CSS3</mark> y sus nuevas
    capacidades.</p>
```

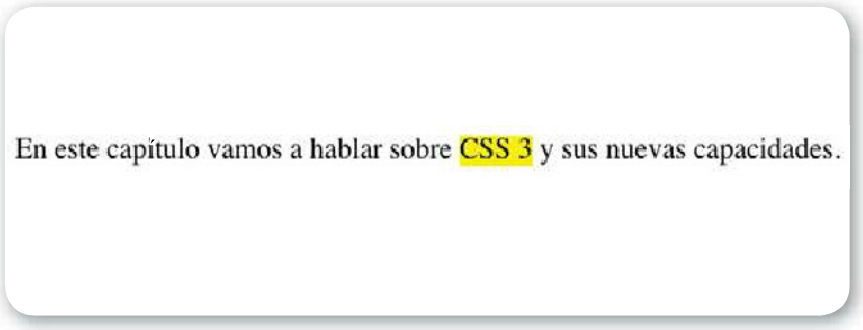


Figura 15. Podemos notar que el texto contenido en la etiqueta **<mark>** se ve resaltado.

Esta etiqueta nunca debe ser usada con el criterio con el que usamos una etiqueta **<b>**, **** o ****. Su función es realizar un **llamado de atención** en algo que no había sido resaltado originalmente por el autor del documento. Está relacionado, más bien, con las acciones del usuario.

Por ejemplo, podríamos implementar esta etiqueta para contener las palabras que coinciden con la búsqueda de un usuario en el buscador

interno de la página. También serviría para indicar cambios surgidos en el contenido.

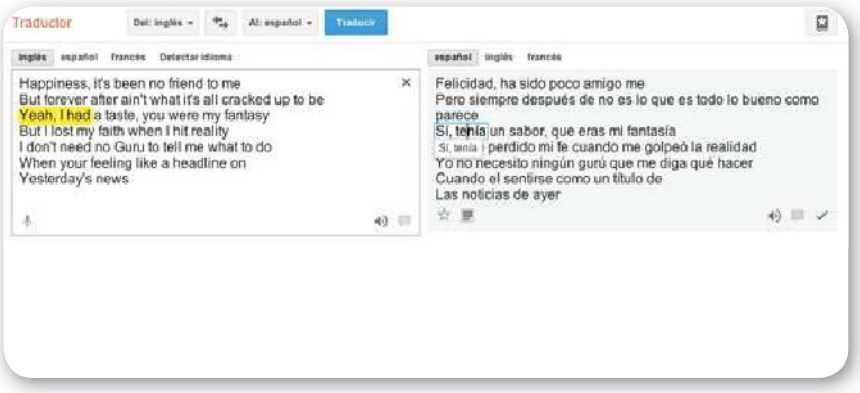


Figura 16. En la caja de traducción de Google Translate observamos que, al señalar una palabra, esta se marca en la caja opuesta.

Las etiquetas <ruby>, <rt> y <rp>

La etiqueta **<ruby>** se utiliza para **insertar anotaciones Ruby dentro de nuestro documento**. Generalmente, utilizaremos esta etiqueta en forma conjunta con las etiquetas **<rt>** y **<rp>**. Las etiquetas **<rt>** y **<rp>** siempre deben estar anidadas dentro de la etiqueta **<ruby>**: no existe otro lugar posible donde puedan ser usadas.

Los caracteres Ruby son anotaciones pequeñas que se ubican arriba o a la derecha, entre paréntesis, en lenguajes ideográficos, como el chino o el japonés. Esas anotaciones son necesarias para poder incluir su pronunciación. En ese sentido, podríamos compararlo con las anotaciones fonéticas que encontramos en un diccionario de inglés, que posiblemente sea un ejemplo más cercano a nosotros.

MICRODATA EN HTML5

Entre las nuevas capacidades de HTML5, encontramos la microdata, basada en una microsintaxis semántica que utiliza los atributos de los elementos de HTML para dar significado a los contenidos que serán usados principalmente por los robots y motores de búsqueda. De hecho, sitios como Google, Facebook y Bing, entre otros, hacen uso a diario de estos contenidos.

Figura 17. En la imagen podemos ver las anotaciones **Ruby** arriba del ideograma.

Veamos un ejemplo de código para comprender mejor la implementación de la etiqueta Ruby:

```
<ruby> B <rt> anotación </ruby>
```

En el ejemplo de arriba, podemos ver que la etiqueta **<ruby>** contiene el carácter ideográfico, representado en este ejemplo por la letra B. Y luego, anidado dentro de él, la etiqueta **<rt>**. La función de esta es contener el texto de la anotación propiamente dicha. Este contenido debe aparecer arriba o a la derecha del ideograma, según como lo tenga implementado el navegador. Veamos otros ejemplos:

```
<ruby>漢<rt>かん</ruby>
```

La etiqueta **<ruby>** puede contener uno o más ideogramas con sus correspondientes anotaciones cada uno. Además, la etiqueta **<rt>** también puede contener alguna traducción, como podemos ver en el siguiente ejemplo:

```
<ruby lang="ja">編集者<rtlang="es">editor</ruby>
```

Si bien su uso principal es para anotaciones en idiomas asiáticos, la etiqueta **<ruby>** también puede contener otro tipo de anotaciones, como vemos en el siguiente ejemplo:

```
<ruby>
        ♥ <rt>Corazón<rtlang=fr>Cœur
        ♣<rt>Trébol<rt=fr>Trèfle
        ✶<rt>Estrella<rtlang=fr>Étoile
</ruby>
```

Dentro de la etiqueta **<ruby>**, también podemos utilizar la etiqueta **<rp>**, que se utiliza para los contenidos que deseamos que sean ignorados dentro de la anotación (como, por ejemplo, los paréntesis).

Algunos navegadores no poseen soporte para que la etiqueta **<ruby>** se renderice con la anotación arriba del ideograma, sino que se verá a la derecha. En estos casos, debemos usar la etiqueta **<rp>** para incluir paréntesis, sin alterar la semántica de las etiquetas **<ruby>** y **<rt>.**

> LAS ETIQUETAS RUBY SIRVEN PARA HACER ANOTACIONES SOBRE PALABRAS EN OTROS IDIOMAS

Veamos un ejemplo de uso con la etiqueta **<rp>**:

```
<ruby>漢<rp> (</rp><rt>かん</rt><rp>) </rp>字<rp>
      (</rp><rt>じ</rt><rp>) </rp></ruby>
```

Los navegadores que tengan soporte completo para la etiqueta **<ruby>** no mostrarán los paréntesis, mientras que aquellos que no la soporten por completo mostrarán las anotaciones a la derecha del ideograma entre paréntesis. Esta etiqueta también puede ser utilizada para contener otros caracteres que no formen parte del ideograma o la anotación Ruby. Veamos un ejemplo de uso:

```
<ruby>
        ♥ <rp>: </rp><rt>Heart</rt><rp>, </rp><rtlang=fr>Cœur</rt><rp>.</rp>
        ♣<rp>: </rp><rt>Shamrock</rt><rp>, </rp><rtlang=fr>Trèfle</rt><rp>.
                </rp>
        ✶<rp>: </rp><rt>Star</rt><rp>, </rp><rtlang=fr>Étoile</rt><rp>.
                </rp></ruby>
```

Este ejemplo debe ser renderizado de la siguiente forma:

```
♥: Heart, Cœur. ☘: Shamrock, Trèfle. ✶: Star, Étoile.
```

El soporte por parte de los navegadores para las anotaciones Ruby es algo pobre, ya que la mayoría las soporta solo parcialmente. Chrome 5, Internet Explorer 5.5, Opera 15 y Safari 5, además de las versiones posteriores de estos navegadores, son compatibles con algunas características de la etiqueta Ruby. Firefox aún no las implementa.

La etiqueta <bdi>

La etiqueta **<bdi>** se utiliza para **contener porciones de texto con el propósito de que este tenga un sentido de lectura distinto** al sentido de lectura del elemento padre. Esto es útil cuando queremos insertar un texto en un idioma cuyo sentido de lectura es distinto al general del documento. Por ejemplo: si colocamos un nombre de usuario en árabe dentro del elemento de una lista en inglés.

La etiqueta **<bdi>** es un elemento de línea y, por ello, puede ser perfectamente anidado dentro de cualquier otra etiqueta, ya sea de bloque o de línea. Esta etiqueta tiene por defecto el atributo **dir** configurado en **auto**, y nunca va a heredar este atributo de su elemento padre. Esto debe ser establecido en la etiqueta **<bdi>** si se quire cambiar.

Veamos un ejemplo de uso para esta etiqueta:

```
        <li>User<bdi>إيان</bdi>: 3 comentarios.
</ul>
<ul>
        <li>User<bdi>eugecas</bdi>: 12 comentarios.
        <li>User<bdi>diego31</bdi>: 5 comentarios.
```

Los navegadores que soportan esta etiqueta solo son Chrome 15 (en adelante) y Firefox desde la versión 8.

La etiqueta <wbr>

La etiqueta **<wbr>** genera un salto de línea de la misma forma en que lo hace una etiqueta **
, pero solo **si el navegador no tiene espacio suficiente para contener la linea de texto completa.
Es decir que, si existe espacio suficiente, esta etiqueta no generará ningún salto de línea. Es común usar esta etiqueta en cadenas de caracteres largas para poder tener mayor control del renderizado del contenido de texto.

Veamos un ejemplo:

```
<p> Este libro habla sobre <wbr> HTML5 <wbr> CSS3 <wbr> y JavaScript </p>
```

Si el navegador tiene espacio suficiente, la frase se verá completa en una sola línea, mientras que, de no haber espacio suficiente, aparecerán saltos de línea por donde lo indiquen las etiquetas **<wbr>**, y el texto se renderizará del siguiente modo:

```
Este libro habla sobre
HTML5
CSS 3
y JavaScript
```

La etiqueta **<wbr>** no posee cierre, es una etiqueta vacía, al igual que la etiqueta **
.**

Esta etiqueta es soportada por Chrome, a partir de la versión 6; Firefox, desde la 4; Internet Explorer, desde la 9; Opera, desde la 11.1, y Safari, desde la 5.

MOTOR DE RENDERIZADO

Así se denomina al software que toma el contenido de marcado HTML y su información de formato, como CSS y JavaScript, para procesarlo y mostrarlo ya formateado en el navegador. Todos los navegadores incluyen uno: **Trident** (IE), **Gecko** (Firefox), **WebKit** (Safari, Chrome) y **Presto** (Opera, hasta la versión 15). Google y Opera desarrollaron **Blink**, basado en WebKit e incluido desde Chrome 28 y Opera 15.

Los elementos que fueron modificados

Hay un grupo de elementos que existen desde versiones anteriores, pero que sufrieron cambios en la especificación de HTML5 para obtener un nuevo valor semántico. Este apartado está dedicado a cada uno de estos elementos.

La etiqueta <a>

Ya todos conocemos la etiqueta **<a>**, sin la cual no podríamos crear enlaces entre archivos. En HTML5, esta etiqueta sufre algunos cambios: si bien su especificación se mantiene, el modo en que la usamos puede ser diferente, pues **ahora puede anidar etiquetas de bloque**, como párrafos y títulos, entre otras.

El beneficio de esta nueva posibilidad es **crear bloques completos que sean clickeables** y –de este modo– más accesibles, utilizables y con menor cantidad de código. Veamos un ejemplo:

```
<article>
        <hl><a href="detalle.html">Cafetera eléctrica Doney</a></hl>
        <ahref="detalle.html"><imgsrc="caferera.jpg"></a>
        <p><a href="detalle.html">Texto de la cafetera</a></p>
        <ahref="detalle.html">Ver detalles</a>
</article>
```

Usando HTML5, el código podría crearse de este modo:

```
<article>
        <a href="detalle.html">
<hl>Cafetera eléctrica Doney</hl>
        <imgsrc="caferera.jpg">
        <p>Texto de la cafetera</p>
        <span>Ver detalles</span>
</article>
```

De esta forma, el código no solo es más simple, sino que también es más accesible ya que todo el bloque podrá ser clickeado por el usuario.Esto **mejorará la experiencia**, especialmente en dispositivos móviles.

Figura 18. La figura muestra un bloque de contenido típico de un sitio web de venta de productos.

La etiqueta <address>

Si bien mantiene la misma función, la etiqueta **<address>** posee, en HTML5, **una semántica de mayor relevancia**, ya que genera una sección con un nivel de importancia similar a un **<article>**, un **<nav>** o un **<h1>.**

La etiqueta **<address>** se utiliza para contener datos de contacto correspondientes al elemento padre, que puede ser –por ejemplo– un **<article>**. Si la etiqueta **<address>** se encuentra directamente en el **<body>**, entonces los datos que contiene corresponden al documento completo. Veamos un ejemplo de uso:

```
<address class="autor">
        <em><a title="Comentarios de John Doe" href="#">Joe Doe</a></em>
</address>
```

La etiqueta **<address>** se encuentra comúnmente dentro de **<footer>.** No debe ser usada para contener datos que no sean de contacto; y tampoco deben incluirse dentro de ella direcciones postales o de correo electrónico. Es soportada por todos los navegadores actuales.

La etiqueta <cite>

La etiqueta **<cite>** existe desde las ediciones más antiguas de este lenguaje, más precisamente, desde HTML2. Se utilizaba para indicar el nombre de un libro o alguna otra cita. En HTML3.2 y HTML4.01 fue reformulada, recibiendo una descripción muy vaga y genérica: se la definió como contenedor de una referencia a otros recursos o una cita.

En HTML5 la etiqueta **<cite>** volvió a reencontrarse casi con su descripción original que, durante muchos años, había sido olvidada. En este nuevo estándar, utilizamos la etiqueta **<cite>** para **contener el título de una obra**, como el nombre de un libro, un poema, una canción, una película, un programa de televisión, un juego, una pintura, una escultura, etcétera. Es importante resaltar que esta etiqueta solo debe contener el nombre de la obra y no el nombre del autor. Veamos un ejemplo de cómo utilizarla:

```
<p>La saga cinematográfica <cite>El señor de los anillos</cite> fue dirigida
      por Peter Jackson</p>
```

Es bastante común que esta etiqueta sea utilizada de forma incorrecta, debido a la vaga especificación que recibió en la versiones anteriores de HTML. El siguiente sería un mal uso para esta etiqueta, según el estándar HTML5:

```
<p><cite>El señor de los anillos, por J.R.R Tolkien </cite></p>
```

También es común confundir la etiqueta **<cite>** con las etiquetas **<blockquote>** y **<q>**. Es soportada por todos los navegadores actuales.

La etiqueta <hr>

La etiqueta **<hr>** es otro de los elementos cuya especificación ha sido modificada en HTML5. En las versiones anteriores del lenguaje, no tenía ningún valor semántico, pues su función era representar simplemente una línea visual. Y, por lo general, era preferible crear líneas separadoras mediante CSS con el fin de mantener más acotado el código HTML.

En HTML5, los navegadores siguen representando visualmente la etiqueta **<hr>** de la misma manera, pero con un significado semántico diferente. En la especificación de HTML5, representa un separador temático, es decir, un divisor entre un tema y otro. Veamos el siguiente ejemplo de uso:

```
<section>
        <hl>Noticias recientes</hl>
        <p>LoremIpsum es simplemente el texto</p>
        <hr>
        <p>LoremIpsum es simplemente el texto de relleno de las imprentas
                y archivos de texto. </p>
</section>
```

Esta etiqueta es soportada por todos los navegadores actuales.

La etiqueta <i>

La etiqueta **<i>** también es una de las etiquetas más antiguas de HTML. En las versiones anteriores, poseía un rol vinculado a la representación visual del texto en cursiva. En HTML5, se la ha dado una nueva especificación y nuevos valores semánticos, con lo cual su uso –que cada vez era menor– vuelve a cobrar un nuevo impulso.

La etiqueta **<i>** representa, en HTML5, **un contenido de texto en una voz alterna**, como puede ser un término técnico, un contenido en otro idioma, etcétera. Veamos un ejemplo de su uso:

```
<p>Los <i>lepidópteros</i> son insectos voladores</p>

<p>Pegar el texto en un documento HTML y<i lang="fr">voilà</i></p>

<p>El término<i>función</i> será explicado en el próximo capítulo.</p>
```

Es recomendable incluir clases en las etiquetas **<i>** para identificar el sentido de ese contenido, como podemos ver en el siguiente ejemplo:

```
<p>El término<i class="termino-tecnico">función</i> será explicado
    en el próximo capítulo.</p>
```

Es importante recordar que si queremos utilizar tipografía en cursiva fuera de este contexto de uso, lo correcto es hacerlo mediante CSS. Esta etiqueta es soportada por todos los navegadores actuales.

La etiqueta <b>

La etiqueta **<b>** también fue modificada en la especificación de HTML5. En las versiones anteriores de HTML se la utilizaba para incluir texto en negrita; en cambio, ahora se utiliza para resaltar visualmente las palabras claves, sin agregar importancia extra a los contenidos que encierra (como sí lo hace la etiqueta ****). Veamos un ejemplo de uso:

```
<p><b>Atención</b> No alimente a los animales</p>
```

En este caso, la etiqueta **<b>** está resaltando visualmente la palabra "atención", para aquellos usuarios que acceden mediante la vista, pero **no la está reforzando semánticamente para los robots** de los buscadores, como sería en el caso del uso de la etiqueta **.** La etiqueta **<b>** es soportada por todos los navegadores actuales.

La etiqueta

La etiqueta **** fue modificada en HTML5 para darle un mayor sentido semántico: representa un contenido al que se le da un énfasis fuerte. Esto significa que ese contenido deberá ser pronunciado de una forma diferente, remarcándolo cuando sea leído.

```
<p>Los conejos <em>son</em> animales muy tiernos.</p>
```

Podemos ver, en este ejemplo, que lo que se está queriendo resaltar es la palabra "son", para enfatizar el hecho al cual se refiere. Esta etiqueta está disponible en todos los navegadores.

La etiqueta

La etiqueta **** también sufrió una leve modificación en su especificación, sin alterar el valor semántico que ya tenía, sino, más bien, reforzándolo. Se utiliza para **contener información importante** del sitio, como, por ejemplo, las palabras claves que buscarán los motores de búsqueda. Esta etiqueta es soportada por todos los navegadores actuales. Veamos un ejemplo:

```
<p>El curso de <strong>HTML5 y CSS3</strong> requiere conocimientos
    previos</p>
```

La etiqueta <small>

La etiqueta **<small>** se utilizaba antes para **disminuir el tamaño de la tipografía**, al contrario de lo que hace la etiqueta **<big>**, eliminada del estándar de HTML5. A diferencia de la etiqueta **<big>**, **<small>** sobrevivió en HTML5 porque recibió un nuevo valor semántico: en esta versión, se la utiliza para contener el texto que conocemos comúnmente como "letra chica". No se refiere al tamaño de la tipografía, sino al concepto semántico. Esta etiqueta se utiliza habitualmente para contener texto relativo a las informaciones de tipo legal, políticas de uso, copyright, licencias, etcétera. Veamos cómo se utiliza:

```
<p><small>Para conocer más sobre esta promoción, ver <a href="apple.com/
    bases.html">bases y condiciones</a>en nuestro sitio web </small></p>
```

Es importante mencionar que la etiqueta **<small>** no debe ser usada para grandes cantidades de texto, como párrafos de líneas múltiples o listas de contenido. Es recomendable el uso de clases en la etiqueta **<small>** para dar un mejor contexto de su significado. Esta etiqueta es soportada por todos los navegadores actuales.

```
<p><smallclass="bases-y-condiciones">Para conocer más sobre esta promoción,
    ver <a href="apple.com/bases.html">bases y condiciones</a>en nuestro
    sitio web </small></p>
```

Atributos globales de HTML5

En HTML contamos con una gran cantidad de atributos, algunos de los cuales son llamados **atributos globales** porque pueden ser usados en cualquier etiqueta. A continuación, repasaremos los atributos globales que pertenecen al estándar de HTML5. La mayoría de ellos proviene de las versiones anteriores y otros fueron introducidos con la nueva versión.

Accesskey

Se utiliza para especificar la tecla de acceso directo para darle foco a un elemento. Ejemplo de uso:

```
<ahref="http://www.facebook.com" accesskey="f">Facebook</a>
```

Class

Sirve para especificar el o los nombres de clases que existen en el archivo de estilo CSS y que serán usados por el elemento. Ejemplo de uso:

```
<p class="destacado">Contenido de texto del párrafo.</p>
```

Dir

Es usado para especificar la dirección de lectura del texto que tendrá el contenido de la etiqueta que lo contiene. Ejemplo de uso:

```
<p dir="rtl">Contenido de texto del párrafo.</p>
```

Hidden

Oculta el elemento que posee este atributo. Ejemplo de uso:

```
<p hidden="hidden"> Este párrafo está oculto </p>
```

Id

Se utiliza para aplicar un identificador único al elemento. Ejemplo:

```
<p id="importante"> Contenido de texto del párrafo</p>
```

Lang

Se usa para declarar el idioma con el cual está escrito el contenido de la etiqueta. Ejemplo de uso:

```
<p lang="it"> Este texto está en Italiano </p>
```

Style

Se utiliza para aplicar un estilo en línea de CSS al elemento. Ejemplo:

```
<p style="color:red;"> Este texto es de color rojo</p>
```

Tabindex

Establece el orden de tabulación que posee el elemento. Por ejemplo:

```
<p tabindex="5"> Contenido de texto del párrafo. </p>
```

Title

Permite agregar información adicional al elemento que el navegador mostrará cuando se ponga el cursor del mouse sobre él. Ejemplo de uso:

```
<p title="ver ficha del producto"> Contenido de texto del párrafo. </p>
```

Hemos visto los atributos de las versiones anteriores del lenguaje. A continuación, veremos los incorporados en la especificación de HTML5.

Contenteditable

Define si el contenido puede o no ser editable. Ejemplo de uso:

```
<p contenteditable="true"> Contenido de texto del párrafo. </p>
```

Contextmenu

Este atributo permite especificar el menú contextual que posee el elemento **<menu>** cuando el usuario hace clic sobre este. Aún no posee soporte en ningún navegador. Veamos, a continuación, un ejemplo de uso:

```
<p contextmenu="mimenu">Este párrafo tiene un menú contextual
      denominado "mimenu"</p>
<menu id="mimenu">
<commandlabel="Copiar" onclick="copiar()">
<commandlabel="Pegar" onclick="oegar()">
</menu>
```

Draggable

Determina si el elemento puede ser arrastrado o no. Este atributo requiere de JavaScript. Ejemplo de uso:

```
<p draggable="true" ondragstart="drag(event)">Este es un párrafo que se puede
      arrastrar</p>
```

Dropzone

Determina si los datos del elemento arrastrado son copiados, vinculados o movidos cuando se suelta el elemento arrastrado.

Ejemplo de uso:

```
<div dropzone="copy/move/link">Este div puede recibir contenido
      arrastrado</div>
```

Spellcheck

Especifica si el elemento va a ser sometido a revisión ortográfica. Los elementos que pueden usar este atributo son los **inputs**, **textarea** o bien elementos que tengan el atributo que los haga editables. Ejemplo de uso:

```
<p contenteditable="true" spellcheck="true">Párrafo con revisión
    ortográfica.</p>
```

RESUMEN

En este capítulo, revisamos el concepto de Web semántica, aprendimos el significado de cada uno de los nuevos elementos de HTML5, así como cuándo y cómo debemos utilizarlos. También entendimos la importancia semántica de los elementos que ofrece HTML5. Finalmente, conocimos todas las etiquetas que ya existían, pero que fueron redefinidas en HTML5. Asimismo, conocimos los nuevos atributos y aquellos que fueron eliminados de la nueva especificación.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Es posible usar más de una etiqueta **<nav>**?
2. Explique en qué casos se utiliza la etiqueta **<footer>**.
3. ¿Es correcto anidar una etiqueta **<header>** dentro de una etiqueta **<nav>**?
4. ¿Es correcto insertar una etiqueta **<section>** dentro de una etiqueta **<article>**?
5. ¿Para qué se utiliza el atributo **contenteditable**?
6. Explique la diferencia entre las etiquetas **<article>** y **<section>**.
7. Explique para qué se utilizan las etiquetas **<figure>** y **<figurecaption>**.
8. Explique cuál es el uso específico en HTML5 de la etiqueta **<small>**.
9. Explique en qué casos se usan las etiquetas ** ** en HTML5.
10. ¿Es posible usar más de una etiqueta **<main>** por documento?

EJERCICIOS PRÁCTICOS

1. Descargue el archivo **tp_2.jpg**.
2. Realice la maquetación en HTML5 según la referencia del archivo descargado.
3. Valide el archivo HTML en el validador de W3C.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*04 –

Multimedia

En este capítulo conoceremos las nuevas características que nos brinda HTML5 para la ejecución de audio y video a través de los reproductores nativos. También aprendremos a usar las APIs nativas de audio y video para controlarlos y personalizarlos a nuestro gusto.

- Audio y video 134
- Video en HTML5.....134
- Audio en HTML5 143
- Audio y video avanzado con JavaScript 144
- Resumen.....149
- Actividades.....150

Audio y video

La reproducción de audio y video en un sitio web siempre ha sido uno de los mayores desafíos que ha tenido que pasar el diseñador o el programador. Al momento de realizarlo, debía cuidar que el tiempo de carga del sitio no se viera reducido, que su peso no interfiriera en la carga de la página y, por sobre todas las cosas, asegurarse de que se reprodujera en todos los navegadores y dispositivos sin problemas.

Para poder reproducir un video en nuestro sitio web, hasta hace no mucho tiempo debíamos optar por subirlo a algún servicio (YouTube o Vimeo, por ejemplo) e insertar el reproductor en el sitio, sin la posibilidad de personalizar a nivel diseño más que lo que estos reproductores permitían. De esta manera, delegábamos en el reproductor la tarea de lidiar con los temas de compatibilidad y descarga del contenido. Y si queríamos alojar el video por nuestra cuenta, debíamos pensar en poner un reproductor de video en Flash y dejar de lado la compatibilidad con dispositivos móviles que no contaran con el **Flash Player.**

> EN HTML5 SE HAN AGREGADO ETIQUETAS PARA LA REPRODUCCIÓN DE AUDIO Y VIDEO

Con HTML5 esto ya no es un impedimento, ya que se han agregado etiquetas especiales para la reproducción de audio y video. Además, se han agregado múltiples métodos y propiedades en JavaScript para poder controlar dichos reproductores.

Estas etiquetas específicas llevan el nombre del tipo de contenido: **<audio></audio>** y **<video></video>**. Al utilizarlas, le estamos indicando al navegador que dentro de tales etiquetas incluiremos información referida a la reproducción de contenido multimedia de audio o de video, respectivamente.

Video en HTML5

Para trabajar con un reproductor de video de HTML5 debemos utilizar la etiqueta **<video>** con algunas etiquetas y elementos dentro de ella. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<title>Trabajando con Videos</title>
</head>
<body>
<h1>Aquí incluimos nuestro video</h1>
<video>
<source src="video.mp4" type="video/mp4"></source>
<source src="video.ogg" type="video/ogg"></source>
Su browser no es compatible para reproducir el contenido multimedia especificado
</video>
</body>
</html>
```

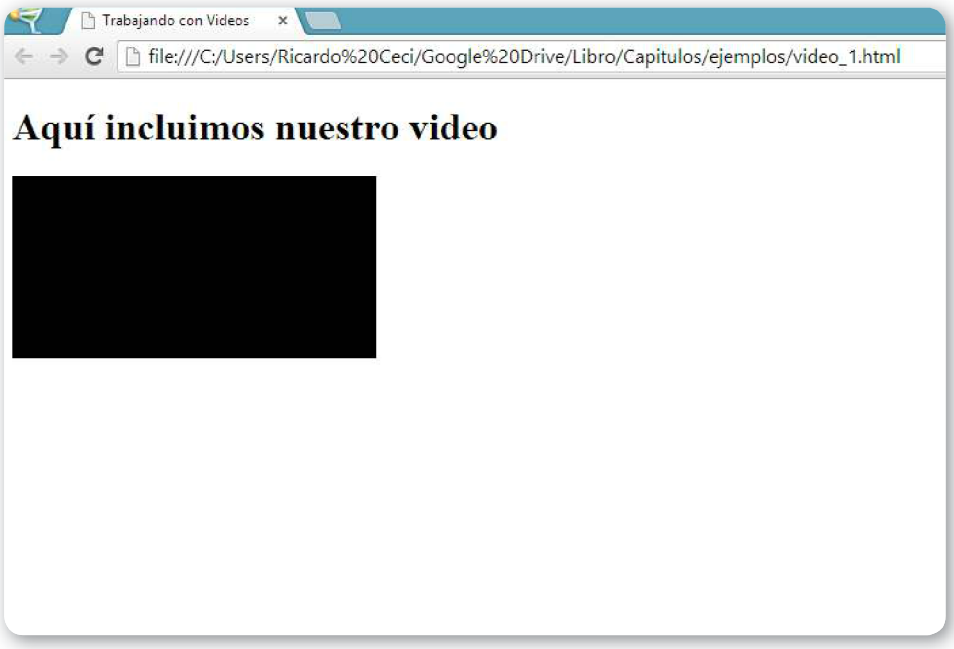


Figura 1. Nuestro reproductor de video.

En el código anterior, hemos insertado la etiqueta **<video>** y la hemos utilizado en conjunto con la etiqueta **<source>**. Esta etiqueta sirve para indicarle al navegador qué archivo de video debe reproducir y qué tipo de archivo es.

En el código puede haber más de una etiqueta **<source>** dentro de una etiqueta **<video>**; estas sirven para indicar los distintos archivos multimedia que el navegador podrá reproducir. **El browser seleccionará el primer elemento compatible y lo reproducirá.**

Veamos a continuación una tabla de formatos admitidos por los navegadores al momento de escribir la presente obra según el sitio **www.w3schools.com/html/html5_video.asp**:

COMPATIBILIDAD ENTRE FORMATOS Y NAVEGADORES

NAVEGADOR	MP4	WEBM	OGG
Internet Explorer	SÍ	NO	NO
Firefox	A partir de la versión 21 en Windows Vista/7/8 y dispositivos Android	SÍ	SÍ
Safari	SÍ	NO	NO
Opera	NO	SÍ	SÍ

Tabla 1. Formatos de video y su compatibilidad con los navegadores.

Para entender mejor la tabla anterior, debemos saber en qué consiste cada archivo:

- **MP4**: códec H264 de video y AAC de audio.
- **WebM**: códec VP8 de video y Vorbis de audio.
- **OGG**: códec Theora de video y Vorbis de audio.

Si bien el formato MP4 de video es utilizado por una gran cantidad de usuarios, no siempre es compatible con el dispositivo y el navegador en donde se está reproduciendo. Por ello, siempre es recomendable tener el video grabado y optimizado en, por lo menos, dos de los formatos nombrados anteriormente.

Ya hemos insertado nuestra etiqueta **<video>** e indicado dos formatos de video; sin embargo, el video no se reproduce aún. Esto es porque no

le hemos indicado al reproductor que lo reproduzca ni hemos activado los controles para realizar dichas acciones. Para poder realizarlo, lo único que tenemos que hacer es especificarle a la etiqueta **<video>** la propiedad **controls**, que sirve para indicar que, además del reproductor, muestre los controles necesarios para la reproducción: pausa, volumen y visualización a pantalla completa. Veamos un ejemplo:

```
<video controls width=500>
<source src="video2.mp4" type="video/mp4"></source>
<source src="video.ogg" type="video/ogg"></source>
Su browser no es compatible para reproducir el contenido multimedia especificado
</video>
```

Activando la propiedad **controls** dentro de la etiqueta **<video>**, se comenzarán a mostrar los controles. Además, incluimos la propiedad **width=500** para fijar un ancho personalizado al control de **500px.**

Veamos cómo se ve el reproductor de video:

Figura 2. Reproductor de video de Google Chrome (arriba, izquierda).

Figura 3. Reproductor de video de Internet Explorer 11 en Windows 8.1. (arriba, derecha).

Figura 4. Reproductor de video de Mozilla Firefox (abajo, izquierda).

Como podemos observar, cada navegador muestra el reproductor con el estilo gráfico para los controles específicos según la estética del navegador, sin necesidad de tener que personalizarlos o hacerle alguna modificación especial para adaptarlo entre versiones.

En todos los reproductores contamos con controles de **Play/Pause**, de **Búsqueda** (la barra que indica en qué posición del video estamos, la cual podemos manipular para ir hacia adelante o atrás en la reproducción), **Volumen** y **Visualización a pantalla completa.**

Otros controles que pueden llegar a aparecernos, siempre y cuando estén disponibles, son los botones de subtítulos y de selección de idioma de subtítulo que se va a reproducir.

Subtítulos en el video

HTML5 ofrece la posibilidad de agregar subtítulos de manera muy sencilla. La etiqueta que hay que utilizar es **<track>**, además de agregar descripciones y añadir capítulos de video y audio.

Los subtítulos para los videos que se reproducirán en el reproductor de HTML5 son archivos de texto que contienen diversos **cues** o pistas. Estos cues pueden hacer referencia a objetos JSON, por lo que podremos modificar o realizar acciones sobre el DOM mientras se está reproduciendo contenido multimedia.

La etiqueta **<track>** está disponible, al momento de escribir esta obra, en Internet Explorer 10, Chrome, Opera y Safari. Firefox aún no es compatible.

¿Cómo es un archivo de subtítulos?

Un archivo de subtítulos es un archivo de texto plano con extensión **VTT**. Veamos un ejemplo de un archivo de subtítulos (**spanish.vtt**):

```
WEBVTT FILE

jugada1
00:00:03.000 --> 00:00:08.500
El jugador pasa la pelota
```

```
jugada2
00:00:08.501 --> 00:00:11.900
El jugador 7 recibe la pelota y envía un centro
```

El archivo está compuesto por una cabecera que indica que es un archivo de formato **WebVTT.**

Luego vienen los cues o pistas. La primera línea es el **ID**; en este caso tenemos dos: **jugada1** y **jugada2.**

A continuación, se indica desde dónde hasta dónde debe mostrarse el subtítulo y, finalmente, el texto a mostrar.

El formato para escribir tiempos en los archivos VTT es el siguiente: **horas:minutos:segundos.milisegundos**, con dos dígitos para las horas, los minutos y los segundos; y tres dígitos para los milisegundos, los cuales deben rellenarse con ceros en caso de ser menor la cantidad de dígitos. En nuestro ejemplo: 00:00:12.000.

Armamos nuevamente nuestro reproductor de video haciendo referencia al archivo de subtítulos. Tengamos en cuenta que deben estar subidos a internet o ejecutados desde un servidor web, ya que no pueden abrirse desde una URL que comience con **file://.**

```
<!DOCTYPE html>
<html>
<head>
<title>Trabajando con Videos</title>
</head>
<body>
```

MÚLTIPLES FORMATOS DE VIDEO

Trabajar con formatos de video y de audio ha sido siempre una tarea bastante compleja de aprender y de entender, por lo que se la ha delegado, muchas veces, a especialistas. Lo primero que debemos conocer es qué son los códecs de video y de audio para luego entender qué formato es el ideal para cada ambiente donde queramos reproducirlo. Muchas veces, el video de nuestros controles HTML5 no es reproducido por estar mal codificado o por no haber tenido en cuenta las características del sitio donde se va a reproducir.

```
<h1>Aquí incluimos nuestro video</h1>
<video controls width=500>
<source src="video.mp4" type="video/mp4"></source>
<source src="video.ogg" type="video/ogg"></source>
<track kind="subtitles" label="Subtítulos en español" src="spanish.vtt"
        srclang="es" default></track>
Su browser no es compatible para reproducir el contenido multimedia especificado
</video>
</body>
</html>
```

La etiqueta **<track>** tiene un atributo **kind**, donde debemos indicar alguna de las siguientes opciones: **subtitles**, **captions**, **descriptions**, **chapters** o **metadata**. Luego, el atributo **src** establece de dónde obtendremos el archivo, mientras que el atributo **srclang** sirve para indicar el código internacional del idioma del subtítulo, que en este caso es "**es**" (español). Escribiendo el atributo booleano **default**, indicamos que es el subtítulo por defecto. Además, con el atributo **label** indicamos, para los navegadores en los cuales es compatible, cómo debe mostrarse la opción para seleccionar ese idioma.

Ahora veamos cómo se observa en el navegador:

Figura 5. En Google Chrome se muestra el botón **CC** en el reproductor.

Figura 6. En Internet Explorer también aparece el botón **CC** y podemos desplegar la lista de opciones.

Añadir un "poster" a nuestro video

Como habremos notado, no es muy agradable estéticamente que el reproductor de video se muestre en negro cuando carga nuestra página. Para mejorarlo, tenemos dos alternativas.

La primera es hacer que el video se inicie automáticamente, como veremos más adelante en este mismo capítulo. La otra opción es poner una imagen o captura de pantalla del video. Técnicamente, a esta imagen se la llama **poster.**

Se trata de una imagen en cualquier formato de imagen apto para la Web. Para incluirla, lo único que debemos hacer es agregarle a la etiqueta **<video>** el atributo **poster**, con la referencia al archivo en cuestión. Veamos un ejemplo:

```
<video controls width=500 poster="pelota.jpg">
...
</video>
```

Veamos, entonces, cómo se ve en el navegador:

Figura 7. El reproductor de video con un poster incorporado.

Reproducción automática

Otra de las opciones que tenemos para nuestro video es, como mencionamos, que arranque la reproducción inmediatamente cuando se carga el reproductor de video. Esta acción no es recomendable ya que afecta el rendimiento y el tiempo de carga de nuestro sitio. Por esto, debemos tener especial cuidado si elegimos esta alternativa.

Para hacerlo, debemos agregar el atributo booleano **autoplay** a la etiqueta video. El código quedaría así:

```
<video controls width=500 poster="pelota.jpg" autoplay>
...
</video>
```

VALIDANDO NUESTROS ARCHIVOS VTT

En internet tenemos múltiples herramientas para validar archivos, y la validación de archivos de formato **WebVTT** no es la excepción. Una de las más recomendables es **http://quuz.org/webvtt**. Allí copiamos y pegamos nuestro texto y, en la parte inferior, nos indicará si el formato de nuestro archivo es correcto o si cometimos algún error al momento de escribir los cues.

Reproducción continua

La etiqueta **<video>**, además de los atributos enumerados, acepta otro que permite indicar si el video debe reproducirse en modo de bucle (es decir, termina y vuelve a empezar). Este parámetro se llama **loop**; si está indicado, el video se reproduce continuamente.

```
<video controls width=500 poster="pelota.jpg" autoplay loop>
...
</video>
```


Audio en HTML5

Para trabajar con audio, HTML5 nos provee de una nueva etiqueta: **<audio>**. Al igual que con la etiqueta **<video>**, debemos agregarle un conjunto de atributos y etiquetas para que funcione adecuadamente. Veamos un ejemplo de un reproductor de audio:

```
<!DOCTYPE html>
<html>
<head>
<title>Trabajando con Audio</title>
</head>
<body>
<h1>Aquí incluimos nuestro reproductor de audio</h1>
<audio>
<source src="temal.mp3" type="audio/mpeg"></source>
<source src="temal.ogg" type="audio/ogg"></source>
Su browser no es compatible para reproducir el contenido multimedia especificado
</audio>
</body>
</html>
```

Al igual que al video, dentro de la etiqueta **<audio>** hay que agregar elementos **<source>** donde debemos hacer referencia al archivo de

audio y al formato de este. De ese modo, el navegador se encargará de reproducir el primero que sea compatible.

En el caso del audio, todos los browsers de escritorio, a excepción de Opera, son compatibles con el formato MP3 de audio. Otros formatos pueden ser WAV y OGG (no soportados por Internet Explorer).

En la etiqueta de **<audio>** también disponemos de los atributos **autoplay**, **loop** y **controls**, que cumplen exactamente las mismas funciones que en la etiqueta **<video>.**

Para esta etiqueta, además, se agrega el atributo **muted**, que permite indicar que el audio estará, por defecto, muteado (en silencio).

Audio y video avanzado con JavaScript

Al ser las etiquetas **<audio>** y **<video>** de HTML5 elementos del DOM, podemos manipularlas y realizar diversas acciones con JavaScript que nos darán una experiencia de usuario muy buena. ¿Cómo hacemos para controlar con JavaScript los reproductores?

Es sencillo: lo primero que debemos hacer es darle un identificador (ID) a nuestro reproductor. Tomemos, por ejemplo, el reproductor de video incorporado en el apartado de subtítulos de este mismo capítulo, y agreguémosle un ID:

```
<!DOCTYPE html>
<html>
<head>
```

TIP AVANZADO

El elemento de video que capturamos con JavaScript podemos utilizarlo como **source** del método **drawImage** de Canvas que veremos en el **Capítulo 5**. Por ejemplo, se puede utilizar de la siguiente manera: **context.drawImage(video,0,0,width,height);**.

```
<title>Trabajando con Videos</title>
</head>
<body>
<h1>Aquí incluimos nuestro video</h1>
<video controls width=500 id="miVideo">
<source src="video.mp4" type="video/mp4"></source>
<source src="video.ogg" type="video/ogg"></source>
<track kind="subtitles" label="Subtítulos en español" src="spanish.vtt"
        srclang="es" default></track>
Su browser no es compatible para reproducir el contenido multimedia especificado
</video>
</body>
</html>
```

A partir de este momento, lo único que debemos hacer es llamar al elemento y almacenarlo en una variable de JavaScript.

```
<body>
…
<video controls width=500 id="miVideo">
…
</video>
<script>
var miVideo = document.getElementById('miVideo');
</script>
</body>
```

Veamos algunos métodos y propiedades del objeto **miVideo**:
Obtener el tiempo de reproducción actual y total:

```
var reproducido = miVideo.currentTime;
var total = miVideo.duration;
if(reproducido == total){
        alert("El video ha finalizado");
        //Podemos ocultar el reproductor o reproducir
```

```
    //otro video
}
```

Ver los estados del reproductor:

```
if(miVideo.paused){
//Si el video está pausado
}
else if(miVideo.ended){
//Si el video ha finalizado
}
else if(miVideo.seeking){
//Si el usuario está buscando en el video
}
```

Si escuchamos el evento **timeupdate**, podemos hacer un seguimiento en tiempo real de la reproducción del video:

```
function controlarVideo(){
if(miVideo.currentTime == 10){
        //se han reproducido 10 segundos, realizar
    //alguna acción
}
}
miVideo.addEventListener('timeupdate',controlarVideo);
```

Además, con JavaScript podemos acceder a ciertas propiedades del elemento y configurarlas. Generalmente, se utilizan cuando se carga el video o bien cuando se dispara un evento dentro del reproductor.

```
// Oculto los controles
miVideo.controls = false;
//Activo reproducción automática
miVideo.autoplay = true;
```

```
//Activo el loop
miVideo.loop = true;
//Muteo el video
miVideo.muted = true;
//Cambio el volume
miVideo.volume = 0.8 //Acepta valores de 0 a 1
```

De manera muy sencilla, con JavaScript podemos crear nuestros propios controles para el reproductor de video: simplemente debemos llamar a los métodos **play()** y **pause()**. Probemos, a continuación, cómo crear nuestros propios botones. Lo primero que debemos hacer es quitar el atributo **controls** de la etiqueta **<video>** y agregar los botones:

```
<!DOCTYPE html>
<html>
<head>
<title>Trabajando con Videos</title>
</head>
<body>
<h1>Aquí incluimos nuestro video</h1>
<video width=500 id="miVideo">
<source src="video.mp4" type="video/mp4"></source>
<source src="video.ogg" type="video/ogg"></source>
<track kind="subtitles" label="Subtitulos en español" src="spanish.vtt"
srclang="es" default></track>
Su browser no es compatible para reproducir el contenido multimedia especificado
</video>
</body>
</html>
```

PERSONALIZANDO VIDEOS

Si nos interesa personalizar todo el diseño de los videos, un buen comienzo es mediante la documentación que se encuentra en: **https://github.com/videojs/video.js/blob/v4.3.0/docs/guides/skins.**

Figura 8. Así se puede ver el video cuando añadimos nuestros controles y manejamos el reproductor.

Reproductores de video de terceros

Desde hace mucho tiempo hay empresas o grupos de desarrolladores que se encargan de crear reproductores de video y de audio que simplifican la tarea de reproducir, agregan más funcionalidades y aseguran compatibilidades entre dispositivos.

Si bien el reproductor de video de HTML5 es muy completo, existe otra solución en internet, llamada **VideoJS**. Se trata de un reproductor de código abierto y uso libre basado en el reproductor de HTML5, pero que permite personalizar el aspecto gráfico de los controles de manera muy sencilla. Además, en caso de que el navegador no sea compatible con el reproductor de HTML5, **automáticamente muestra un reproductor en Flash** que permitirá que el video se reproduzca.

Viene con un "skin" –o piel, como se denomina al aspecto de la interfaz– por defecto, que los diseñadores pueden descargar y personalizar a su gusto.

Para los programadores, existe una documentación completa de la API de JavaScript para este reproductor. Para descargarlo y empezar a utilizarlo, debemos ingresar a **www.videojs.com.**

Veamos un ejemplo rápido, utilizando los scripts hosteados en los servidores de VideoJS y la demo que se incluyen en su sitio:

```
<!doctype html>
<html>
<head>
<title>VideoJS</title>
<link href="//vjs.zencdn.net/4.2/video-js.css" rel="stylesheet">
<script src="//vjs.zencdn.net/4.2/video.js"></script>
</head>
<body>
<video id="example_video_1" class="video-js vjs-default-skin"
  controls preload="auto" width="640" height="264"
  poster="http://video-js.zencoder.com/oceans-clip.png">
 <source src="http://video-js.zencoder.com/oceans-clip.mp4" type='video/mp4' />
 <source src="http://video-js.zencoder.com/oceans-clip.webm"
        type='video/webm' />
 <source src="http://video-js.zencoder.com/oceans-clip.ogv" type='video/ogg' />
</video>
</body>
</html>
```

Cuando ejecutemos este código, a simple vista veremos que es el mismo reproductor de video que nos da HTML5. La diferencia radica en que podremos personalizarlo a nivel diseño utilizando las recomendaciones que nos brinda **www.videojs.com**.

RESUMEN

En este capítulo hemos repasado las nuevas opciones que nos ofrece HTML5 a la hora de reproducir contenido multimedia. Ninguna de estas características era posible de manera nativa antes de la llegada de HTML5, por lo que se debía utilizar un reproductor de video en Flash. Hoy, si bien debemos tener en cuenta temas de compatibilidad entre navegadores, contamos con la ventaja de poder personalizar el reproductor y manejar el comportamiento a nuestro gusto a través de JavaScript. Aunque a los componentes multimedia les queda todavía un largo camino por recorrer, los elementos con los que contamos ahora son más que aprovechables y nos permiten reproducir contenido audiovisual en múltiples dispositivos y plataformas.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Con qué nuevas etiquetas de HTML5 contamos para video y para audio?
2. ¿Qué etiqueta utilizamos para indicar las pistas de video o audio?
3. ¿Qué atributos obligatorios lleva esta etiqueta?
4. Enumere tres atributos de la etiqueta **<video>**.
5. ¿Cuál es la etiqueta para agregar subtítulos a un video?
6. ¿Qué formato tienen los archivos de subtítulos? ¿Cómo se utiliza?
7. ¿Por qué no es recomendable el uso del atributo **autoplay**?
8. ¿Para qué sirve el atributo **loop**?
9. Enumere tres propiedades del objeto **video** de JavaScript.
10. Enumere dos métodos del objeto **video** de JavaScript.

EJERCICIOS PRÁCTICOS

1. Inserte un elemento de video en un documento HTML que no posea controles y que se reproduzca automáticamente.
2. Agréguele subtítulos en inglés.
3. Cuando el video que esté reproduciendo llegue a la mitad, indique que se pause.
4. Agregue un elemento de audio al reproductor y, cuando llegue al 15% de la reproducción, suba el volumen.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*05

La API de dibujo Canvas

En este capítulo, nos introduciremos en la API de dibujo de HTML5 a través del elemento <canvas> que, junto a JavaScript, nos permitirá generar cualquier tipo de gráfico. También realizaremos animaciones y efectos visuales muy interesantes para nuestro sitio web. Por último, conoceremos herramientas para trabajar de forma intuitiva con este elemento, como los frameworks EaselJS y KineticJS.

▾ **Introducción al uso de Canvas 152**
Trabajar con Canvas 153
Primeros dibujos 155
Círculos 159
Bordes redondeados 160
Curvas cuadráticas 161
Curvas de Bézier 162
Relleno con gradientes 163
Trabajar con imágenes 166
Trabajar con texto 168

▾ **Canvas avanzado, animaciones y frameworks 170**
EaselJS 170
KineticJS 173

▾ **Resumen 175**

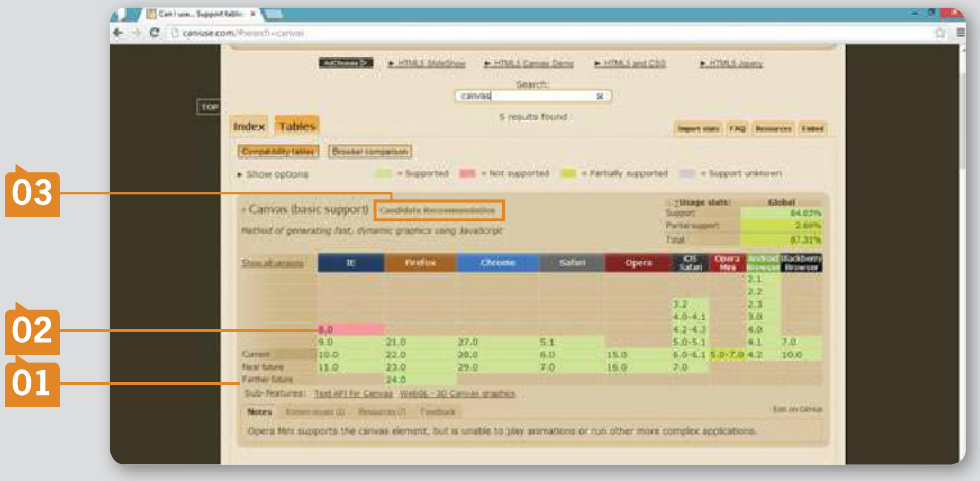
▾ **Actividades 176**

Introducción al uso de Canvas

Habitualmente nos preguntaremos si podemos utilizar un elemento al programar en HTML5. En el **Capítulo 1** introdujimos una serie de herramientas, entre ellas el sitio web **CanIUse.com**, donde se presenta una tabla muy completa que nos indica hasta qué punto es compatible el elemento HTML5 que estamos intentando utilizar; si es o no compatible con determinado navegador; y, en caso de no serlo, si existe un **polyfill**, es decir, un script o plugin que provea al navegador de las capacidades necesarias para utilizar el elemento en cuestión.

Al ser esta la primera API que analizaremos en detalle, haremos las consultas necesarias y revisaremos en qué navegadores está disponible. Para eso, ingresamos al sitio **www.caniuse.com** y utilizamos como criterio de búsqueda "canvas". El sitio nos presentará un resultado de búsqueda como el que vemos en la siguiente guía visual.

GV: UTILIZANDO CAN I USE



01 En esta columna, encontraremos la versión actual del navegador en el mercado, la versión anterior y, de estar disponibles en etapa de prueba, las próximas dos versiones.

02 En esta columna se muestra la versión y el color de fondo cambia de acuerdo a la disponibilidad de la característica en el navegador: **rojo** = no disponible; **verde** = disponible; **verde oscuro** = soporte parcial; **celeste** = soporte desconocido.

03 Aquí nos indica en qué estado de desarrollo se encuentra la característica, según la W3C.

En este caso, **<canvas>** está disponible a partir de Internet Explorer 9, Firefox 21, Chrome 27, Safari 5.1, Opera 15.0, Safari iOS 3.2, Opera Mini 5.0 de manera parcial, Android Browser 2.1 y Blackberry Browser 7.0. Nótese que para Internet Explorer 8 nos informa que no está soportado, pero que existe un **polyfill**.

Trabajar con Canvas

El término **canvas**, en inglés, significa "lienzo", pero en HTML5 se trata de un contenedor donde podremos realizar gráficos dinámicamente y –JavaScript mediante– controlar todo lo que sucede dentro de este elemento, ya que todo el trabajo que se realiza en el elemento **<canvas>** se hace con JavaScript.

Lo primero que necesitamos para trabajar es tener el elemento **<canvas>** en nuestro documento, tal como podemos ver a continuación:

```
<canvas id="miCanvas" width="200" height="200"></canvas>
```

¿TIENE POLYFILL?

Por lo general, las versiones antiguas de Internet Explorer no ofrecen compatibilidad con varias propiedades de HTML5. El sitio **CanIUse.com** muestra esta falta de soporte en color rojo, pero si situamos el puntero del mouse sobre esa celda sabremos si existe un **polyfill**, es decir, si hay un script que permita añadir la compatibilidad y lograr que nuestro sitio web pueda verse en las versiones anteriores de IE.

Si escribimos esto en nuestro documento HTML5, no notaremos ningún cambio. Pero, sin darnos cuenta, con esta etiqueta estamos creando un área rectangular sin bordes ni fondo.

Ahora, como vimos en el **Capítulo 2**, vamos a obtener el elemento creado mediante JavaScript:

```
var miCanvas = document.getElementById('miCanvas');
```

Una vez que tenemos el elemento, debemos obtener el contexto **2d**, que nos proveerá de objetos, métodos y propiedades para dibujar y manipular gráficos en el elemento **<canvas>**:

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="200" height="200">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');
miCanvas = miCanvas.getContext("2d");
</script>
</html>
```

El contexto **2d** representa un eje cartesiano en el cual el origen **0,0** está en la esquina superior izquierda, donde el eje de las X se incrementa hacia la derecha y el de la Y, hacia abajo.

Realicemos nuestro primer dibujo:

```
<head>
<title>Canvas HTML5</title>
```

```
<html>
</head>
<body>
<canvas id="miCanvas" width="200" height="200">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.fillRect(25,20,150,100);
</script>
</html>
```

En el código anterior, mediante el método **fillRect**, estamos dibujando un **rectángulo** con el color de fondo por defecto (negro), que empieza en el punto (**25,20**) y tiene **150** de ancho por **100** de alto. Entonces, el método **fillRect** debería ser ejecutado de la siguiente manera:

```
contexto.fillRect(x,y,w,h);
```

Primeros dibujos

Como todo artista, primero debemos conocer las bases de la técnica para luego profundizar y profesionalizar nuestro arte. En este caso, ya conocemos el lienzo sobre el cual vamos a trabajar; el siguiente paso es aprender **cómo dibujar** sobre él.

Teniendo en cuenta que el punto de origen de Canvas es (0,0), el contexto nos brinda una serie de métodos para poder realizar movimientos con el "lápiz" y luego rellenar esos movimientos para dejarlos marcados en nuestro lienzo. Estos métodos son: **beginPath()**, **moveTo(x,y)** y **lineTo(x,y).**

Con **beginPath()** indicamos que comienza el trazo, en tanto que con **moveTo(x,y)** movemos el puntero al punto especificado para comenzar a dibujar. Mediante **lineTo(x,y)**, trazamos una línea hasta el punto especificado.

Mientras estemos ejecutando estos métodos, no veremos nada dibujado en el canvas hasta tanto lo rellenemos. Imaginemos que con el método **lineTo** lo que hacemos es realizar los trazos que luego deben ser pintados.

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.moveTo(100,20);
miCanvas.lineTo(100,300);
miCanvas.lineTo(200,300);
miCanvas.moveTo(200,20);
miCanvas.lineTo(200,300);
</script>
</html>
```

Hasta este punto, no veremos nada en pantalla, simplemente habremos realizado un "trazo" en el canvas, sin haberlo "pintado".

Para pintarlo, podemos utilizar el método **stroke()** que, junto con la propiedad **strokeStyle**, realizará un trazo del color indicado sobre el camino recorrido. En el siguiente ejemplo, le pondremos un color azul a nuestro dibujo:

```
<!doctype html>
<html>
```

```
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.moveTo(100,20);
miCanvas.lineTo(100,300);
miCanvas.lineTo(200,300);
miCanvas.moveTo(200,20);
miCanvas.lineTo(200,300);

// Ponemos un color azul:
miCanvas.strokeStyle = '#0101DF';

// Pintamos:
miCanvas.stroke();
</script>
</html>
```

El resultado será el que vemos en la figura de la página siguiente.

DÓNDE ENCONTRAR POLYFILL

Como vimos, existe un **polyfill** para Internet Explorer 8 que podremos utilizar para usar el objeto canvas, aunque en realidad existen varios. Uno de ellos es **FlashCanvas** (**www.flashcanvas.net**), que es una combinación de JavaScript, SWF y PHP. Otro polyfill interesante es **ExploreCanvas** (**http://code.google.com/p/explorecanvas**), que consiste en un archivo que debemos incluir al documento. Tendremos que crear el elemento canvas y podremos obtener el contexto.

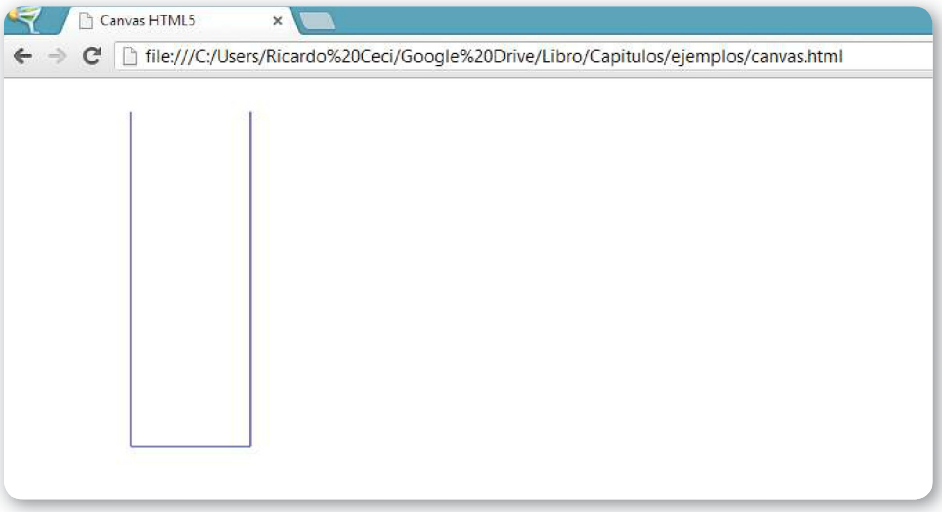


Figura 1. El resultado de nuestro primer dibujo en **Canvas**.

Otra opción para "pintar" es utilizar el método **fill**, que se complementa con la propiedad **fillStyle**. El método **lineCap** nos permitirá especificar con qué estilo terminará nuestra línea. Debemos utilizarlo en conjunto con el método **lineWidth** para darle grosor.

El método **lineCap** puede recibir tres valores posibles: **butt**, **round** o **square**. Si no se especifica ningún tipo de valor, por defecto toma el valor **butt**. Con **round** añadimos una terminación redondeada a la línea y con **square** añadimos una terminación con forma de cuadrado.

A simple vista, la diferencia entre **butt** y **square** no se nota; la diferencia radica en que las propiedades **round** y **square** incrementan el tamaño de la línea a lo largo, en una medida del ancho. Por ejemplo, si tenemos una línea de 100 px de largo por 10 px de ancho, y le agregamos una terminación **round** o **square**, la línea resultante será de 110 px de largo por 10 px de ancho, ya que se agregaron 5 px a los bordes superior e inferior. Por ejemplo:

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
```

```
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.moveTo(100,20);
miCanvas.lineTo(100,300);
miCanvas.lineTo(200,300);
miCanvas.moveTo(200,20);
miCanvas.lineTo(200,300);

//Ponemos un color azul:
miCanvas.strokeStyle = '#0101DF';

//Definimos un grosor de 20 píxeles

miCanvas.lineWidth = 20;

//Le damos una terminación redondeada a las líneas:

miCanvas.lineCap = 'round';

//Pintamos:
miCanvas.stroke();
</script>
</html>
```

Círculos

Para dibujar círculos, tenemos el método **arc**, que utiliza las siguientes propiedades: **x**, **y**, **radio**, **inicio** y **fin**. Los puntos (x,y) indicarán el centro de la circunferencia. El radio de la circunferencia se pasa como tercer parámetro y hay que indicar desde qué punto hasta qué punto se trazará la circunferencia. Por último, mediante el método **stroke** completamos el trazo pintándolo.

Bordes redondeados

Para crear bordes redondeados, utilizamos el método **arcTo().** Debemos pasarle, como parámetros, un punto de origen (x,y), un punto final (x2,y2) y un radio. Por ejemplo, hagamos otro dibujo:

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.moveTo(200,20);
miCanvas.lineTo(200,250);
miCanvas.arcTo(200,300,150,300,50);
miCanvas.lineTo(120,300);
miCanvas.lineWidth = 10;
miCanvas.strokeStyle = 'red';
miCanvas.stroke();
</script>
</html>
```

En este caso, en el lienzo de 400 px trazamos una línea desde el punto 200,20 al punto 200,250.

Desde ahí, armamos un arco, entre 2 tangentes que van desde 200,300 a 150,300 con un radio de 50 px. Luego prolongamos un poco más la línea. El resultado es el que vemos en la imagen de la página siguiente:

Figura 2. Podemos redondear bordes con la función **arcTo**.

Curvas cuadráticas

Una curva cuadrática es utilizada para representar una función –como su nombre lo dice– cuadrática, donde se indica un punto de origen, un punto final y un punto máximo de la curva.

Con el método **moveTo** indicamos dónde comenzará nuestra curva; luego, con el método **quadraticCurveTo**, indicamos hacia dónde se pronunciará la curva y hasta qué punto llegará. Veamos un ejemplo:

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
```

EL FUTURO DE LOS JUEGOS

Canvas permite generar imágenes y dibujos en el momento. Y con JavaScript se pueden modificar y capturar eventos de teclado y del cursor en tiempo real. Entonces, ¿por qué no utilizarlo para generar juegos? En el sitio **canvasdemos.com** encontraremos variados ejemplos de uso de la propiedad **canvas**, incluyendo los juegos listados dentro de la sección **Games**.

```
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');
miCanvas = miCanvas.getContext("2d");
miCanvas.beginPath();
miCanvas.moveTo(20,250);
miCanvas.quadraticCurveTo(100,0,200,150);
miCanvas.lineWidth = 10;
miCanvas.strokeStyle = 'green';
miCanvas.stroke();
</script>
</html>
```

Curvas de Bézier

Otro elemento que no puede faltar para dibujar son las curvas de Bézier. En algunos softwares de diseño las podemos usar directamente; en otros, las realizamos con la herramienta "pluma". En Canvas, podremos trazarlas mediante el método **bezierCurveTo()**, que puede recibir seis parámetros: **X0**,**Y0**; **CX0**,**CY0**; **CX1**,**CY1**; y **X1**,**Y1**, siendo **X** e **Y** los puntos iniciales y finales, y **CX** y **CY** los puntos de control con los cuales podremos darle forma a nuestra curva.

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>
```

```
<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');

miCanvas = miCanvas.getContext("2d");
miCanvas.beginPath();
// Indicamos dónde queremos comenzar nuestra curva:
miCanvas.moveTo(20,250);
// Indicamos nuestros puntos de control y los puntos iniciales y finales de
nuestra curva:
miCanvas.bezierCurveTo(20,0,400,0,200,250);
miCanvas.lineWidth=10;
miCanvas.strokeStyle='orange';
miCanvas.stroke();
</script>
</html>
```

El resultado es el siguiente:

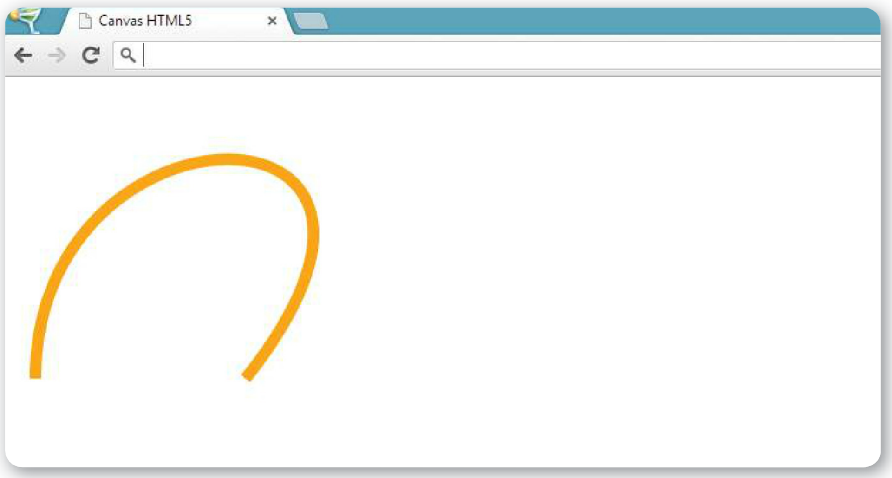


Figura 3. Dibujo de una curva de Bézier.

Relleno con gradientes

En Canvas también podemos rellenar nuestras figuras y dibujos con gradientes. Por ejemplo, para un gradiente lineal, lo primero que debemos hacer es pensarlo como una línea imaginaria que indica la dirección del gradiente y luego insertarle colores.

Para crear esta línea imaginaria, usamos el método **createLinearGradient()**, que recibe como parámetro el punto de origen y el punto final de nuestra línea. Luego, mediante la función **addColorStop(stop,color)**, agregaremos los colores por los que queremos que nuestro gradiente atraviese. El valor de **stop** puede tomar cualquier valor entre 0.0 y 1.0 y, mediante el segundo parámetro, pasamos un color.

En el siguiente ejemplo, crearemos un rectángulo, pero no lo rellenaremos. Sólo lo dibujaremos y, luego, sí lo rellenaremos con un gradiente. Para trazar el rectángulo, utilizaremos la ya conocida función **rect(x,y,ancho,alto).**

```
<!doctype html>
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="400" height="400">
</canvas>
</body>

<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");
contexto.rect(0,0,miCanvas.width,miCanvas.height);

var gradiente = contexto.createLinearGradient(0,0,miCanvas.width,miCanvas.
        height);

gradiente.addColorStop(0,'red');
gradiente.addColorStop(0.5,'orange');
gradiente.addColorStop(1,'blue');
contexto.fillStyle = gradiente;
contexto.fill();
</script>
</html>
```

El resultado lo vemos en la siguiente figura:

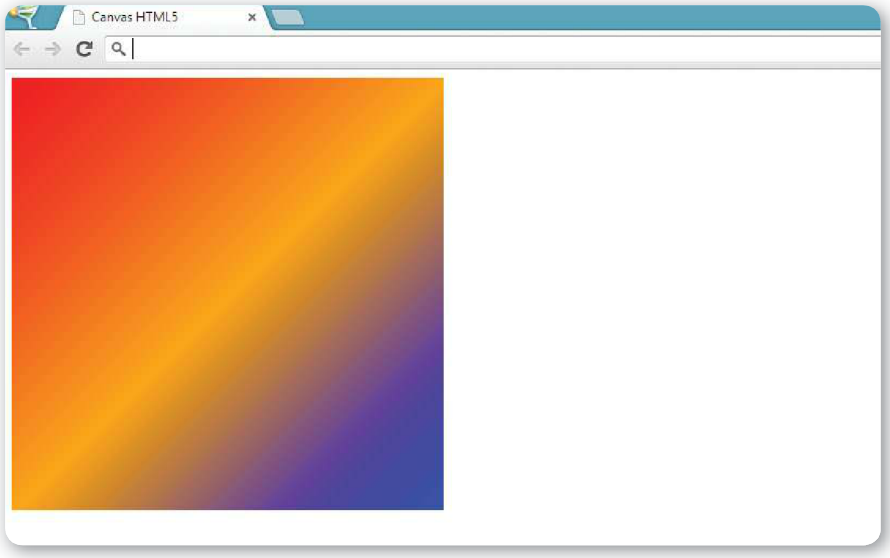


Figura 4. Gradientes lineales creados con **Canvas**.

También podemos aplicar gradientes circulares. Para ello, debemos pensar en dos círculos, uno dentro de otro, donde el origen del gradiente es el círculo menor y se va rellenando hacia el círculo mayor con los colores definidos.

La función para crear gradientes de este tipo es **createRadialGradient (x0,y0,r0,x1,y1,r1)**; donde **x0**, **y0** y **r0** son los puntos de origen del círculo 1 y su radio, en tanto que **x1**, **y1** y **r1** son los puntos de origen del círculo 2 y su radio.

```
<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");
contexto.rect(0,0,miCanvas.width,miCanvas.height);

var gradiente = var gradiente = contexto.create
        RadialGradient(200,50,10,238,50,300);
gradiente.addColorStop(0,'red');
gradiente.addColorStop(0.5,'orange');
gradiente.addColorStop(1,'blue');
contexto.fillStyle = gradiente;
```

```
contexto.fill();
</script>
```

En la siguiente imagen vemos el resultado:

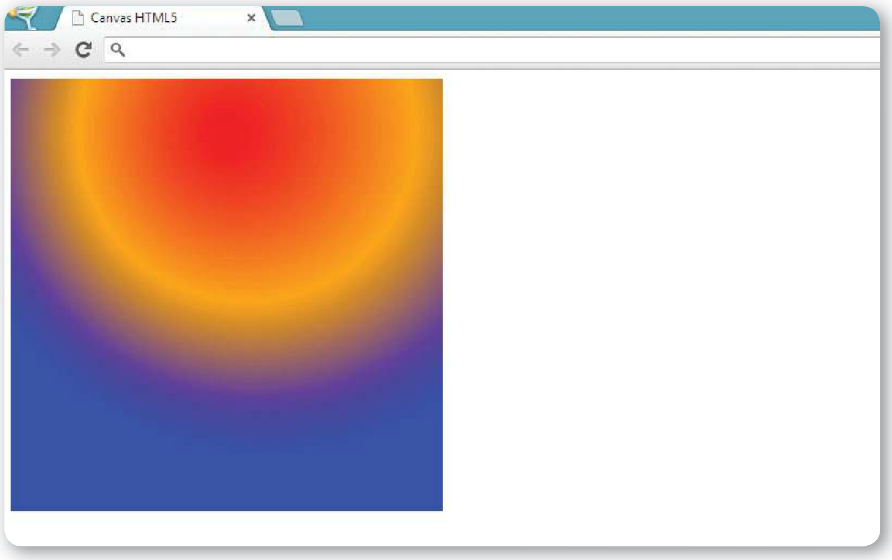


Figura 5. Gradiente circular.

Trabajar con imágenes

El manejo de imágenes en Canvas se lleva a cabo mediante el método **drawImage()**, que requiere como parámetro un objeto del tipo **imagen** y un punto donde queremos dibujar la imagen. Este punto que pasamos por parámetro será el punto superior izquierdo desde donde se comenzará a dibujar la imagen.

Si lo deseamos, podemos modificar **el tamaño de la imagen** pasando, como tercer parámetro opcional, el método **drawImage()**, que permite configurar el ancho y el alto de la imagen. Veamos un ejemplo:

```
<script type="text/javascript">
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");
var objImagen = new Image();
var ancho = 211;
```

```
var alto = 45;

objImagen.onload = function(){
 contexto.drawImage(objImagen,20,20,ancho,alto);
}
objImagen.src ='http://www.redusers.com/noticias/wp-content/themes/ru60//images/logo-redusers-header.jpg';
</script>
```

Además, podemos aplicar modificaciones a las imágenes, como un **crop** o recorte, por ejemplo. Para aplicar un crop a una imagen, lo único que tenemos que hacer es pasarle al método **drawImage()** seis parámetros adicionales que indicarán el área de la imagen original que queremos recortar y el área de destino donde queremos ubicar lo recortado con sus respectivos anchos y altos.

Supongamos que queremos hacer un recorte como el de la siguiente imagen:

Figura 6. Extracción de la marca USERS del logo de RedUSERS.

El código a escribir sería el siguiente:

```
<script>
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");
var objImagen = new Image();
var ancho = 211;
var alto = 45;
```

```
objImagen.onload = function(){
 contexto.drawImage(objImagen,75,0,136,35,10,10,125,60);
}

objImagen.src =' http://www.redusers.com/noticias/wp-content/themes/ru60//
        images/logo-redusers-header.jpg ';
</script>
```

Trabajar con texto

En Canvas también podemos escribir texto y, para hacerlo, tenemos dos métodos disponibles en el contexto: **fillText()**, **strokeText()** y la propiedad **font**.

Para poder escribir un texto, primero debemos configurar la fuente y su estilo, que puede ser: normal, itálica (*italic*) o negrita (*bold*). Por defecto, el estilo es normal. La fuente puede ser cualquiera admitida por CSS; incluso, podemos utilizar la propiedad **@font-face** de CSS para utilizar familias de fuentes personalizadas. Para ello, debemos primero definirlas en la hoja de estilos y luego llamarlas desde esta propiedad. Esto lo veremos en profundidad en el **Capítulo 9**, dedicado a la Interfaz de Usuario.

```
<script>
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");

contexto.font = 'italic 24pt Verdana';
```

IMÁGENES EN CANVAS

Con canvas podremos trabajar y generar imágenes, o cargarle imágenes en tiempo real a nuestro lienzo. Estas imágenes podrán provenir de cualquier origen, como la cámara de un dispositivo mediante la API de media que vimos en el **Capítulo 4**, capturas de videos, imágenes subidas por el usuario y otros elementos. También podremos generar un editor de imágenes propio.

```
contexto.fillText('Hola Mundo Canvas!',20,20);

</script>
```

Como podemos observar, el método **fillText()** recibe como parámetro el texto y un punto x,y donde empezar a dibujar el texto.

Modifiquemos ahora las propiedades del texto. Al igual que cualquier trazo en Canvas, podemos definir un relleno con la propiedad **fillStyle**:

```
<script>

var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");

contexto.font = 'italic 24pt Verdana';
contexto.fillStyle = 'red';
contexto.fillText('Hola Mundo Canvas!',20,20);

</script>
```

Figura 7. Texto escrito en Canvas.

Además, disponemos del método **strokeText()** que trazará el texto pasado como parámetro y no lo rellenará.

```
<html>
<head>
<title>Canvas HTML5</title>
</head>
<body>
<canvas id="miCanvas" width="500" height="500">
</canvas>
</body>

<script>
var miCanvas = document.getElementById('miCanvas');
var contexto = miCanvas.getContext("2d");

contexto.font = 'italic 24pt Verdana';
contexto.strokeStyle = 'blue';
contexto.strokeText('Este texto no lleva relleno',0,40);

</script>
</html>
```


Canvas avanzado, animaciones y frameworks

Veamos algunas herramientas para mejorar nuestro trabajo con el lienzo, y software para realizar animaciones excepcionales.

EaselJS

Una de las librerías líderes del mercado para el trabajo con Canvas es **EaselJS**, que se puede descargar desde **www.createjs.com**. En este sitio, además, encontraremos otras librerías para complementar el trabajo con Canvas y HTML5. EaselJS está siendo patrocinado por grandes marcas que apuestan a la creación de una librería que permita trabajar de manera sencilla con Canvas.

Para trabajar en Canvas con EaselJS debemos crear un **stage** (escena) y trabajar sobre él, agregándole elementos como "hijos". EaselJS busca proveer una API que sea similar al desarrollo con Adobe Flash.

Veamos unos ejemplos. Descargamos EaselJS, lo incorporamos en nuestro documento y agregamos una forma:

```
<html>
<head>
<title>Canvas HTML5</title>
<script src="js/easeljs-0.6.1.min.js" type="text/javascript"></script>
</head>
<body>
<canvas id="miCanvas" width="500" height="500">
</canvas>
</body>

<script>
var stage = new createjs.Stage("miCanvas");
estrella = new createjs.Shape();
estrella.graphics.beginFill("red").drawPolyStar(150, 150, 40,5,0,40);
stage.addChild(estrella);
stage.update();

</script>
</html>
```

En este ejemplo, creamos un objeto **stage** del tipo **createJS Stage** y un objeto "estrella" del tipo **createjs.Shape()**. Luego, utilizamos la propiedad **graphics** del objeto estrella y le indicamos que comience a rellenar con color rojo –mediante **beginFill("red")**– y que, a continuación, dibuje.

El dibujo que le pedimos que realice es un dibujo del tipo **Polystar**, que lleva los siguientes parámetros:

```
drawPolyStar(x,y,radio,lados/puntos,pointSize,angulo);
```

Dibuja una estrella si **pointSize** es mayor a 0, o un polígono regular si **pointSize** es 0 con el número especificado de lados.

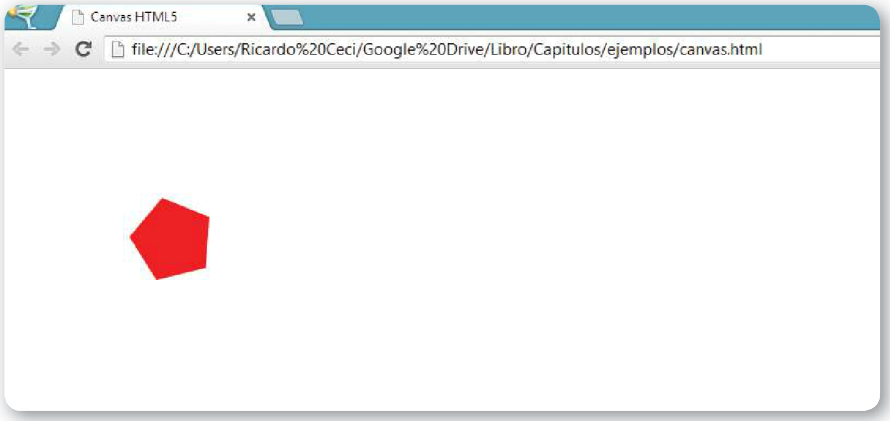


Figura 8. Polígono dibujado con **EaselJS**.

Utilizar imágenes y animaciones

EaselJS nos provee de un objeto **Ticker** que se encarga de recibir eventos del tipo **tick**, que son como latidos del corazón para el **stage.** Podremos escucharlos y realizar acciones cada vez que se recibe este evento. Como ejemplo, movamos un elemento dentro del canvas:

```
<script>
var stage = new createjs.Stage("miCanvas");
var imagen = new Image();
var logo = new createjs.Bitmap(imagen);
imagen.onload = function (){
   stage.addChild(logo);
   stage.update();
}
imagen.src = 'http://www.w3.org/html/logo/downloads/HTML5_Logo_512.png';
var mover = 0;
createjs.Ticker.addEventListener('tick',function(e){
        mover += 5;
        if(mover > stage.canvas.width){
        //Si llegamos al ancho, reiniciamos el contador
                mover = 0;
```

```
                }
        logo.x = mover;
        stage.update();
});

</script>
```

KineticJS

Otra herramienta que podemos utilizar para trabajar con Canvas es **KineticJS**, que se obtiene desde el sitio web **www.kineticjs.com**. En KineticJS, también disponemos de un objeto **stage**. Este objeto contiene **layers**, que son las capas que contienen los elementos, como círculos, rectángulos, líneas y polígonos.

Veamos un ejemplo:

```
<html>
<head>
<title>Canvas HTML5</title>
<script src="js/kineticjs.js" type="text/javascript"></script>
</head>
<body>
<div id="miCanvas" width="500" height="500">
</canvas>
</body>

<script>
var stage = new Kinetic.Stage({
        container:'miCanvas',
        width:500,
        height:500
});
var capa1 = new Kinetic.Layer();
var fondo = new Kinetic.Layer();

var rectangulo = new Kinetic.Rect({
```

```
        x:250,
        y:100,
        width:200,
        height:150,
        fill:'red',
        stroke:'green',
        strokeWidth:5
});

var circulo = new Kinetic.Circle({
        x:0,
        y:0,
        radius:300,
        fill:'black',
        stroke:'orange',
        strokeWidth:10
});
var texto = new Kinetic.Text({
       x: 20,
        y: 20,
       text: 'Esto es KineticJS',
       fontSize: 30,
       fontFamily: 'Verdana',
       fill: 'green'
     });
fondo.add(circulo);
capa1.add(rectangulo);
capa1.add(texto);
stage.add(fondo);
stage.add(capa1);
stage.draw();
</script>
</html>
```

Aquí vemos que es necesario tener un **div** con un **id**, donde insertaremos nuestro **stage**. Veamos el resultado con la consola:

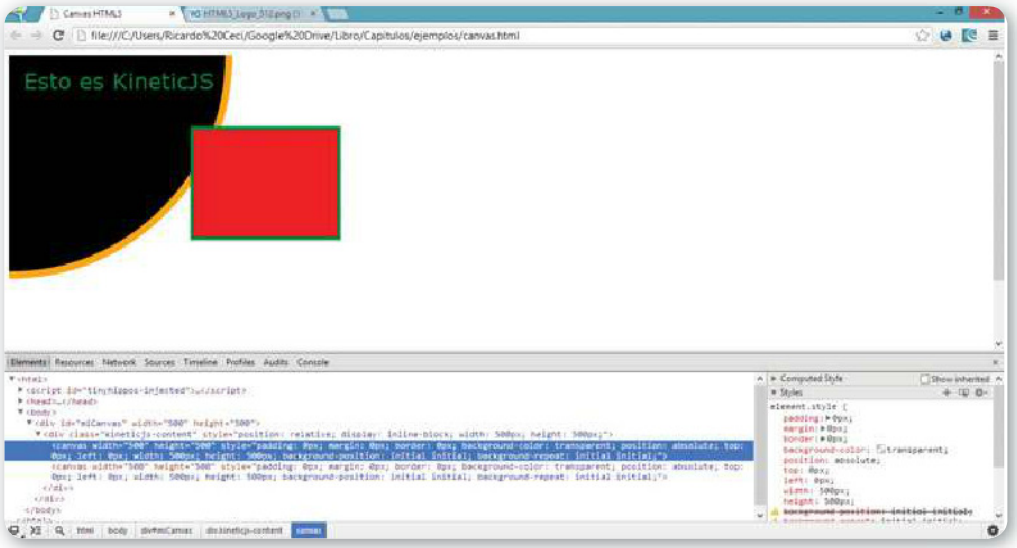


Figura 9. Figuras, texto y capas creados con **KineticJS**.

Como vemos en la consola, por cada layer o capa, nos agrega un elemento **canvas** dentro de nuestro **div**. El trabajo con Canvas y KineticJS se simplifica bastante: sólo debemos conocer sus propiedades y ubicar los elementos de pantalla en capas.

RESUMEN

Hasta ahora, realizar dibujos "al instante" en HTML y animarlos era prácticamente imposible, por lo que debíamos hacerlo mediante herramientas como Flash. En este capítulo vimos que HTML5 pone a nuestra disposición un lienzo en el cual podemos generar estos dibujos, insertar imágenes y textos e, incluso, animarlos. También vimos cómo controlar este lienzo desde JavaScript para reproducir imágenes, animaciones y todos sus contenidos en todos los dispositivos que soporten el elemento canvas. De esta forma, le daremos el poder a HTML5 de reproducir animaciones en dispositivos que no cuenten con Flash Player.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione dos tipos de gradientes posibles de realizar con Canvas.
2. ¿Se debe utilizar Canvas para suplir la falta de tipografías web?
3. ¿Cuáles son los métodos disponibles para "pintar" un trazo?
4. ¿Cuál es el método para dibujar un círculo?
5. ¿Qué diferencia hay entre los métodos **fillStyle** y **strokeStyle**?
6. ¿Qué método debemos invocar luego de utilizar **lineTo** para que éste sea "pintado" en pantalla?

EJERCICIOS PRÁCTICOS

1. Dibuje una estrella en Canvas.
2. Con algún framework (KineticJS o EaselJS), realice una marquesina que desplace de derecha a izquierda su nombre.
3. De una foto de su colección, mediante Canvas, recorte sólo el rostro de una persona que esté en la foto.
4. Realice un círculo por cada clic que haga el usuario sobre el Canvas (deberá escuchar el evento clic).

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*06 –

Formularios

En este capítulo, conoceremos las nuevas posibilidades que nos ofrece HTML5 para trabajar con formularios. Aprenderemos a utilizar campos especiales que mejorarán la experiencia del usuario: descubriremos, entre otras cosas, cómo lograr que en un campo sólo se puedan ingresar caracteres para números de teléfono y cómo validar números de tarjetas de crédito y direcciones de e-mail sin la necesidad de implementar un plugin de JavaScript.

- **Trabajar con formularios 178**
  - Elementos 178
- **Nuevos elementos en HTML5 ..... 187**
  - Compatibilidad 187
  - Inputs 188
  - Trabajar con fechas 194
  - Nuevos atributos 198
  - Autofocus 199
  - Realizar validaciones 202
- **Resumen 205**
- **Actividades 206**

Trabajar con formularios

Un formulario es, en HTML, un elemento que contiene determinados campos para que el usuario envíe información y nosotros la recibamos de manera estructurada.

A lo largo del tiempo y en la evolución de la programación web, los formularios han sido –y son actualmente– la herramienta indispensable del programador web para interactuar con el usuario.

En ellos, el usuario completará, por ejemplo, su nombre, apellido, dirección de correo electrónico, fecha de nacimiento y demás datos que sean necesarios.

Así como son una gran herramienta, durante mucho tiempo fueron un gran dolor de cabeza para el programador, ya que, si bien su uso e implementación –como veremos a continuación– es muy sencillo, a la vez el usuario tiene mucha libertad para completar la información solicitada.

Supongamos este simple ejemplo: solicitar fecha de nacimiento de una persona. El usuario puede completarnos en el campo del formulario con el formato de fecha **10-01-1988**, si es que está en un país hispanoparlante, o bien puede completarnos **01-10-1988**, si se encuentra en los Estados Unidos. También puede ingresar **1988-01-10** o **1988-10-01**; todos estos formatos de fechas son válidos siempre y cuando sepamos de antemano cómo los ingresó o le indiquemos la forma de ingresarlos. Dado que no somos magos ni mentalistas, debemos aprovechar nuestras habilidades como programadores para que el usuario ingrese el dato tal como lo necesitamos.

Elementos

Para trabajar con formularios, primero debemos indicarle al documento HTML que vamos a utilizarlos. Para ello, utilizamos las etiquetas **<form>** y **</form>**, y dentro de ellas incorporamos los campos que queramos que el usuario complete. Repasemos los campos más utilizados:

Input

Uno de los campos más importantes dentro de los formularios es el **input**, un campo de texto al que, mediante ciertos atributos, le podremos configurar su comportamiento.

ATRIBUTOS GENÉRICOS PARA INPUT	
ATRIBUTO	DESCRIPCIÓN
Type	Indica el tipo de dato a ingresar.
Name	Indica el nombre del input.
Id	Indica el ID del input.
Size	Indica la cantidad de caracteres que se podrán escribir en el input.
Value	Indica el valor por defecto del input.

Tabla 1. Atributos del elemento **`Input`**.

Los inputs pueden ser precedidos por una etiqueta (**<label></label>**). Por ejemplo:

```
<form>
        <label for="txtNombre">Nombre:</label><input type="text"
                name="txtNombre"/>
</form>
```

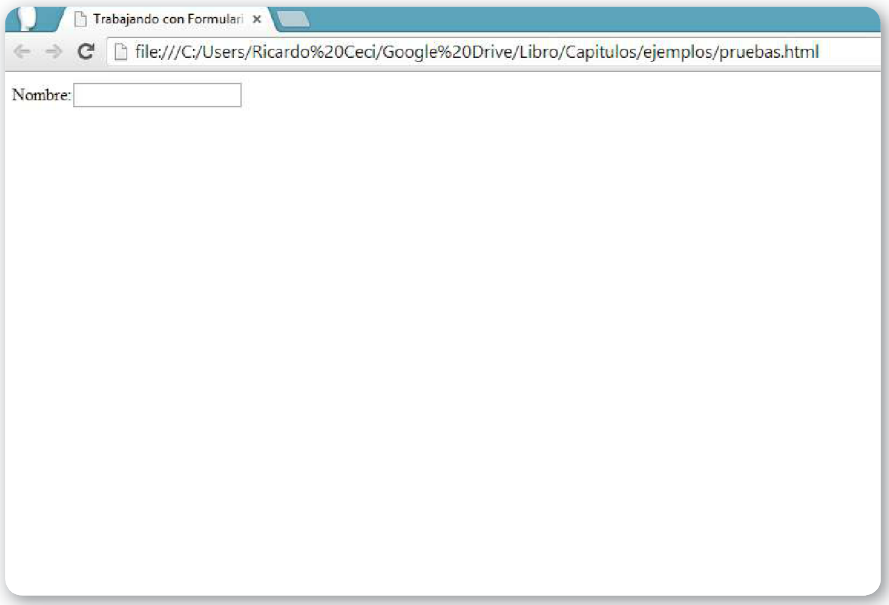


Figura 1. Nuestro primer input de tipo **`texto`**.

Hagamos un repaso por los distintos tipos de inputs soportados en HTML para ver, luego, los nuevos tipos de inputs que incorpora HTML5.

INPUTS SOPORTADOS EN HTML

TIPO	DESCRIPCIÓN
Text	Permite insertar texto plano.
Password	Permite ingresar texto, pero al usuario le muestra los caracteres con una máscara.
File	Permite seleccionar un archivo de nuestra computadora.
Button	Muestra un botón.
Radio	Mediante un input de "opción" o "radial", el usuario podrá seleccionar una u otra opción de una lista.
Checkbox	Caja donde el usuario tildará una o varias opciones de una lista (por ejemplo, para aceptar términos y condiciones).
Submit	Es un botón encargado de enviar el formulario al servidor.
Reset	Es un botón encargado de limpiar todos los campos del formulario.
Hidden	Es un campo oculto que sirve a los programadores para enviar datos adicionales al formulario sin que el usuario los vea en pantalla.

Tabla 2. Tipos de inputs soportados en HTML 4.01 y XHTML 1.1.

Veamos un ejemplo:

```
<!doctype html>
<html>
<head>
<title>Trabajando con Formularios</title>
<style>
        label {
                display:block;
        }
```

```
</style>
</head>

<body>
<form>
<label for="txtNombre">Nombre:</label><input type="text"
        name="txtNombre"/>
    <label for="txtApellido">Apellido:</label><input type="text"
        name="txtApellido"/>
    <label for="txtEmail">Email:</label><input type="text"
        name="txtEmail"/>
    <label for="clave">Contraseña:</label><input type="password"
        name="clave"/>
    <label for="foto">Seleccione una foto:</label><input type="file"
        name="foto" />
    <label for="sexo">Sexo:</label><input type="radio" name="sexo"
        value="mujer" />Mujer<input type="radio" name="sexo"
            value="hombre"   >Hombre<br/>
    Acepto los términos y condiciones <input type="checkbox"
        name="aceptaTyC" value="si" /><br/>
    <input type="hidden" name="campoOculto" value="Esto no lo ve el
        usuario" />
    <input type="submit" value="Enviar"><input type="reset" value=
        "Limpiar campos"/>
    </form>
</body>
</html>
```

ETIQUETA FORM

La etiqueta **form** es una etiqueta que va a contener los elementos del formulario. En esta etiqueta, además, se definen diferentes atributos y formas que indicarán cómo se enviará al servidor la información recopilada. Para saber más sobre el tratamiento de la información una vez enviado el formulario, recomendamos leer el libro **PHP 5** de esta misma editorial.

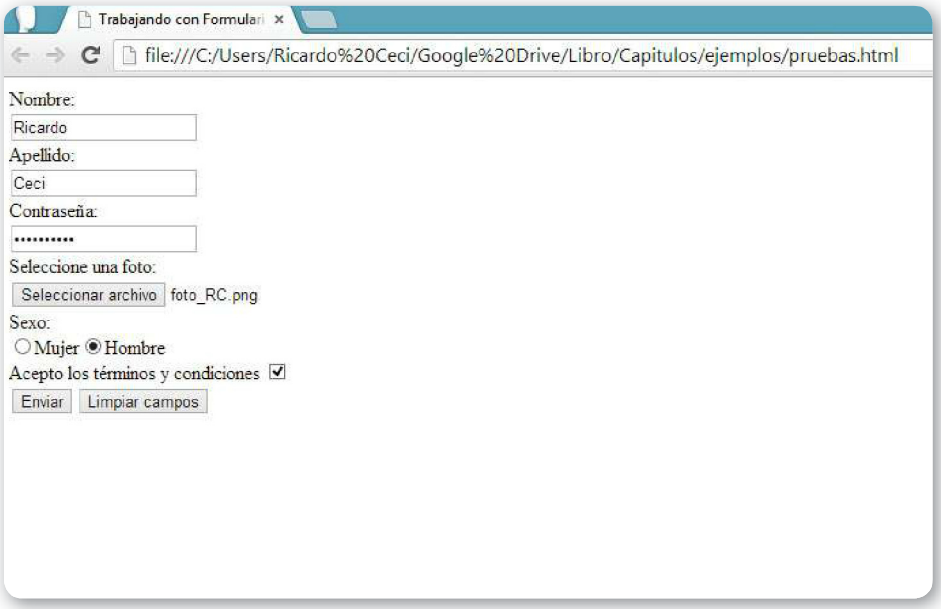


Figura 2. Nuestro formulario con los distintos tipos de inputs.

Radio Buttons y Checkboxes

Los inputs del tipo **radio** y **checkbox** (casillas de verificación) son aquellos con los cuales el usuario puede seleccionar una opción de una lista o tildar varias opciones, respectivamente.

Para estos tipos de inputs tenemos que tomar ciertos recaudos. Por ejemplo, para que los **radio buttons** se comporten como tales, es decir, permitan seleccionar entre una u otra opción, **deben tener el mismo valor** en el atributo **name**. Caso contrario, el navegador los interpretará como dos grupos de opciones distintos y presentará un comportamiento erróneo. Veamos un ejemplo.

```
<input type="radio" name="selHombre" />Hombre
<input type="radio" name="selMujer" />Mujer
```

Ahora, si ambos tienen el mismo atributo **name**, ¿cómo hacemos para saber qué opción seleccionó el usuario? Esto lo logramos utilizando el atributo **value**, que permitirá que el navegador envíe al servidor únicamente el valor del **radio button** seleccionado y no el otro. Por esta razón, en nuestro ejemplo indicamos el mismo valor en el atributo **name** con distinto valor en el atributo **value.**

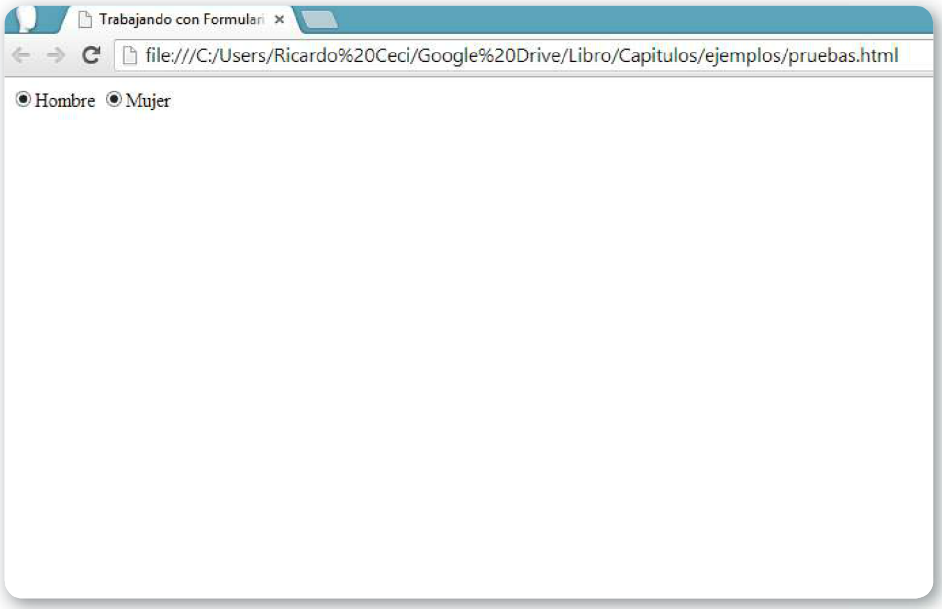


Figura 3. Comportamiento erróneo de los radio buttons.

Algo similar sucede con los **checkboxes**. Estos inputs pueden tomar dos valores posibles: **seleccionado** (checkeado/tildado) o **no seleccionado** (no checkeado/tildado). Mediante el atributo **value,** definimos cuál será el valor que recibiremos cuando este campo esté tildado. Si no definimos el atributo **value**, por defecto recibiremos el valor **on** si está checkeado y **off** si no está checkeado.

Otros elementos de formularios

ELEMENTOS ADICIONALES

ELEMENTO	DESCRIPCIÓN
Select	Lista desplegable de opciones.
TextArea	Área para ingresar texto.

Tabla 3. Elementos adicionales de formularios HTML 4.01 y XHTML 1.1.

Listas

Las listas son elementos que permiten que el usuario seleccione una o varias opciones de un conjunto. Para utilizarlas, debemos combinar

dos elementos de HTML: **<select>** y **<option>**. El primero indicará que lo que se va a mostrar es una lista desplegable, mientras que cada **<option>** será una opción para elegir. Ejemplo:

```
<label for="equipo">Seleccione su equipo favorito:</label>
<select name="equipo">
        <option>Boca Juniors</option>
        <option>River Plate</option>
        <option>Barcelona FC</option>
</select>
```

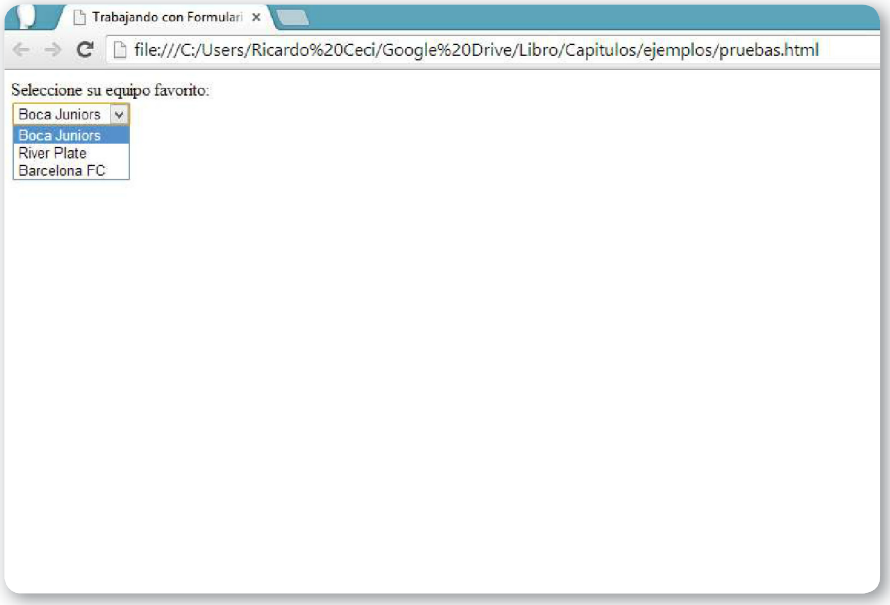


Figura 4. Así se verá nuestra lista desplegable.

Si al elemento **<select>** le agregamos el atributo **multiple**, el usuario podrá seleccionar más de una opción a la vez al mantener presionada la tecla **CTRL** y, simultáneamente, tocar con el mouse cada elemento que desea seleccionar. Ejemplo:

```
<label for="selFrutas">Seleccione las frutas que le gustaría comer</label>
<select name="selFrutas" multiple>
        <option>Manzana</option>
```

```
        <option>Banana</option>
        <option>Kiwi</option>
        <option>Naranja</option>
</select>
```

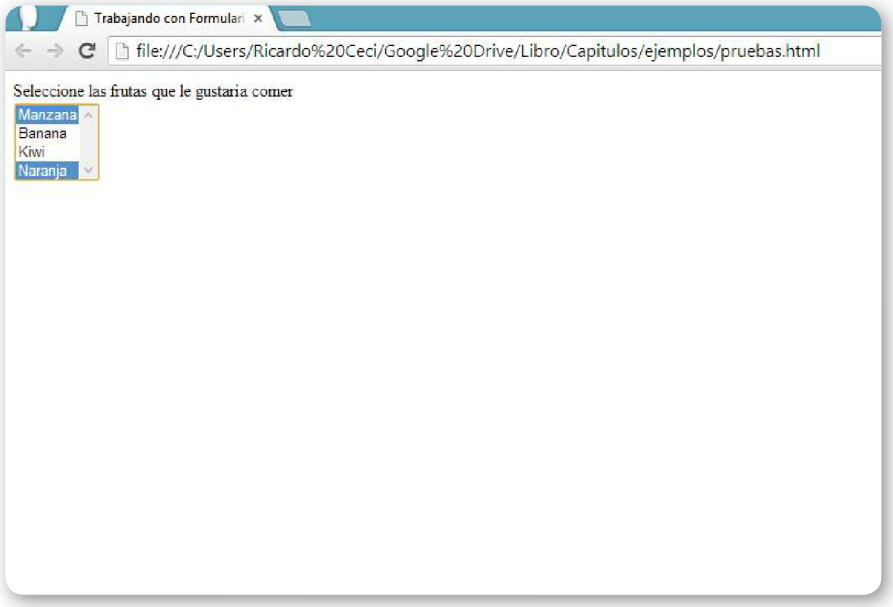


Figura 5. Así funciona una lista con selección múltiple.

También se puede indicar qué opción será la que estará seleccionada por defecto al momento de mostrar la lista al usuario. Esto se consigue agregándole el atributo **selected** a la opción que deseemos que esté seleccionada de manera predeterminada. Si no se indica este atributo, por defecto se seleccionará el primer elemento de la lista. A veces, es recomendable incluir como primer ítem de la lista la palabra "Seleccione".

```
<label for="equipo">Seleccione su equipo favorito:</label>
<select name="equipo">
        <option>Boca Juniors</option>
        <option>River Plate</option>
        <option selected="selected">Barcelona FC</option>
</select>
```

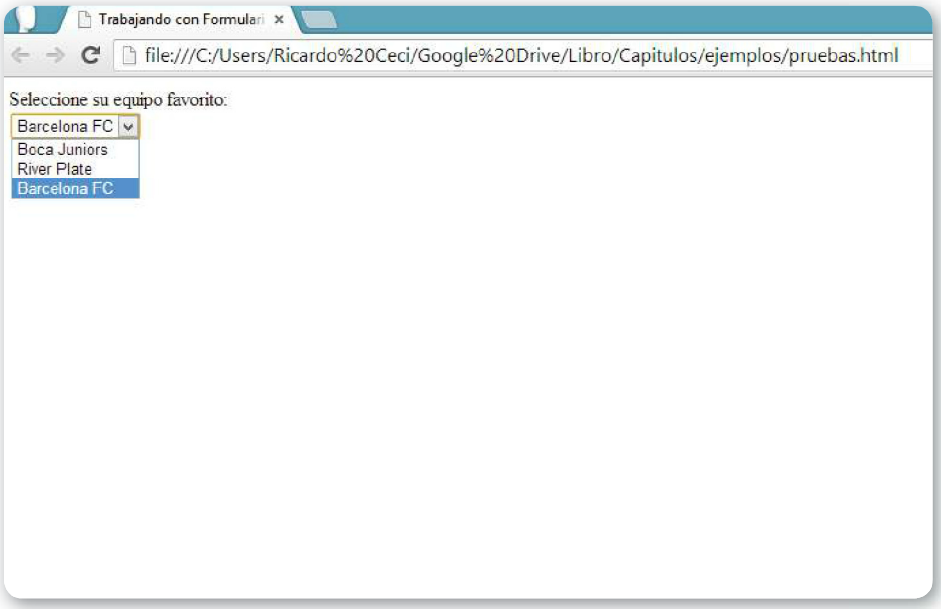


Figura 6. Al fijar una opción seleccionada por defecto, ese ítem aparecerá la primera vez que se cargue la página.

Áreas de Texto (textareas):

Se trata de campos de texto donde el usuario puede ingresar párrafos o múltiples líneas. A diferencia de los **inputs**, este elemento tiene una etiqueta propia.

El texto que se ingresará dentro de estos campos de texto no poseerá formato. Podemos darle vida a estos campos mediante plugins de JavaScript como **CKeditor**, por ejemplo.

Mediante los atributos **cols** y **rows** se puede definir la cantidad de filas y columnas disponibles en el campo. Para indicar el valor por defecto del **textarea**, en lugar de ingresarlo en el atributo **value**, deberá escribirse entre la etiqueta de apertura **<textarea>** y la de cierre **</textarea>**. Veamos un ejemplo:

```
<label for="txtMensaje">Dejenos su mensaje:</label>
<textarea name="txtMensaje"></textarea>
<label for="txtDescripcion">Comente brevemente su experiencia</label>
<textarea name="txtDescripcion" cols=25 rows=5>El servicio brindado fue
        [Completar aqui] y el personal se comportó [Completar aqui]</textarea>
```

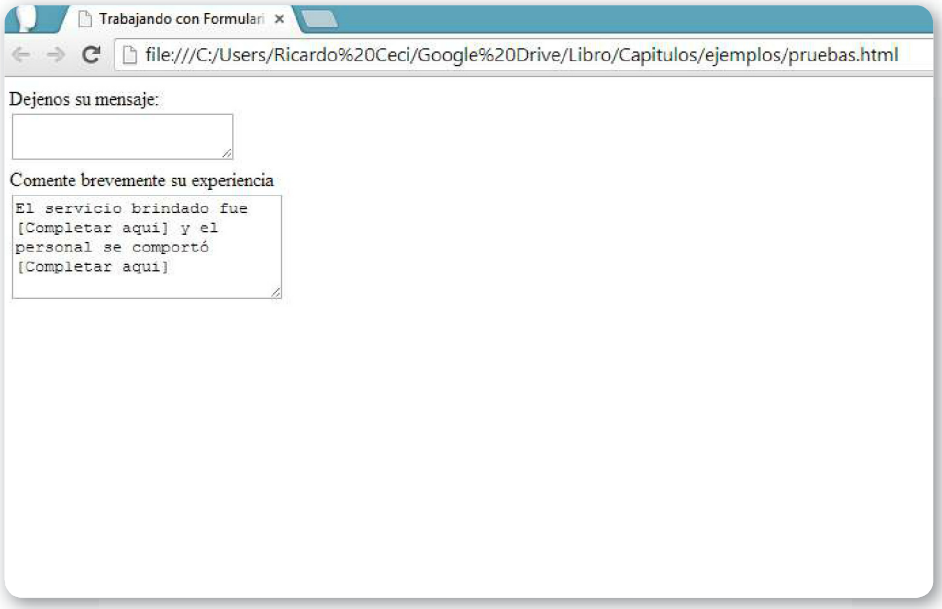


Figura 7. **Textareas** con y sin valor por defecto

Nuevos elementos en HTML5

Hasta el momento, para pedir que el usuario ingresara una dirección de correo electrónico debíamos colocar un **input** del tipo **texto** y esperar a que el usuario agregara correctamente su e-mail, o utilizar códigos en JavaScript para analizar la validez de la dirección antes de procesarla.

Actualmente, con los nuevos elementos de HTML5, esta tarea es mucho más sencilla. HTML5 nos provee de una serie nueva de inputs y de atributos que simplificarán el trabajo de las validaciones y potenciarán al máximo la experiencia del usuario.

Compatibilidad

Antes de empezar a trabajar con los nuevos tipos de inputs y sus nuevos atributos, debemos verificar la compatibilidad. Para ello, abrimos el sitio **www.caniuse.com** y tipeamos **HTML5 forms** en el cuadro de búsqueda. Como podremos observar, el soporte es muy variado: parcial en algunos navegadores, no soportado en otros. Muy pocos ofrecen compatibilidad para la totalidad de los nuevos inputs.

Para poder hacer funcionar todos los nuevos inputs en todos los browsers, vamos a requerir de un **polyfill**. En este caso, podremos utilizar **webforms2**, que podemos obtener de la siguiente URL: **https://code.google.com/p/webforms2**.

Inputs

Ahora sí, es momento de empezar a analizar los nuevos tipos de inputs incorporados en la especificación de HTML5.

Email

HTML5 incorpora el tipo de campo **email**, especificándole al navegador que, en este input, el usuario solo podrá ingresar una dirección de correo electrónico válida, y mostrará un error en caso de que se escriba algo que no sea una dirección de e-mail.

Para ejemplificar, volveremos a utilizar el formulario que hicimos como ejemplo, pero le introduciremos un pequeño cambio:

```
<!doctype html>
<html>
<head>
<title>Trabajando con Formularios</title>
<style>
        label {
                display:block;
        }
</style>
</head>
<body>
<form>
<label for="txtNombre">Nombre:</label><input type="text"
                name="txtNombre"/>
        <label for="txtApellido">Apellido:</label><input type="text"
                name="txtApellido"/>
        <label for="txtEmail">Email:</label><input type="email"
                name="txtEmail"/>
```

```
        <label for="clave">Contraseña:</label><input type="password"
                name="clave"/>
        <label for="foto">Seleccione una foto:</label><input type="file"
                name="foto" >         /
        <label for="sexo">Sexo:</label><input type="radio" name="sexo"
                value="mujer" />Mujer<input type="radio" name="sexo"
                        value="hombre"   >Hombre<br/>
        Acepto los términos y condiciones <input type="checkbox"
                name="aceptaTyC" value="si" /><br/>
        <input type="hidden" name="campoOculto" value="Esto no lo ve el
                usuario"             />
        <input type="submit" value="Enviar"><input type="reset" value="Limpiar
                campos"/>
        </form>
</body>
</html>
```

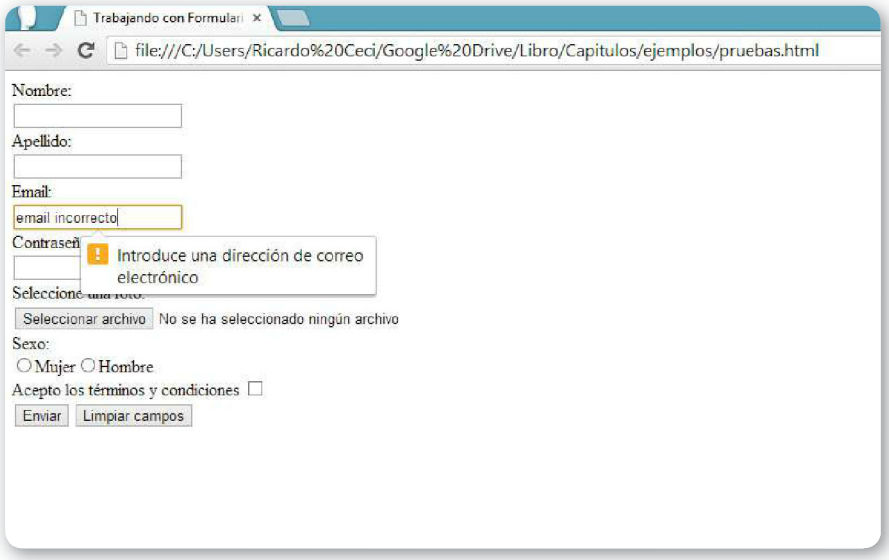


Figura 8. Ingresamos un e-mail de modo incorrecto y nos muestra un mensaje de error.

Además, el hecho de especificar el tipo de input también tendrá sus beneficios al momento de visualizar nuestro sitio web en un dispositivo móvil. Veamos qué sucede con el input del tipo **email**:

Figura 9. Así se ven los teclados al momento de completar un e-mail en Blackberry 10 y en iOS.

Como se observa en la imagen anterior, indicar un input del tipo **email** permitirá que los teclados virtuales de los dispositivos móviles **ofrezcan mejor accesibilidad** a las teclas necesarias para escribir una dirección de correo electrónico, como el arroba o el punto.

Búsquedas (search)

Al determinar que un input es del tipo **search**, le indicamos al navegador que se tratará de un campo para realizar búsquedas, para que lo interprete como tal y ubique, por ejemplo, un botón con una **x** que servirá para borrar el contenido. Veamos el siguiente código:

MONITOREAR CAMBIOS CON JAVASCRIPT

A cualquier elemento de los formularios podemos agregarle los atributos **onclick**, **onkeyup**, **onkeydown**, **onchange**, **onfocus** y **onblur** para poder monitorear los cambios desde JavaScript. Por ejemplo, podremos, mediante AJAX, monitorear el cambio de un **select** de países y completar otro **select** con las provincias, de acuerdo al país seleccionado.

```
<label for="txtBusqueda">Ingrese la palabra a buscar:</label><input
        type="search" name="txtBusqueda"/>
```

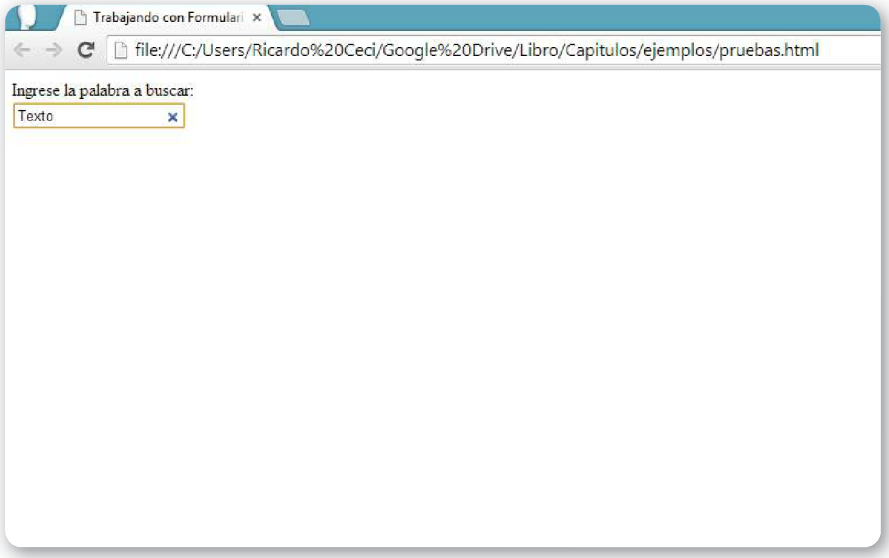


Figura 10. El navegador nos agrega la opción de borrar el contenido.

En el sistema operativo iOS, por ejemplo, al especificar este tipo de campo, se mostrará el input con bordes redondeados y, además, nos cambiará el botón **Enviar** del teclado por el botón **Buscar.**

A los usuarios del navegador Google Chrome podemos darles la posibilidad de escribir en los inputs mediante el dictado en voz alta del término a buscar. Es muy fácil de hacer: solo hay que agregar el atributo **x-webkit-speech** a nuestro input, como vemos en el ejemplo:

```
<label for="txtBusqueda">Ingrese la palabra a buscar:</label><input
        type="search" name="txtBusqueda" x-webkit-speech/>
```

El navegador automáticamente agregará el micrófono dentro del campo y, si el usuario presiona ese micrófono, podrá dictar con su voz lo que desea buscar. De este modo, Chrome comparará lo que dijo el usuario con un patrón de voz que posee Google en sus servidores y lo escribirá en el campo. Al momento de escribir este libro, la función solo está disponible en Google Chrome.

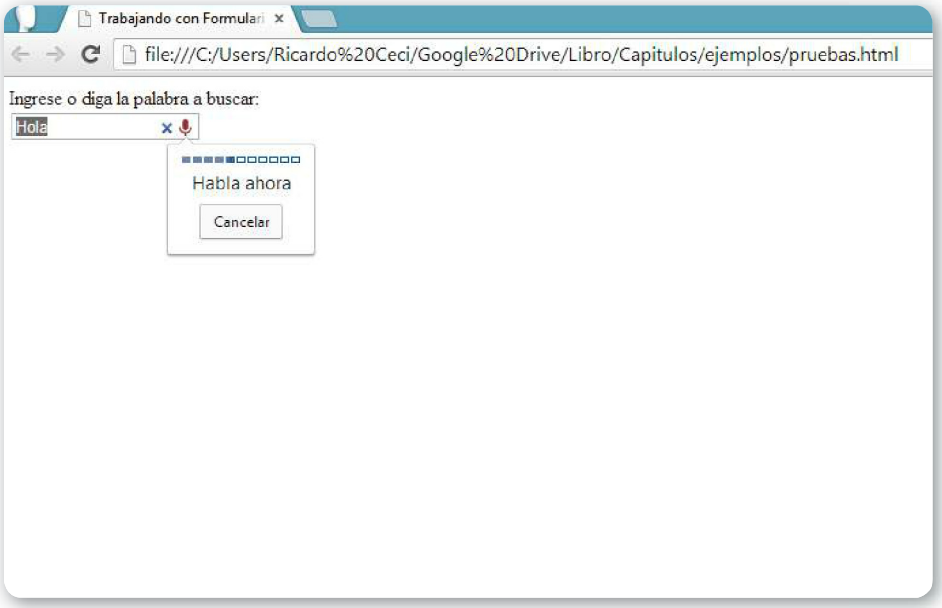


Figura 11. En Chrome podemos permitir el uso de la voz para los inputs.

Teléfonos (tel)

A través de este tipo de inputs, solo se aceptarán caracteres válidos para un teléfono y, en dispositivos móviles, se mostrará el mismo teclado numérico que utilizamos cuando deseamos realizar una llamada. A continuación, podemos ver un ejemplo de la utilización de este tipo de input:

```
<label for="txtTelefono">Ingrese su teléfono:</label>
<input type="tel" name="txtTelefono"/>
```

TIPOS DE CAMPOS

El uso de los nuevos tipos de input de HTML5 nos permite mejorar la experiencia de usuario, porque estos ayudan a que el usuario ingrese en cada campo lo que estamos esperando, ahorrando tiempo de desarrollo y evitando usar verificaciones mediante otros lenguajes, como JavaScript. Por ejemplo, si esperamos un número, podemos impedir que el usuario ingrese letras. Del mismo modo, si esperamos una dirección de correo electrónico, se verifica que la escrita por el usuario sea correcta.

Figura 12. Vista de los teclados para ingresar teléfonos en dispositivos móviles.

Números (number)

También disponemos de un input para que el usuario escriba solo números, impidiendo, así, ingresar otro tipo de valor.

```
<label for="numero">Escriba su número de la suerte:<input type="number"
    name="numero"/>
```

Dirección Web (URL)

Este tipo de input sirve para comprobar que el texto ingresado sea una dirección web válida. El contenido deberá comenzar con **http://** y poseer, como mínimo, un **punto** (.). Si bien no se trata de una URL correcta el uso de **http://.** solamente, por lo menos llevamos al usuario a escribir una dirección web. Antes, esta verificación solo era posible mediante JavaScript.

```
<label for="txtWeb">Ingrese su sitio web:</label>
<input type="url" name="txtWeb"/>
```

Al poner un campo del tipo URL en los diferentes dispositivos móviles, también se adaptarán los teclados. Por ejemplo, se añadirán teclas como el **punto** (**.**), la barra o **slash** (**/**) y la tecla **.com**, entre otras que aparecerán.

Trabajar con fechas

Para trabajar con fechas, HTML5 nos trae varios tipos de inputs, que describiremos en la siguiente tabla:

CAMPOS DE FECHA

TIPO	DESCRIPCIÓN
Date	Nos mostrará un seleccionador de fechas y podremos elegir una fecha.
Time	Seleccionador de horas.
Datetime y datetime-local	Mix de campos de los tipos date y time.
Week	Selecciona una semana y un año.
Month	Selecciona un mes y un año.

Tabla 4. Nuevos campos para el trabajo con fechas.

Los inputs del tipo **date** nos enviarán la fecha en formato compatible, según lo definido por el **Request For Comments (RFC) 3339**, una serie de documentos que recopilan datos y estandarizan el uso de internet. En este caso, el RFC 3339 regula el formato de fechas en la Web:

REQUESTS FOR COMMENTS

Para conocer y ahondar más en los **Requests For Comments**, es recomendable visitar la página de la **Internet Engineering Task Force (IETF)**, el organismo que se encarga de estas publicaciones: **www.ietf.org/rfc.html**. También hay un buscador de RFC en: **www.rfc-editor.org/search/rfc_search.php**.

```
<form>
<label for="fechaNacimiento">Ingrese su fecha de nacimiento</label><input
        type="date" name="fechaNacimiento"/>
<label for="horaDespertador">Seleccione la hora que desea que suene su
        despertador</label><input type="time" name="horaDespertador"/>
<label for="fechaHora">Seleccione la fecha y la hora que desea tomar su vuelo</
        label><input type="datetime" />
<label for="semanaVacaciones">Indique la semana que desea tomarse
        vacaciones</label><input type="week" name="semanaVacaciones"/>
<label for="mesIngreso">Indique su mes y año de ingreso</label><input
        type="month" name="mesIngreso"/>
<input type="submit" value="Enviar"/>
</form>
```

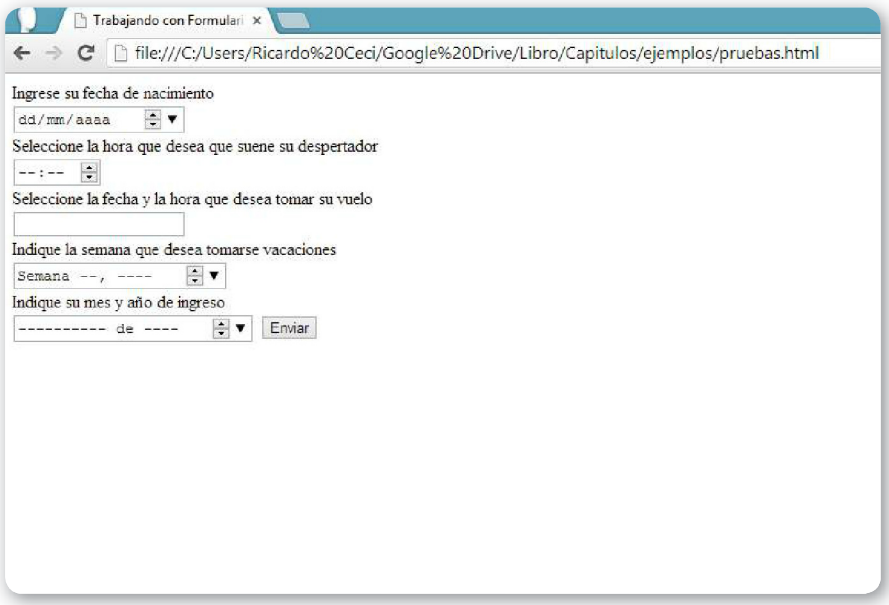


Figura 13. Inputs para fechas.

Nótese que la vista de nuestro formulario cambió. Ahora, veamos cómo se comportan los controles. Por ejemplo, el input del tipo **date** nos mostrará un calendario para seleccionar una fecha, como veremos en la **Figura 14**:

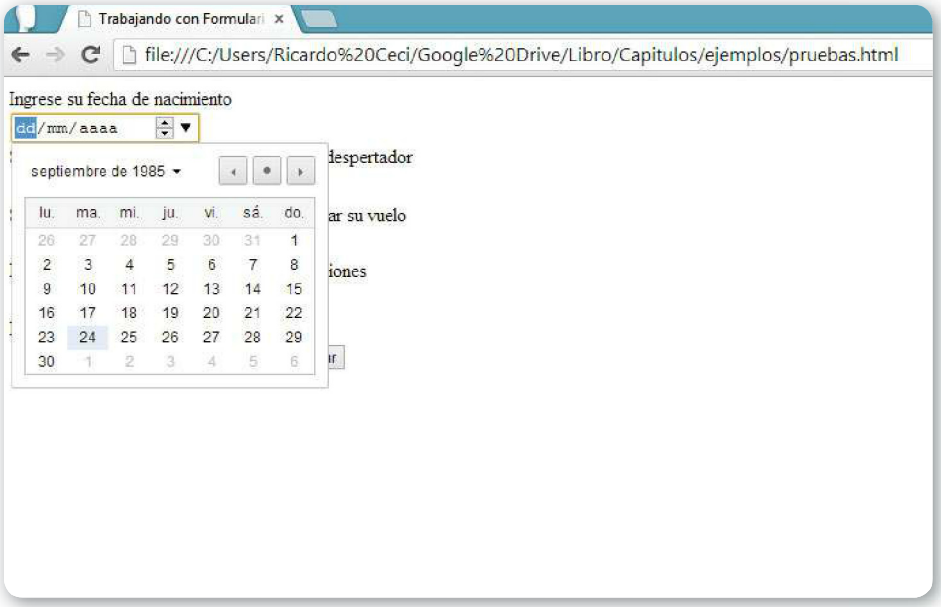


Figura 14. Así se comporta un input del tipo **`date`** en el navegador.

Veamos ahora cómo se visualiza en un dispositivo móvil un campo del tipo **datetime**:

Figura 15. Los dispositivos móviles muestran sus **selectores de fechas** (**`datepickers`**) con el formato de su sistema operativo, y no es necesario escribir un código propio para cada uno.

Otros tipos de inputs

HTML5 también nos trae dos inputs especiales para mejorar la experiencia de usuario. Al momento de escribir esta obra, estos campos no se encuentran implementados en la mayoría de los navegadores.

- **range**: dibuja una barra que se puede desplazar para poder seleccionar un valor.

```
<label for="nivelVolumen">Seleccione el volumen deseado</label><input
        type="range" name="nivelVolumen">
```

- **color**: muestra un campo para seleccionar un color desde una paleta de colores. Funciona sólo en Chrome, Blackberry Browser (7.0+), Opera Mobile y Chrome for Android (29+).

```
<label for="selColor">Seleccione su color favorito</label>
<input type="color" name="selColor">
```

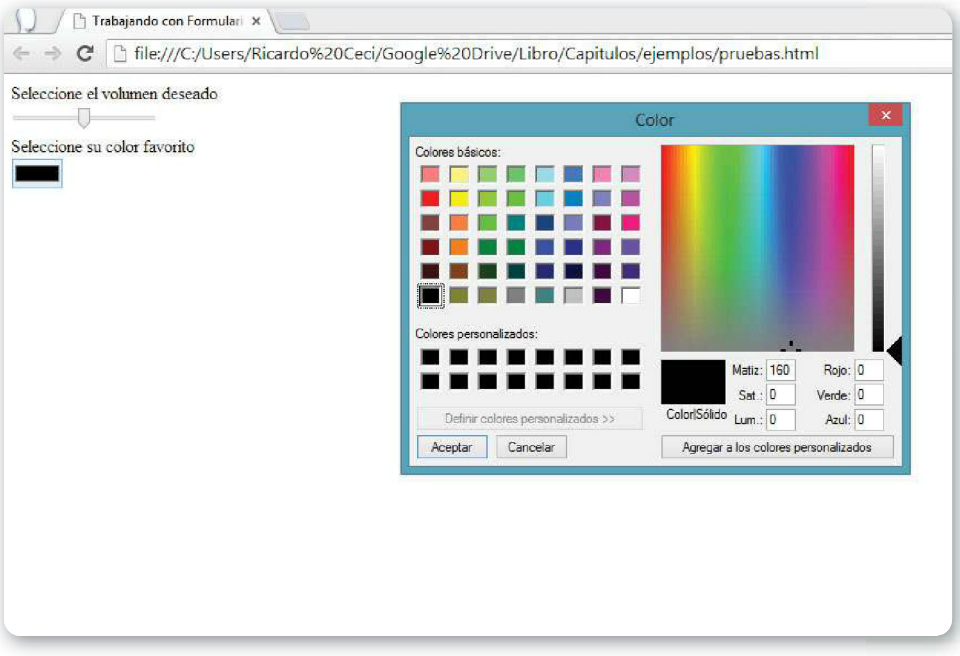


Figura 16. Inputs para rangos y colores. Si bien son muy útiles, solo están disponibles en algunos navegadores.

Nuevos atributos

Además de los nuevos tipos de inputs, HTML5 ofrece nuevos atributos que permitirán personalizar aún más los controles. A continuación, los conoceremos.

Placeholder

Es un atributo que permite indicar cuál será el texto que aparecerá escrito en el campo ni bien se muestre. Cuando el usuario haga foco (toque con el mouse, por ejemplo) en el campo, este texto desaparecerá. Al momento de perder el foco, si el usuario escribió algo en el campo, este texto no se verá más; en cambio, si lo dejó vacío, volverá a mostrarse.

```
<input type="email" placeholder="Ingrese su email"/>
```

Min, Max y Step

Estos atributos nos permitirán definir un número mínimo, un número máximo y un número de incremento o decremento que sufrirá el valor si presionamos algún control para aumentar o disminuirlo.

```
<label for="numeroElegido">Ingrese un número entre 0 y 100</label>
<input type="number" min=0 max=100 name="numeroElegido"/>
<label for="cada5">Ingrese un número entre 0 y 50</label>
<input type="number" min=0 max=50 step=5/>
```

Estos atributos también sirven para fechas. Por ejemplo, le pediremos al usuario que ingrese su fecha de nacimiento e indicaremos como límite máximo el 30 de septiembre de 1995:

```
<label for="fechaNacimiento">Indique su fecha de nacimiento</label>
<input type="date" max="1995-09-30"/>
```

De este modo, cuando ejecutemos este código, no se permitirá seleccionar fechas posteriores al 30 de septiembre de 1995.

List y Datalist

A través del atributo **list** en un input podemos definir una lista de valores de la cual el usuario deberá seleccionar uno. Mientras el usuario vaya tipeando, se mostrarán los valores posibles a elegir debajo del campo.

Las listas deben ser definidas dentro de una etiqueta **<datalist>.** Veamos un ejemplo:

```
<form>
<label for="pais">Ingrese un país del MERCOSUR</label><input type="text"
name="pais" list="mercosur">
</form>
<datalist id="mercosur">
        <option>Argentina</option>
        <option>Brasil</option>
        <option>Paraguay</option>
        <option>Uruguay</option>
        <option>Venezuela</option>
</datalist>
```

Si ejecutamos este código en nuestro navegador, veremos que la lista no se mostrará en pantalla. Pero cuando empezamos a tipear en el campo, el navegador nos sugerirá completarlo con alguno de los valores de la lista: si tipeamos la letra "A" nos recomendará completarlo con Argentina; si tipeamos la "U", Uruguay; etcétera.

Atributos booleanos

A estos atributos se los llama **booleanos** porque solo aceptan los valores **verdadero** y **falso**. Si no se especifica ningún valor, significa que son verdaderos. El único valor que hay que indicarles es el **falso** o, en algunos casos, no escribirlos.

Autofocus

Este atributo permitirá que, al momento de cargarse el documento, el input que tenga este atributo reciba el foco automáticamente.

```
<form>
<label for="usuario">Ingrese su Usuario:</label>
<input type="text" name="usuario" autofocus />
<label for="clave">Ingrese su Contraseña:</label>
<input type="password" name="clave" />
<input type="submit" value="Enviar datos"/>
</form>
```

Solamente un campo de todos los que componen el formulario deberá tener este atributo para que el comportamiento sea el esperado. Si hubiera varios campos con el atributo **autofocus**, carecería de sentido la necesidad de posicionar el puntero del mouse en un lugar al cargar la página.

Required

Cuando a un input le escribimos este atributo, no se permite el envío del formulario si no se tiene ningún valor ingresado o, en caso de ser un **select**, si no se ha seleccionado un valor.

```
<input type="text" name="usuario" required>
```

Autocomplete

Si le especificamos el valor **autocomplete**, el navegador sugerirá al usuario completar el campo con los valores que él cargó previamente. En cambio, si además de especificarlo le asignamos el valor **off**, impedirá este comportamiento por defecto del navegador. Veamos un ejemplo:

```
<form>
<label for="txtEmail">Ingrese su email (con autocomplete)</label><input
        type="email" name="txtEmail" autocomplete required />
<label for="txtEmail">Ingrese su email (autocomplete off)</label><input
        type="email" name="txtEmail" autocomplete="off" required />
<input type="submit" value="Enviar" />
</form>
```

Novalidate

Especificando este atributo a la etiqueta **form**, el navegador no procesará ni realizará las validaciones por defecto del navegador en los campos. Es decir, no se validará que sean direcciones de e-mail o URL válidas, que los campos requeridos estén completados, etcétera.

```
<form novalidate>
<label for="txtEmail">Ingrese su email (con autocomplete)</label><input
      type="email" name="txtEmail" autocomplete required />
<label for="txtEmail">Ingrese su email (autocomplete off)</label><input
type="email" name="txtEmail" autocomplete="off" required />
<input type="submit" value="Enviar" />
</form>
```

Multiple

Este atributo permite que el usuario ingrese múltiples valores en un mismo input, pero funciona sólo en inputs del tipo **email** y **file**. Para los inputs del tipo **file**, cuando el usuario vea el cuadro de diálogo para seleccionar el archivo, podrá seleccionar más de uno, presionando la tecla **CTRL** y haciendo clic con el mouse en cada ítem. Para ingresar varias direcciones de e-mail, habrá que escribirlas separadas con una **coma** (**,**). Este atributo es útil ya que nos evita tener que cargar múltiples inputs para recibir múltiples datos homogéneos.

```
<form>
<label for="archivos">Seleccione los archivos a subir</label>
<input type="file" name="archivos" multiple>

<label for="emails">Ingrese sus direcciones de email separadas por ,(coma)</
      label>
<input type="email" name="emails" multiple required>
</form>
```

Como vemos, **los atributos se pueden combinar**. Simplemente hay que separarlos con un espacio entre uno y otro.

Realizar validaciones

Si queremos dar un paso más al momento de validar formularios, podemos utilizar expresiones regulares para poder validar lo que el usuario ingresa. Ahora bien, ¿qué es una expresión regular? Una **expresión regular** (**regex**) es una secuencia de caracteres que, combinados entre sí, permiten generar un patrón contra el cual se compararán los datos ingresados por el usuario.

En la siguiente tabla se muestra un ejemplo sencillo de cómo funcionan las expresiones regulares, comparando una cadena de texto con un patrón definido.

Cadena de texto de ejemplo: **Libro HTML5 de RedUSERS**

EXPRESIONES REGULARES

EXPRESIÓN (PATRÓN)	TIPO	DESCRIPCIÓN
M	Encuentra la M en HTML5.	Válido (cumple).
m	No hay m minúscula.	No válido (no cumple).
[a-zA-z]	Busca cualquier carácter de la a a la z y de la A a la Z.	Válido.
[0-9]	Busca cualquier número entre 0 y 9.	Válido (HTML5).

Tabla 5. Ejemplos de patrones de validación y resultados.

Para poder utilizar las expresiones regulares en nuestros campos de formularios, usaremos el atributo **pattern**, que recibirá como valor una expresión regular con la cual se comparará lo que escribió el usuario. Devolverá la descripción "válido" si lo escrito coincide con el patrón o "inválido" en caso contrario.

Validar números de tarjetas de crédito

Si miramos varias tarjetas de crédito, veremos que cumplen determinados patrones. Por ejemplo, una tarjeta American Express tiene un bloque de cuatro dígitos de 0 a 9; luego, otro bloque de seis dígitos de 0 a 9; y, por último, otro bloque de cinco dígitos de 0 a 9.

La expresión regular para describir este patrón sería: **[0-9]{4} *[0-9]{6} *[0-9]{5}**. Veamos cómo lo aplicamos en nuestro formulario:

```
<form>
<label for="tcAmex">Ingrese un número de tarjeta American Express</label>
<input type="text" required pattern="[0-9]{4} *[0-9]{6} *[0-9]{5}"/>
<input type="submit" value="Enviar"/>
</form>
```

Dar estilo a nuestros formularios

CSS3 ofrece una serie de pseudoclases para realizar validaciones de formularios. Al utilizarlas, podremos comprobar visualmente y en el momento si el campo pasó o no las validaciones. Veamos algunas:

- **:required**: esta pseudoclase se activará si el campo es requerido. En el siguiente ejemplo, pintaremos de amarillo los campos obligatorios.

```
<!doctype html>
<html>
<head>
<title>Trabajando con Formularios</title>
<style>
        label {
                display:block;
        }
        :required{
        background-color:yellow;
}
</style>
</head>
<body>
<form>
<label for="txtNombre">Nombre:</label><input type="text" name="txtNombre"
                required/>
        <label for="txtApellido">Apellido:</label><input type="text"
```

```
                name="txtApellido" required/>
        <label for="txtEmail">Email:</label><input type="email"
                name="txtEmail"/>
        <label for="clave">Contraseña:</label><input type="password"
                name="clave" required/>
        <label for="foto">Seleccione una foto:</label><input type="file"
                name="foto" >         /
        <label for="sexo">Sexo:</label><input type="radio" name="sexo"
                value="mujer" />Mujer<input type="radio" name="sexo"
                value="hombre"   >Hombre<br/>
        Acepto los términos y condiciones <input type="checkbox"
                name="aceptaTyC" value="si" /><br/>
        <input type="hidden" name="campoOculto" value="Esto no lo ve el
                usuario"            />
        <input type="submit" value="Enviar"><input type="reset" value="Limpiar
                campos"/>
        </form>
</body>
</html>
```

También podemos unirlo con selectores clásicos. Por ejemplo, si queremos que los inputs que sean requeridos y que tengan foco sean pintados de naranja, deberíamos utilizar el selector **input:required:focus.**

```
<style>
        input:required:focus {
        background-color:orange;
}
</style>
```

- **:valid**: esta pseudo clase se cumplirá cuando lo ingresado en el campo sea válido, es decir, cuando coincida con el patrón solicitado: una dirección de e-mail, una URL o algún patrón predefinido. Su opuesto es **:invalid.**

Conociendo estas dos pseudoclases, podemos cambiar el estilo de un input de inválido a válido en el momento y apreciar la validación instantánea que hace el navegador.

```
<style>
        :valid {
                background-color:green;
}
:invalid {
        background-color:red;
}
</style>
<body>
        <form>
                <input type="email" name="email" required/>
        </form>
```

Cuando comenzamos a tipear el e-mail, el campo indica que es inválido. Cuando escribimos el patrón correcto, se torna verde.

De esta manera, HTML5 nos ayuda a hacer más potentes nuestros formularios, evitándonos el desarrollo de múltiples scripts para validar campos que comúnmente son utilizados en los formularios y, a la vez, mejorando la experiencia del usuario al momento de completarlos.

RESUMEN

En este capítulo hemos visto que HTML5 trae una amplia variedad de campos y atributos para utilizar formularios. El trabajo del programador web se ve potenciado con el uso de estos nuevos elementos. Con pequeñas modificaciones a nuestros formularios ya desarrollados o a los nuevos que vayamos a desarrollar, podremos mejorarlos notablemente y realizar validaciones que antes requerían de mucha programación en JavaScript para lograr los mismos resultados. Ahora, estas tareas de validación quedan delegadas al navegador. Si bien estos nuevos elementos y atributos tienen bastante compatibilidad, tendremos que probarlos en múltiples plataformas y dispositivos, pues se encuentran en constante evolución. Todos los tipos de inputs que no sean compatibles con el navegador se mostrarán automáticamente como de tipo texto.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione dos tipos de inputs de formularios.
2. ¿Qué tipo de inputs tenemos que utilizar para ingresar contraseñas?
3. ¿Qué pasos tenemos que realizar para que, de dos radio buttons, se pueda seleccionar uno o el otro y no ambos?
4. ¿Qué ocurre si un input nuevo de HTML5 no es compatible con el navegador donde se está ejecutando?
5. Mencione dos atributos nuevos para formularios en HTML5.
6. ¿Qué es una **RFC**?
7. ¿Qué es una **expresión regular**?
8. Si quiero indicar que un campo no ha pasado la validación con CSS, ¿qué pseudoclase tengo que utilizar?

EJERCICIOS PRÁCTICOS

1. Desarrolle un formulario de contacto básico para un sitio web con los siguientes campos: Nombre, Apellido, E-mail y Comentarios.
2. Realice un formulario de inscripción para un club utilizando los nuevos inputs y atributos de HTML5.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*07

Selectores avanzados

En este capítulo revisaremos los selectores que pertenecen a las distintas versiones de CSS y veremos en detalle los selectores llamados, comúnmente, “avanzados”. También aprenderemos las pseudoclases y pseudoelementos que se incorporan en CSS3, y para qué podemos usar cada uno de ellos.

- **La magia de los selectores 208**
  - Selectores básicos..... 208
  - Selectores avanzados 218
- **Pseudoclases y pseudoelementos 227**
  - Pseudoclases y pseudoelementos de CSS3 229
- **Resumen.....237**
- **Actividades.....238**

La magia de los selectores

Generalmente nos preocupamos por conocer en detalle todas y cada una de las propiedades de CSS y creemos que así tendremos un dominio absoluto del lenguaje. Si bien es verdad que es muy importante conocer las distintas propiedades de CSS y las posibilidades que brindan, la sabiduría real se encuentra en tener un profundo conocimiento de los selectores, ya que esto nos permitirá crear archivos de estilo mejor estructurados, optimizar la cantidad de líneas de código de nuestras hojas de estilo y mantener nuestros archivos HTML más limpios.

Entonces, ¿queda claro que saber CSS no implica conocer las propiedades de memoria, sino dominar los selectores? Comencemos por revisar los selectores que ya conocemos.

Selectores básicos

Así llamamos al grupo de selectores de uso más frecuente. No hay un selector más importante que otro, sino que cada uno de ellos posee una función completamente distinta y nos será más o menos útil dependiendo del proyecto que debamos llevar a cabo.

Selector universal

Se utiliza para seleccionar todos los elementos del documento y aplicar las propiedades que necesitemos. Veamos un ejemplo:

```
* {
      Color:red;
}
```

En este caso, se aplicará color de texto rojo a todos los elementos del documento HTML. El selector universal generalmente se utiliza para resetear los márgenes y los rellenos de las etiquetas, que suelen ser distintos para cada navegador. Comúnmente, agregaremos al inicio de nuestros archivos de estilo la siguiente regla de estilo:

```
* {
  margin: 0;
  padding: 0;
}
```

Más allá de este ejemplo, es poco probable que sea usado, ya que raramente necesitaremos aplicar las mismas propiedades a todos los elementos del documento.

Sí es común que lo combinemos con otros selectores, o bien que lo usemos para aplicar algún tipo de hack.

Es importante resaltar que este selector aplica las propiedades de CSS a cada elemento de manera independiente y no trabaja por herencia, como veremos en otros selectores.

Selector de etiquetas

También llamado **selector de tipo** o **selector de elementos**, es un tipo de selector que siempre usaremos en los archivos de estilo de nuestros proyectos. Se utiliza para elegir un elemento dado y aplicar propiedades a todos los elementos del mismo tipo.

Este selector utiliza el nombre de la etiqueta sin los signos de menor y mayor (**<** y **>**) para seleccionar dicho elemento. Por ejemplo:

```
h2 {color:orange;}
```

De esta forma, todas las etiquetas **<h2>** de nuestro proyecto tendrán color de texto naranja.

¿QUÉ SON LOS HACKS?

Se denomina **hacks** a los pequeños trucos que podemos incluir en los archivos CSS para que distintos navegadores tomen diferentes valores de una misma regla de estilos. Esto nos resultará muy útil cuando usemos propiedades que no soporten todos los navegadores. Son muy utilizados para aplicar correcciones para las distintas versiones de Internet Explorer.

Selector de clase

Se trata de uno de los selectores más usados por los desarrolladores, ya que permite seleccionar un elemento que posee un atributo **class.** El signo que utilizamos para identificar a este tipo de selector es el **punto** (**.**). Veamos un ejemplo:

```
.importante {color:red;}
```

En el archivo HTML tendremos el siguiente código:

```
<h2 class="importante">Título con clase aplicada</h2>
<h2> Título sin clase aplicada</h2>
```

Este selector es útil cuando queremos aplicar un estilo a un elemento específico y sobre todo cuando queremos reutilizar ese estilo.

Pensemos en el siguiente caso: definimos una regla de estilo para que todos nuestros títulos **<h2>** tengan un color determinado, pero luego necesitamos que uno de nuestros títulos tenga características distintas al resto. En ese escenario, podemos agregar una o más clases a este elemento para lograr el estilo visual requerido.

Uno de los aspectos más importantes de los selectores de clases es que se pueden utilizar tantas veces como se necesiten y aplicar a distintos elementos de HTML. Esto significa que la misma clase puede aplicarse tanto a un párrafo como a un título, una imagen o cualquier otra etiqueta que necesitemos. Es bastante común que muchos desarrolladores utilicen excesivamente las clases. Suele ser más interesante utilizar –siempre que sea posible– otros selectores que permitan aplicar un estilo sin tener que agregar en el archivo HTML un atributo específico como **class.**

Selector de ID

Se trata de otro tipo de selector que usamos con mucha frecuencia, a veces incluso sin saber del todo por qué lo usamos. Se parece mucho al selector de clase, aunque es más específico y debe ser usado con mayor precaución. Al igual que el selector de clase, el selector de ID utiliza como referencia un atributo del HTML. Sirve para asignar un nombre

de identificación único a un elemento del documento HTML y utiliza el signo **numeral** (#) como elemento identificador. Veamos un ejemplo:

```
#columna {color:#666666;}
```

En el archivo HTML tendremos el siguiente código:

```
<article id="columna"> Texto del título </article>
```

La diferencia esencial entre el selector de ID y el selector de clase radica en la naturaleza misma del atributo ID, el cual no debe ser usado más de una vez en cada documento con el mismo valor. En cambio, el atributo clase sí puede recibir el mismo valor tantas veces como sea necesario. En un mismo archivo puede haber muchos elementos que usen las misma clase, pero **nunca debería haber más de un elemento con el mismo ID**. Este ejemplo no es correcto:

```
<section>
        <article id="columna"> Texto del título </article>
        <article id="columna"> Texto del título </article>
</section>
```

En cambio, este ejemplo sí es correcto:

```
<section>
        <article class="columna"> Texto del título </article>
        <article class="columna"> Texto del título </article>
</section>
```

En resumen, usamos selectores de ID solo cuando el elemento posee un estilo único que no se repetirá, como es el caso del bloque de la navegación principal, el pie de página, el encabezado principal, etcétera. Por el contrario, usamos selectores de clase cuando queremos aplicar dicho estilo a más de un elemento de igual o distinto tipo.

Un último punto importante para tener en cuenta sobre el selector de ID es que posee un valor de jerarquía mayor al selector de clase y al selector de etiqueta.

Restricción del alcance de los selectores

Si bien no se utiliza con frecuencia, el alcance de los selectores de clase y de ID puede restringirse si les aplicamos el contexto en donde pueden ser usados. Veamos un ejemplo:

```
p#columna { color: green; }

h2.importante { color: red; }
```

En este caso, estamos definiendo que el color de texto verde se aplique únicamente al elemento **<p>** que contenga el ID cuyo valor sea igual a **columna** y que el color de texto rojo se aplique solo a los elementos que contengan el atributo clase con valor igual a **importante**, pero estos deben ser sí o sí etiquetas **<h2>.**

Selector de grupo

Si prestamos un poco de atención a nuestas hojas de estilo, veremos que en muchos casos creamos reglas de estilo prácticamente iguales. Por ello, existe un selector que permite agrupar reglas de estilo para aplicar en forma conjunta las propiedades de CSS a más de un elemento. El selector de grupo utiliza la **coma** (**,**) para sumar los

HERENCIA

Este concepto es la base del funcionamiento de CSS y funciona igual que en la vida: cuando definimos un estilo, no lo hacemos solo para ese elemento, sino también para sus elementos hijos, nietos, etcétera. Si definimos una regla de estilo que indique que los párrafos tienen color de texto gris, todos los elementos hijos de esos párrafos también tendrán color de texto gris, a menos que en la hoja de estilo exista una regla de estilo más específica para el elemento hijo.

selectores a los que les aplicará las propiedades de CSS. Veamos un ejemplo de uso:

```
p , a {margin-bottom:20px;}
```

Según este ejemplo, se le aplicará un margen inferior de 20 píxeles a los elementos **<a>** y a los elementos **<p>**. Veamos un caso típico de uso posible en nuestro archivo de estilo:

```
h1 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;}

h2 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;}

h3 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;}
```

En este caso, podemos optimizar el código CSS de la siguiente forma, agrupando todas las reglas de estilo individuales en una sola para obtener el mismo resultado con menos líneas de código:

```
h1 , h2 , h3 {color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;}
```

El uso de selectores de grupo es una práctica bastante frecuente, sobre todo en archivos de estilo complejos, ya que nos permite reducir

la cantidad de líneas de código y crear reglas de estilo más eficientes que permitirán efectuar modificaciones con menos esfuerzo.

Seguramente se preguntarán qué ocurre si las reglas de estilo que queremos agrupar no son exactamente iguales. Este escenario es muy frecuente, por lo que también podemos usar el selector de grupo para optimizar el archivo de estilo. Veamos un ejemplo con reglas de estilo que no son exactamente iguales:

```
h1 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;
        font-size:45px;}

h2 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;
        font-size:35px;}

h3 { color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;
        font-size:25px;
        text-align:center;}
```

En este caso, podemos ver que, si bien la mayoría de las propiedades se repiten, el tamaño de texto de cada una es diferente y, además, la última regla de estilo posee una propiedad que no está en las anteriores. Podemos agruparlas de la siguiente manera:

```
h1 , h2 , h3 {color: #333333;
        font-style: italic;
        font-family: Arial, Helvetica, sans-serif;
        margin-bottom:20px;
        font-size:45px;}
```

```
h2 {   font-size:35px;}

h3 {   font-size:25px;
       text-align:center;}
```

En este ejemplo, podemos notar que seguimos usando el selector de grupo para unificar en una misma regla de estilo todas las propiedades que comparten los distintos elementos, y luego definimos individualmente las reglas de estilo solo con las propiedades específicas para cada elemento.

Hay que tener en cuenta que, en estos casos, el orden de las reglas de estilo es importante, ya que **las reglas de estilo que están más abajo sobrescriben a aquellas que están más arriba** en el archivo de estilo.

En el ejemplo que estamos trabajando, el selector de grupo le aplica tamaño de texto **45px** a las etiquetas **<h1>**, **<h2>** y **<h3>**, pero este valor está sobrescrito más abajo para las etiquetas **<h2>** y **<h3>**, respectivamente.

Selector descendente

Es uno de los selectores más usados de CSS, no por capricho o moda, sino porque resulta de gran utilidad a la hora de realizar la maquetación de un proyecto de mediana o gran envergadura. Este selector nos permite seleccionar un elemento que se encuentra anidado dentro de otro.

La sintaxis que se utiliza para este selector usa un espacio entre un elemento y otro, lo cual indica la descendencia entre los elementos. Veamos un ejemplo:

```
p strong {color: red; }
```

En este caso, se aplicará color rojo a toda etiqueta **** que cumpla con la condición de ser un descendente de **<p>**, es decir, que esté anidado dentro de él u otro descendente de **<p>.**

El código HTML sería:

```
<p>
 Texto
 <strong>texto</strong>
 texto
 <a href=""> este es un <strong>link</strong></a>
 Último texto.
</p>
```

Es importante entender que el color de texto rojo aplicará al texto contenido de las etiquetas **** que están dentro de la etiqueta **<p>**, así como también a las que están contenidas en la etiqueta **<a>**, ya que todas son descendientes de **<p>.**

Esto se debe a que este selector no requiere que sea un descendente directo, sino que **aplica a cualquier elemento que sea descendente de él**, sin importar el nivel de profundidad. El selector descendente puede estar compuesto por varios niveles para aumentar su precisión, como por ejemplo:

```
p a strong em { color: pink; }
```

En este caso, el color de texto se aplicará a los contenidos de las etiquetas **** que cumplan con la condición de ser descendentes de ****, que además debe ser descendente de **<a>**, que a su vez debe ser descendente de **<p>.**

Es muy importante entender la diferencia entre un selector descendente y el selector de grupo, ya que su sintaxis es muy similar pero su función es totalmente distinta. Ejemplos:

```
p a strong em { color: pink; }
```

En este caso, el color rosa solo se aplicará a los contenidos de las etiquetas **.**

```
p,a,strong,em { color: pink; }
```

En este caso, el color será aplicado a los elementos **<p>**, **<a>**, **** y ****.

Selector descendente y selector universal

También se puede combinar el selector universal con el selector descendente, con el fin de restringir su alcance. Supongamos que tenemos el siguiente código HTML:

```
<p><a href="#">Link</a></p>
<p><strong><a href="#">Link</a></strong></p>
```

y creamos la siguiente regla en el archivo de estilo:

```
p a {color: red;}
```

Está claro que el texto será rojo para ambos links. Pero si creamos la siguiente regla de estilo:

```
p * a {color: red;}
```

solo el segundo link se verá de color rojo, ya que el primero no cumple con la condición requerida por este selector.

JERARQUÍA Y ESPECIFICIDAD

El concepto de jerarquía es uno de los más importantes de entender en CSS, ya que cuando una regla de estilo no funciona se debe generalmente a un conflicto de jerarquía entre dos o más reglas de estilo que le dan instrucciones contradictorias a un mismo elemento. En estos casos, se aplicará la regla cuyo selector tenga mayor jerarquía. Recordemos que el selector de ID tiene mayor jerarquía que los selectores de clase y de etiqueta y, a su vez, el selector de clase posee más jerarquía que el selector de etiqueta.

El selector **p * a** afecta a todos los elementos **<a>** que se encuentren dentro de cualquier elemento que, a su vez, esté contenido dentro de un elemento de tipo **<p>**. Dado que el primer elemento **<a>** se encuentra directamente bajo un elemento **<p>**, no se cumple la condición requerida por el selector **p * a.**

Selectores avanzados

Los selectores avanzados permiten simplificar las hojas de estilo y seleccionar elementos del documento sin necesidad de agregar atributos adicionales en el documento HTML. No todos estos selectores pertenecen a la especificación de CSS3; por lo tanto, algunos de ellos no son soportados por todos los navegadores, en especial por ciertas versiones de Internet Explorer.

Un punto muy importante a considerar al respecto es que casi todos están atados al flujo del documento HTML. Esto significa que, al cambiar el orden de los elementos o el nivel de anidamiento, nuestras reglas de estilo dejarán de aplicarse, ya que el contexto del HTML se habrá modificado.

Selector hijo

Se parece mucho al selector descendente, ya que permite seleccionar un elemento que es hijo de otro. Solo que, en este caso, la descendencia debe ser directa. Este selector utiliza el signo **mayor que** (**>**) para indicar cuál es el elemento padre y cuál el elemento hijo. Veamos un ejemplo:

```
p > em { color: red; }
```

En este caso, el contenido de las etiquetas **** será de color rojo únicamente cuando la etiqueta **** sea hija directa de la etiqueta **<p>.** En el código HTML tendríamos:

```
<p> Texto de <em> párrafo </em><p>
<p> Texto del <a href="#"> <em> segundo </em> </a> </p>
```

En este caso, la palabra **párrafo** quedará de color rojo, ya que se cumple la condición solicitada de que **** sea hija directa de **<p>**; pero la regla se estilo no afectará a la palabra **segundo**, dado que en ese caso no se cumple la condición exigida, pues la etiqueta **** es hija directa de la etiqueta **<a>.**

Este selector es particularmente útil cuando creamos menús desplegables, ya que permite seleccionar un elemento hijo sin afectar a los otros niveles de descendencia. Consideremos que tenemos el siguiente código HTML:

```
<ul>
        <li> Empresa </li>
        <li> Servicios
                <ul class="submenu">
                        <li> Sevicio 1 </li>
                        <li> Servivio 1
                                <ul class="submenu">
                                        <li> Sub Sevicio 1 </li>
                                        <li> Sub Servivio 1 </li>
                                </ul>
                        </li>
                </ul>
        </li>
        <li> Clientes </li>
        <li> Contacto </li>
</ul>
```

Esto se verá en el navegador de la siguiente forma:

IMÁGENES EN CANVAS

En la etiqueta **<canvas>** podemos insertar todo tipo de elementos, pero no mediante la etiqueta **<img>** como lo haríamos en un documento HTML, ya que todo lo que vamos a insertar en la etiqueta **<canvas>** debe ser manejado mediante JavaScript, utilizando los métodos y propiedades correspondientes. Los fornatos de imágenes soportados son los mismos que usamos en un documento HTML, como PNG, JPG y GIF.

Figura 1. Menú desplegado antes de aplicar los estilos.

Podemos crear, entonces, las reglas de estilo que controlen la visibilidad de las listas hijas sin afectar a las listas nietas. Con esta regla, ocultamos las listas hijas:

```
.submenu {display:none;}
```

Con esta regla, mostramos las listas hijas:

```
li:hover>ul {display:block}
```

Selector hermano adyacente

Se trata de un selector que permite elegir un elemento que debe cumplir con dos condiciones: ser hermano –es decir, debe ser hijo de un mismo padre– y, por otro lado, ser adyacente del elemento que estamos utilizando como contexto. La sintaxis de este selector utiliza un **signo de sumar** (**+**) para marcar la relación de los elementos. Veamos un ejemplo:

```
h2 + p {color:red;}
```

Esta regla de estilo se aplicará al elemento **<p>** que cumpla con la condición de ser hermano adyacente del elemento **<h2>**. Supongamos que tenemos el siguiente código HTML:

```
<h2> Titulo </h2>
<p> Texto </p>
<p> Texto </p>
<p> Texto </p>
<p> Texto </p>
```

En este caso, el color de texto rojo se aplicará únicamente al párrafo que se ubica inmediatamente después de la etiqueta **<h2>**, dado que es el único que cumple con las condiciones requeridas por este tipo de selector.


EL CIRCUITO ATP

Andy Murray descansa con su novia en Bahamas, tras ganar Wimbledon

Luego de ser el primer británico en 77 años en ganar el certamen, el campeón se relaja.

A los 26 años, Andy Murray dio uno de los golpes más grandes al ser el primer británico en ganar Wimbledon, luego de 77 años. La gesta la logró hace 9 días en Londres y por estas horas el campeón disfruta y descansa en Bahamas junto a su novia, Kim Sears. Sin tenis, el hombre se divierte con la paleta.

La pareja descansa en el caribe

A los 26 años, Andy Murray dio uno de los golpes más grandes al ser el primer británico en ganar Wimbledon, luego de 77 años. La gesta la logró hace 9 días en Londres y por estas horas el campeón disfruta y descansa en Bahamas junto a su novia, Kim Sears. Sin tenis, el hombre se divierte.

A los 26 años, Andy Murray dio uno de los golpes más grandes al ser el primer británico en ganar Wimbledon, luego de 77 años. La gesta la logró hace 9 días en Londres y por estas horas el campeón disfruta y descansa en Bahamas junto a su novia, Kim Sears. Sin tenis, el hombre se divierte con la paleta.

Figura 2. Usando el selector de hermano adyacente podemos afectar a un elemento por su contexto.

Selector hermano

Este nuevo selector fue incorporado en la especificación de CSS3 y, en principio, podríamos decir que es muy parecido al selector de hermano adyacente explicado anteriormente. Solo que tiene una leve diferencia: aplica a todos los elementos hermanos que existan después de él, sin necesidad de que estos sean estrictamente adyacentes a él.

La sintaxis de este selector utiliza el signo **virgulilla** (~) para indicar la relación de los elementos que componen el selector, como podemos ver en el ejemplo:

```
h2 ~ p {color:red;}
```

Esta regla de estilo se aplicará a todos los elementos **<p>** que cumplan la condición de ser hermanos del elemento **<h2>** y que se encuentren luego de él en el flujo del HTML.

Supongamos que tenemos el siguiente código HTML:

```
<h1> Titulo principal </h1>
<p> Texto negro </p>
<h2> Titulo </h2>
<p> Texto rojo </p>
<p> Texto rojo </p>
<h3> Titulo 2 </h3>
<p> Texto rojo </p>
<p> Texto rojo </p>
```

En este caso, todos los párrafos cuyo contenido es "texto rojo" serán afectados por la regla de estilo que creamos, ya que cumplen con las dos condiciones necesarias para este selector: son hermanos y están luego de él en el flujo del documento HTML. No así el párrafo cuyo contenido es texto negro, que cumple solo una de las dos condiciones.

Figura 3. El selector hermano permite afectar a los elementos hermanos que se encuentran luego de él.

En conclusión, podríamos decir que ambos selectores se utilizan para seleccionar elementos hermanos, pero el selector hermano aplica a todos los elementos que cumplan esta condición, mientras que el selector hermano adyacente aplica solo a un elemento que, como su nombre lo indica, debe ser hermano y, además, adyacente.

Selector de atributo

Es uno de los selectores más interesantes de CSS, ya que permite seleccionar un elemento por alguno de los atributos que posee. En primera instancia, esto no sería muy diferente de lo que hacen el selector de clase o el de ID, pero la diferencia significativa está en que este selector puede utilizar cualquier atributo que el elemento contenga. Esto significa que podemos aprovechar esos atributos que existen en el HTML por efectos funcionales para asociarles una regla de estilo.

Asimismo, este selector presenta distintas variantes que permiten hacer selecciones más precisas o bien más genéricas, dependiendo de nuestras necesidades.

Si bien este selector corresponde a la versión CSS 2.1, algunas de sus variables fueron especificadas en la versión CSS3. La sintaxis de este selector utiliza **corchetes** (**[ ]**) para encerrar el atributo que va a seleccionar. Veamos un ejemplo de este selector:

```
[atributo] { propiedad:valor; }
```

En un caso de uso real, sería así:

```
[title] { color:red; }
```

Esta regla de estilo aplica a todos los elementos del HTML que posean el atributo **title**. Consideremos que tenemos el siguiente código HTML:

```
<h1 title="título principal"> Titulo 1 </h1>
<p title="párrafo principal"> Texto </p>
<p> Texto </p>
```

```
<p> Texto </p>
<p> Texto </p>
```

En el ejemplo anterior, se le aplicará el color rojo al contenido de la etiqueta **<h1>** y al primer párrafo, dado que los dos cumplen con la condición de poseer el atributo **title.**

Ahora bien, este selector puede resultar algo genérico; no nos apresuremos aún, pues posee muchas opciones para volverlo más específico y más adecuado a lo que necesitemos realizar.

Para empezar, podemos colocar delante de este selector cualquier otro selector para darle un contexto, por ejemplo:

```
article [title] { color:red; }
```

En este caso, la regla de estilo se aplicará a los elementos que poseen **title** y que son descendientes de **<article>**. También podemos enunciar el selector de esta forma:

```
section[title] { color:red; }
```

Este caso es totalmente distinto: acá estamos definiendo que la regla de estilo se aplique a los **<section>** que poseen el atributo **title.** Recordemos que los espacios, comas y otros signos indican distintos tipos de relaciones en un selector de CSS.

Otra posibilidad del selector de atributo es la de seleccionar un elemento que tenga un atributo y un valor dado.

Veamos este ejemplo:

```
p[title="facebook"] { color:red; }
```

En esta caso, la regla de estilo se aplica a las etiquetas **<p>** que posean el atributo **title** con un valor que sea exactamente igual a "facebook".

Observemos el siguiente documento HTML:

```
<p title="facebook"> Parrafo 1 </p>
<p title="seguinos en facebook"> Parrafo 2 </p>
<p> Parrafo 3 </p>
```

Esta regla de estilo sólo se aplicará al primer párrafo, dado que es la única cuyo **title** posee un valor exactamente igual a "facebook", que es la condición que requiere el selector de nuestra regla de estilo.

Podríamos también ser un poco más genéricos definiendo el selector de la siguiente forma:

```
p[title*="face"] { color:red; }
```

En el ejemplo, el asterisco indica que el valor del atributo **title** debe contener la palabra "facebook", pero no necesariamente debe ser igual. Veamos otro ejemplo en el código HTML:

```
<p title="facebook"> Parrafo 1 </p>
<p title="seguinos en face"> Párrafo 2 </p>
<p title="votanos en facebook"> > Párrafo 3 </p>
```

La regla de estilo aplicará a los tres párrafos, ya que todos poseen la palabra "face" como parte de la cadena de caracteres del valor del atributo **title.**

Otra posibilidad similar consiste en enunciar el selector utilizando el signo **~**, que implica "que contenga exactamente":

```
p[title~="face"] { color:red; }
```

Si bien este caso es similar al que vimos previamente, es más estricto; la condición que se debe cumplir es que el valor del atributo **title** contenga la palabra "face", pero esta no puede ser parte de una palabra, sino que debe ser la palabra completa.

Esto significa que si el valor fuera **title="facebook"**, la condición requerida ya no se cumpliría.

También podemos definir el selector de atributo de manera que podamos detectar el comienzo o el final de una cadena de caracteres, como ejemplificamos a continuación:

```
a[href^="http://"] { color:red; }
```

En este caso, se aplicará la regla de estilo a todos los elementos **<a>** que tengan el atributo **href**, comenzando con el valor **http://**. O bien podemos definir el selector de forma inversa:

```
a[href$=".pdf"] { color:red; }
```

De este modo, la regla de estilo afectará a todos los elementos **<a>** que tengan el atributo **href**, terminado en **.pdf.**

Figura 4. Usando el selector de atributo podemos aplicar estilos diferenciados a este listado de links.

Por último, tenemos una variante de este selector un poco más compleja:

```
article[class|="columna"] { width:600px; }
```

En este caso, se seleccionarán los elementos **<article>** que tengan el atributo **class** y cuyo valor esté declarado entre guiones. Veamos cómo sería el documento HTML al que aplica:

```
<main class="contenedor">
  <article class="columna-izq"></article>
  <article class="columna-der"></article>
</main>
```

Pseudoclases y pseudoelementos

Las pseudoclases y los pseudoelementos se utilizan para aplicar estilos a un estado particular de un elemento; por ejemplo, cuando el usuario pasa el cursor sobre algún elemento.

Las pseudoclases se fueron especificando junto con las distintas versiones de CSS: en CSS3 se han agregado una gran cantidad de pseudoclases que permiten realizar selecciones más específicas.

Es importante resaltar que **no todas las pseudoclases son soportadas por todos los navegadores**. Por citar un ejemplo, Internet Explorer incluye soporte para muchas de ellas recién desde sus versiones 9 y 10. Veamos el listado completo de pseudoclases y pseudoelementos y cuál es su uso específico:

LISTADO DE PSEUDOCLASES Y PSEUDOELEMENTOS

SELECTOR	DESCRIPCIÓN	VERSIÓN
:link	Selecciona los links en su estado inicial.	1
:active	Selecciona los links cuando el usuario hace clic sobre ellos.	1
:visited	Selecciona los links que fueron visitados.	1
:hover	Aplica un regla de estilo cuando el usuario pasa el cursor sobre el elemento.	1

LISTADO DE PSEUDOCLASES Y PSEUDOELEMENTOS		
:first-letter	Selecciona la primera letra de un bloque de texto.	1
:first-line	Selecciona la primera línea de un bloque de texto.	1
:focus	Selecciona el elemento que tiene foco.	2
:before	Aplica estilos antes de un elemento.	2
:after	Aplica estilos después de un elemento.	2
:first-child	Selecciona el primer elemento hijo.	2
:lang	Selecciona elementos que poseen el atributo **lang**.	2
:first-of-type	Selecciona el primer elemento de un tipo determinado.	3
:last-of-type	Selecciona el último elemento de un tipo determinado.	3
:only-of-type	Selecciona el único elemento de un tipo determinado.	3
:only-child	Selecciona el elemento que es único hijo.	3
:nth-child(n)	Permite seleccionar un hijo determinado.	3
:nth-last-child(n)	Permite seleccionar un hijo determinado partiendo desde abajo.	3
:nth-of-type(n)	Permite seleccionar un elemento determinado.	3
:nth-last-of-type(n)	Permite seleccionar un elemento determinado contando desde abajo.	3
:last-child	Selecciona el elemento que es último hijo.	3
:root	Refiere al elemento raíz del documento.	3
:empty	Aplica estilos a los elementos vacíos.	3
:target	Selecciona un elemento que posee un target.	3
:enabled	Aplica estilos a los elementos cuyo atributo es igual a **enabled**.	3
:disabled	Aplica estilos a los elementos cuyo atributo es igual a **disabled**.	3

LISTADO DE PSEUDOCLASES Y PSEUDOELEMENTOS		
:checked	Aplica estilos a los elementos cuyo atributo es igual a **checked**.	3
:valid	Aplica estilos a elementos con estado válido.	3
:invalid	Aplica estilos a elementos con estado inválido.	3
:not(selector)	Se utiliza para crear excepciones.	3
::selection	Aplica estilos al contenido seleccionado por el usuario.	3

Tabla 1. Listado completo de pseudoclases y pseudoelementos.

Pseudoclases y pseudoelementos de CSS3

Como pudimos ver en la tabla del apartado anterior, en CSS3 se ha incorporado una gran cantidad de pseudoclases nuevas para realizar selecciones más complejas o bien que mejoran la experiencia del usuario y simplifican la labor de los desarrolladores. Veamos en detalle en qué casos podemos usar cada uno de estos nuevos selectores.

Pseudoclases de estado de elementos de UI

Este grupo de pseudoclases se utiliza para crear estilos para estados avanzados de elementos de interfaz de usuario (UI). Son particularmente útiles para aplicar a elementos de formularios que utilicen atributos de HTML5 para realizar la validación de los campos. Mediante estas pseudoclases podemos aplicar estilos a los distintos estados de los elementos. Veamos qué pseudoclases forman parte de este grupo:

:valid
:invalid
:enabled
:disabled
:checked

Supongamos que estamos creando un formulario que utilice validación nativa de HTML5 y tenemos el siguiente código HTML:

```
<input type="text" required>
```

Entonces podemos usar la pseudoclase **:valid** para crear el estilo que verá el usuario cuando la información con la que complete el campo sea la esperada. La sintaxis sería la siguiente:

```
input:valid {border:1px solid green;}
```

Podemos, también, crear otra regla de estilo que se aplicará cuando el usuario introduzca información que no se ajuste al formato requerido por la validación. La sintaxis para esa regla de estilo sería:

```
input:invalid {border:1px solid red;}
```

También podríamos aplicarle estilos que dependan de si el campo está activo por defecto (**enabled**) o inactivo (**disabled**).

Supongamos, ahora, que tenemos el siguiente código HTML:

```
<input type="text" disabled>
```

En el archivo de estilo, tendríamos la siguiente regla de estilo:

```
input:disabled {background:#ccc;}
```

Por el contrario, si el elemento del formulario estuviera activo por defecto, como en la mayoría de los casos, en el archivo de estilo tendríamos la siguiente regla de estilo:

```
input:enabled {background:#333;}
```

Por último, podríamos crear un estilo sumamente útil para aplicarles a los **input** de tipo **checkbox** y, así, indicar visualmente cuando hayan sido chequeados. Veamos el ejemplo:

```
input[type="checkbox"]:checked {border:1px solid green;}
```

Figura 5. Mediante el uso de pseudoclases, podemos crear reglas de estilo para los formularios que usan atributos de HTML5.

Pseudoclases estructurales

Las pseudclases de este grupo permiten seleccionar un elemento específico dependiendo de su posición en el flujo del archivo HTML. Veamos una a una las pseudoclases que forman parte de este grupo.

:root

Se utiliza para seleccionar el elemento raíz del documento, que generalmente es el elemento **<html>**. A continuación, vemos su sintaxis:

```
html:root {color:red;}
```

:first-child y **:last-child**

Permiten seleccionar el primero y el último elemento hijo.

Considerando que tenemos el siguiente código HTML:

```
<ul>
  <li>Primero</li>
  <li>Segundo</li>
  <li>Tercero</li>
  <li>Cuarto</li>
  <li>Quinto</li>
  <li>Sexto</li>
  <li>Séptimo</li>
  <li>Octavo</li>
  <li>Noveno</li>
  <li>Décimo</li>
</ul>
```

En el archivo de estilo, tenemos las siguientes reglas de estilo:

```
li:first-child{color:red;}

li:last-child{color:blue;}
```

El resultado visual que obtendremos es que el texto del primer **<li>** se verá de color rojo, mientras que el texto del último **<li>** se mostrará de color azul.

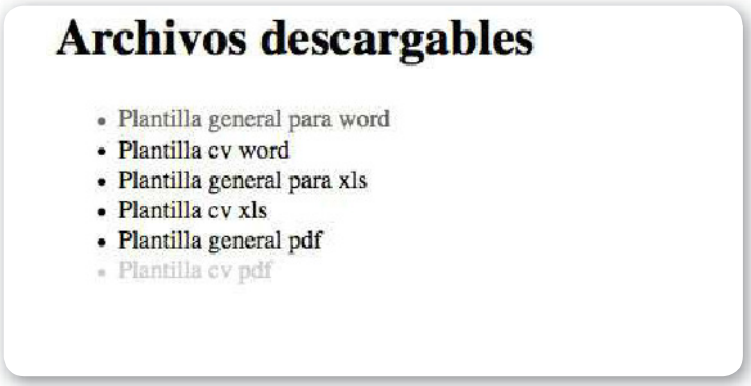


Figura 6. Las pseudoclases **`:first-child`** y **`:last-child`** permiten controlar de forma independiente el primer y el último elemento de la lista.

:nth-child y **:nth-last-child**

Entre las nuevas pseudoclases, encontramos las que comienzan con la sigla **nth**, las cuales son muy útiles para aplicar estilos que formen patrones de repetición dentro de un grupo de elementos del flujo del documento. También son muy útiles para seleccionar un elemento específico. La sintaxis para esta pseudoclase es:

```
li:nth-child(n){....}
```

El parámetro **(n)** determina el patrón al cual queremos aplicar la regla de estilo o el elemento específico que queremos seleccionar. Las posibilidades con este selector son muchas; veamos algunos casos de uso muy provechosos.

Supongamos que queremos aplicar una regla de estilo al sexto **<li>** de nuestra lista: una opción válida es agregarle una clase al elemento **<li>**, pero una mejor opción es utilizar la siguiente pseudoclase:

```
li:nth-child(6){color:green;}
```

En este caso, el contenido del **<li>** número seis se verá de color verde. Como podemos ver, esta pseudoclase admite un parámetro numérico que determina el elemento hijo al cual se le aplicarán los atributos de la regla de estilo.

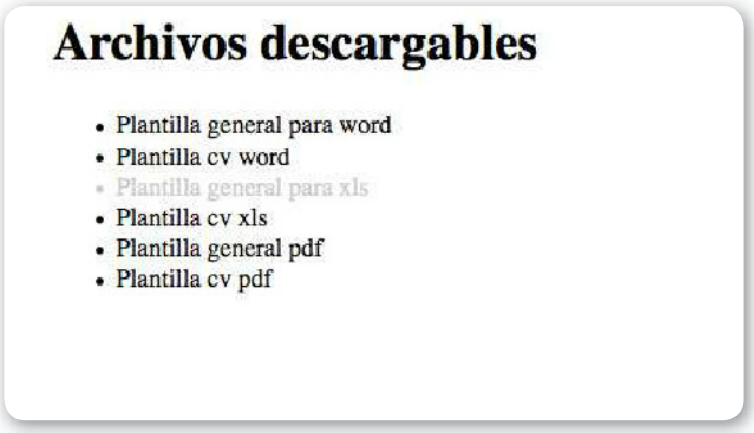


Figura 7. Mediante la pseudoclase **`:nth-child`** es posible aplicar estilos a un elemento específico de la lista.

Otros valores que podemos usar como patrón son las palabras reservadas **odd** (impar) y **even** (par) para crear las siguientes reglas de estilo:

```
li:nth-child(odd){color:red;}

li:nth-child(even){color:blue;}
```

Con estas dos reglas de estilo, podemos crear lo que se conoce comúnmente como el **diseño de la cebra**, ya que la primera regla de estilo aplicará a todos los **<li>** impares y la segunda, a todos los **<li>** pares.

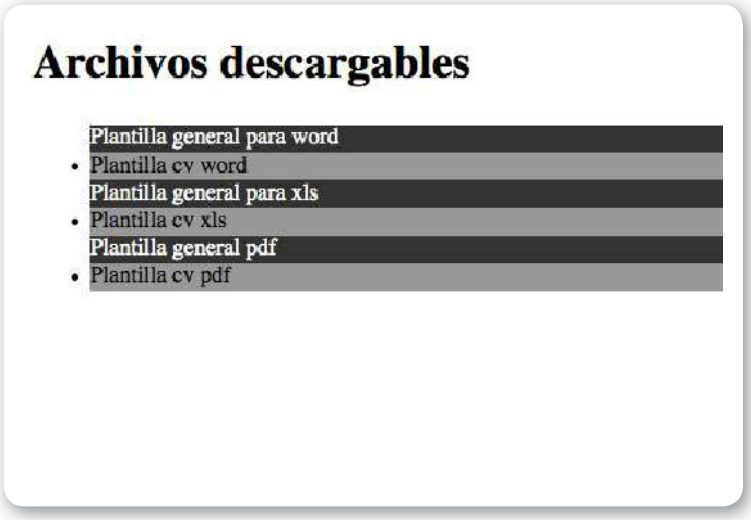


Figura 8. Mediante la pseudoclase **`:nth-child`** podemos crear el conocido diseño de cebra para aplicar a listas y tablas.

También podríamos aplicar pequeñas fórmulas, por ejemplo:

```
li:nth-child(2n+1){...}

li:nth-child(2n+0){...}

li:nth-child(4n+1){...}

li:nth-child(3n-2){...}
```

Todos estos ejemplos que vimos podríamos también definirlos por medio del siguiente selector:

```
nth-last-child(n){...}
```

Este selector es exactamente igual al que vimos anteriormente, pero con una pequeña diferencia: el patrón se aplicará contando desde abajo hacia arriba. Es decir, que si el parámetro indica que tiene que seleccionar el elemento número tres, será el tercero contando desde el último hacia arriba.

Pseudoclase de negación :not()

Esta pseudoclase se utiliza para aplicar estilos a elementos que no fueron seleccionados por un selector simple; es decir que permite crear excepciones. Entendamos su sintaxis con un ejemplo:

```
:not(p){...}
```

Pensemos un caso de uso más real, suponiendo que tenemos el siguiente código HTML:

```
<article id="cap1"> ... </article>
<article id="cap2"> ... </article>
<article id="cap3"> ... </article>
<article id="cap4"> ... </article>
<article id="cap5"> ... </article>
<article id="cap6"> ... </article>
```

Si la idea es crear una regla de estilo que aplique a todos los elementos **<article>**, menos al que posee **cap2** como **ID**, podríamos entonces crear el siguiente estilo:

```
article:not(#cap2) { border:2px solid red;}
```

En este caso, se aplicará el **border** a todos los elementos **<article>** que posean un **ID** distinto de **cap2.**

¿Cuál es el beneficio, en este caso, de usar este selector? Sin él, la regla de estilo que hubiéramos necesitado crear tendría mucho más código y, además, excluiría a otros posibles elementos que podríamos agregar en el futuro.

A continuación, vemos cómo sería el código sin usar el selector **:not**:

```
#cap1, #cap3, #cap4, #cap5, #cap6 {
border:2px solid red;}
```

Pseudoclase de destino :target

Esta pseudoclase permite seleccionar un elemento **<a>** que contenga como **target** la URL de la página actual, para aplicar estilos al hacer clic sobre un link y, de esta forma, crear efectos interesantes cuando se navega dentro del mismo archivo. Veamos un ejemplo simple:

```
<a href="#target">Link al mismo archivo</a>
<div id="target">Contenido</div>
```

En el documento de estilo, tendríamos:

```
div:target {Contenido de la caja }
```

Pseudoelemento ::selection

Este pseudoelemento permite aplicar una regla de estilo al texto que sea seleccionado por el usuario. Por ejemplo, podríamos cambiar el color de fondo o de texto en el momento en que el usuario selecciona el contenido, con el fin de mejorar la visibilidad del texto y, por ende, la experiencia del usuario. Veamos un ejemplo de la sintaxis de este selector:

```
::selection{color: #fff; background-color:#000;}
```

De esta forma, cuando el usuario seleccione cualquier porción de texto del documento, se verá con texto blanco y fondo negro.

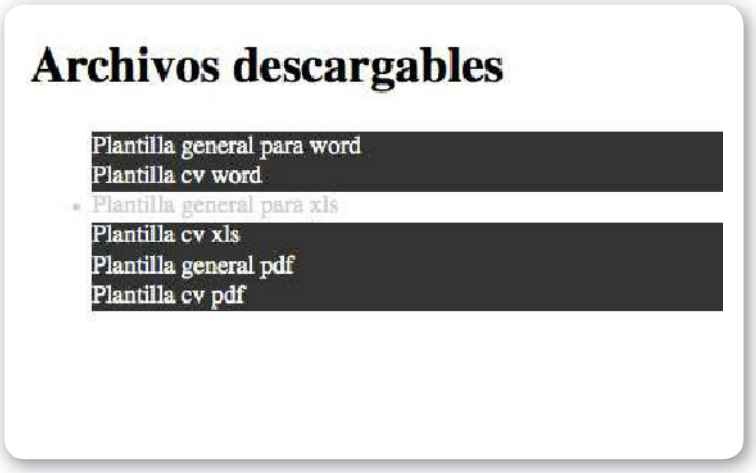


Figura 9. La regla de estilo se aplica cuando el usuario ejecuta la acción de selección sobre el contenido de texto.

Es importante remarcar que, en CSS3, los pseudoelementos se escriben con **doble dos puntos** (**::**) al comienzo, a diferencia del estándar de CSS 2.1 que solo tenía **dos puntos** (**:**). Por lo tanto, deberemos cambiar la forma en la que escribimos todos los pseudoelementos, por ejemplo: **:after {...}** pasa ahora a ser **::after {...}**. El motivo de este cambio tiene que ver con marcar una diferenciación sintáctica entre los pseudolelementos y las pseudoclases, las cuales se siguen escribiendo con la sintaxis ya conocida de dos puntos.

RESUMEN

En este capítulo repasamos el concepto general de selector y los tipos de selectores que pertenecen a las versiones 1 y 2 de CSS. También vimos en profundidad los selectores avanzados de CSS y conocimos en qué casos podemos hacer uso de cada uno de ellos, para lograr optimizar nuestro archivo de estilo.

Actividades

TEST DE AUTOEVALUACIÓN

1. Explique para qué se utiliza el selector de ID.
2. Si necesitamos aplicar un estilo específico a un vínculo, ¿qué selector debemos usar?
3. Para aplicar un estilo a una imagen cuyo atributo **alt** tiene un valor igual a “Facebook”, ¿cuál sería el selector más apropiado?
4. Si necesito crear una regla de estilo que aplique propiedades a los párrafos que están dentro de las etiquetas **<aside>**, ¿qué selector debería utilizar?
5. ¿En qué parte de la hoja de estilo debe incluirse el **selector universal**?
6. Explique cuál es la función del **selector hermano adyacente**.
7. Explique la diferencia entre el **selector hijo** y el **selector descendente**.
8. Si queremos aplicar un estilo al primer párrafo que se encuentra justo después de un etiqueta **<h3>**, ¿cuál sería el selector más adecuado para usar?
9. Si tenemos varios estilos cuyas propiedades se repiten, ¿cuál sería el selector que podríamos usar para optimizar las líneas de código del archivo de estilo?
10. Explique la diferencia entre el **selector hermano** y el **selector hermano adyacente**.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*08 –

Las nuevas capacidades de CSS3

En este capítulo veremos las nuevas propiedades que incluye CSS3 y cómo utilizarlas para crear interfaces de usuario enriquecidas con menor esfuerzo de desarrollo. Aprenderemos, así, a generar archivos más livianos y ágiles a la hora de ser renderizados por el navegador.

- Un nuevo universo de posibilidades 240
- Nuevos modos de color 242
- Nuevas opciones para fondos 245
- Bordes 252
- Sombras 256
- Propiedades de texto 259
- Columnas de texto 262
- Propiedades de Interfaz de Usuario (UI) 265
- Resumen 267
- Actividades 268

↗↗↗

Un nuevo universo de posibilidades

En la versión 3 de CSS, encontraremos una gran cantidad de propiedades nuevas que nos permitirán crear estilos gráficos que, hasta entonces, requerían el uso de imágenes o bien la generación de códigos complejos y poco semánticos. Crear interfaces enriquecidas es ahora mucho más fácil gracias a las múltiples opciones que nos brinda CSS3. Lógicamente, nos toparemos con algunas piedras en el camino, ya que, una vez más, tendremos problemas de soporte con algunos navegadores. Por suerte, la compatibilidad mejora día a día con las nuevas versiones de browsers que salen al mercado.

Es importante que veamos las tablas de compatibilidad antes de hacer uso de las nuevas propiedades. Nuevamente recomendamos el sitio **www.caniuse.com** para ver no solo qué navegadores soportan cada propiedad, sino también cuáles de ellos requieren el uso de prefijos.

Prefijos en etapa de desarrollo

Seguramente ya habrán notado que muchas propiedades de CSS3 utilizan una sintaxis un tanto rara y confusa que utiliza prefijos antes de cada propiedad. Esto se debe a que estamos trabajando con una versión de CSS que aún está en proceso de desarrollo.

La W3C publicó una política de implementación para las propiedades de CSS donde se establece que estas deberán usar un prefijo que haga referencia al navegador cuando se encuentren en

NAVEGADORES PARA DISPOSITIVOS MÓVILES

En el caso de smartphones y tablets, también tendremos diversos navegadores: Safari para iOS, el navegador nativo de Android, Opera mobile, Opera mini, Firefox, Chrome para Android y IE para Windows Phone son los principales. Para todos ellos necesitaremos usar los mismos prefijos que utilizamos para los navegadores tradicionales. Algunas tablas de compatibilidad ya incluyen la información sobre la implementación de las propiedades de CSS para estos navegadores.

estado de desarrollo. Recién cuando la propiedad llegue al estado de candidata a recomendación podrá implementarse de forma estándar.

Esta decisión fue tomada para evitar problemas de compatibilidad entre las implementaciones experimentales de los navegadores y el avance del desarrollo de la estandarización del lenguaje CSS por parte de la W3C.

LISTADO DE PREFIJOS

PREFIJO	MOTOR DE RENDERIZADO QUE REFERENCIA
-mz-	Gecko, utilizado en Firefox.
-ms-	Trident, incluido en Internet Explorer.
-o-	Opera, antes de su versión 15.
-webkit-	Se utiliza para referenciar al motor de renderizado usado por Safari, Chrome y Opera a partir de la versión 15+.

Tabla 1. Cada navegador utiliza un prefijo distinto. En esta tabla vemos los de uso más común.

Es importante mencionar que, a partir de la versión 15, el navegador Opera comienza a usar el motor de renderizado **WebKit**. Por su parte, Chrome deja de usarlo desde su versión 28, ya que comienza a utilizar un motor de renderizado propio llamado **Blink**.

> DEBEMOS VERIFICAR CÓMO SE IMPLEMENTAN LAS PROPIEDADES EN CADA NAVEGADOR

Cuándo usar los prefijos

Si bien ya vimos el porqué del uso de los prefijos, los navegadores no siempre siguen estrictamente la política de implementación de las nuevas propiedades. Es por eso que, a la hora de utilizar cada una de las propiedades, debemos verificar en las tablas de compatibilidad cuáles son los navegadores que requieren prefijos.

Veamos un ejemplo de regla de estilo con prefijos: supongamos que la propiedad **color** es nueva en la especificación de CSS3 y que sabemos –según la tabla– que debemos usar los cuatro prefijos que vimos antes. Nuestra regla de estilo debería ser definida del siguiente modo:

```
h1 {
        color:red;
        -moz-color:red;
        -webkit-color:red;
        -ms-color:red;
        -o-color:red;
}
```

En algunos casos, encontraremos en las tablas de compatibilidad que algunas propiedades no han sido implementadas sin prefijos por ningún navegador. No obstante, siempre debemos incluir la propiedad estándar en nuestra regla de estilo para contemplar las futuras implementaciones de los navegadores.

Nuevos modos de color

En CSS3 se introducen dos nuevos modos de color que nos permiten crear efectos muy interesantes a fondos, color de texto, bordes, etcétera. Veamos la sintaxis de estos nuevos modos de color.

Color RGB

Este modo de color permite definir el valor de un color mediante el sistema RGB, que conocemos y usamos en los aplicativos para diseño.

```
h1 {color:rgb(255,0,0);}
```

Debemos declarar el modo de color (rgb) y luego, entre paréntesis, los valores de rojo, verde y azul. En este ejemplo, el color de texto será rojo pleno. Los valores para el sistema RGB se expresan entre 0 y 255, aunque también podemos definirlos usando porcentajes. Por ejemplo:

```
h1 {color:rgb(100%,0,0);}
```

Color HSL

El modo de color HSL es muy parecido al modo RGB, pero sus parámetros son distintos. Estos se controlan mediante el matiz, la saturación y la luminosidad. Veamos la sintaxis:

```
h1 {color:hsl(75,50%,50%);}
```

En este caso, el primer valor (75) determina el **matiz**; este valor puede variar entre 0 y 360. Los valores siguientes expresan la **saturación** y la **luminosidad**; estos deben estar expresados en porcentajes entre 0 y 100%.

Colores translúcidos

Lo interesante de estos modos de color es que permiten agregar un cuarto canal, el de la transparencia, con el que podemos lograr colores translúcidos para fondos, textos, bordes, etcétera. Ejemplo con canal alfa:

```
h1 {color:rgba(100%,0,0, 0.6);}
```

El canal de la trasparencia admite valores entre 0 y 1; en este caso, el texto de la etiqueta **<h1>** es de color rojo y con 60% de opacidad.

Figura 1. El mismo contenedor con fondo sólido y con fondo translúcido.

Opacidad

La propiedad **opacity** permite controlar la opacidad de un elemento. A diferencia de los modos de colores con canal alfa, esta propiedad controla la opacidad de los elementos y su contenido. No puede ser aplicada, por ejemplo, a un fondo. Si la aplicamos a una etiqueta **article**, el texto que incluya será también translúcido. Veamos la sintaxis de esta propiedad:

```
article {opacity:0.5;}
```

Los posibles valores que admite esta propiedad son numerales entre 0 (totalmente transparente) y 1 (completamente opaco). Por ello, siempre utilizaremos valores intermedios: en el código de ejemplo, el elemento será 50% translúcido. Esta propiedad puede ser aplicada a cualquier elemento y es particularmente útil para crear efectos donde las imágenes y otras etiquetas pueden aparecer o desaparecer.

Figura 2. La propiedad de opacidad afecta a todo el contenido del elemento.

COMPATIBILIDAD CON INTERNET EXPLORER

En IE, esta propiedad fue implementada recién en la versión 9, pero podemos usar algunos trucos si queremos usar la transparencia del elemento en versiones anteriores. En ese caso, deberíamos usar la siguiente línea de código: **filter:alpha(opacity=50)** para lograr un efecto similar al de la propiedad estándar **opacity**.

Nuevas opciones para fondos

El módulo de propiedades de fondo ha incorporado varias mejoras en la versión CSS3. Las más importantes son **background-size**, **background-clip**, **background-origin** y **background-image**. A continuación, veremos en detalle las nuevas propiedades y valores.

Background-size

Esta propiedad permite controlar el tamaño de una imagen de fondo y es sumamente útil para crear fondos que se adapten en diseños líquidos. Veamos la sintaxis de esta propiedad:

```
body {background-image:url(bg.jpg); background-size:100% 100%; }
```

En este caso, el tamaño de la imagen de fondo tendrá siempre 100% de ancho y 100% de alto en la ventana del navegador. Con estos valores, nuestra imagen de fondo posiblemente se deforme en los distintos formatos de pantalla. Para lograr un efecto en el que la imagen se adapte proporcionalmente, podemos usar otros valores predeterminados.

El primero de ellos es **cover**, que escala la imagen de fondo de forma tal que el fondo quede totalmente cubierto por la imagen sin que esta se deforme: es decir, aplica una escala proporcional. Dependiendo de la proporción de la ventana del navegador, es posible que la imagen no se vea completa.

```
body {background-size:cover}
```

El otro valor es **contain**, que ajusta la imagen al máximo tamaño posible, de tal forma que tanto el ancho como el alto de esta entren en la ventana del navegador sin que la imagen se deforme. El resultado que obtendremos con este valor dependerá mucho de la proporción de la imagen y de la ventana del navegador.

```
body {background-size:contain}
```

Background-origin

Esta nueva propiedad de CSS3 permite controlar el punto de origen donde se mostrará la imagen de fondo dentro de la caja. Veamos su sintaxis:

```
div {background-image:url('bg.jpg');
background-origin:content-box;
}
```

Esta propiedad admite tres valores: **content-box**, **padding-box** y **border-box**, que están relacionados a las áreas de la caja. En el ejemplo que vimos, el origen de la imagen de fondo será el área del contenido de la caja, es decir, donde termina el **padding**, siempre y cuando la caja tenga uno.

En cambio, usando el valor **padding-box**, la imagen comenzará desde el área del padding, luego del borde. Por último, el valor **border-box** determinará que la imagen de fondo comenzará desde el área del borde. Este caso será más evidente si usamos un borde discontinuo.

Figura 3. Posibles aplicaciones de la imagen de fondo usando la propiedad **background-origin**.

Background-clip

Esta propiedad es muy parecida a la anterior, pero se relaciona con el color de fondo en lugar de la imagen de fondo.

Veamos un ejemplo:

```
div {background-color:red;
background-clip:content-box;
}
```

Los valores posibles son los mismos que vimos para las propiedades **background-clip**, **content-box**, **padding-box** y **border-box.**

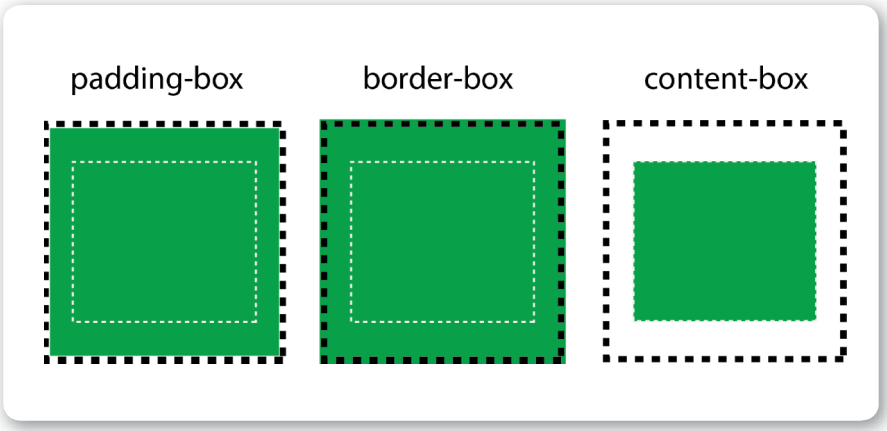


Figura 4. Posibles aplicaciones del color de fondo usando la propiedad **background-clip**.

Múltiples imágenes para los fondos

CSS2 no permitía utilizar más de una imagen de fondo en un contenedor. Por suerte, CSS3 incorpora la posibilidad de usar dos o más imágenes de fondo. Para esto, no necesitaremos ninguna propiedad nueva, ya que ahora **background-image** soporta múltiples valores. Veamos cómo sería la sintaxis:

```
div {
background-image:url('bg.jpg') ,
url('guarda.jpg'), url('logo.jpg');
}
```

En este ejemplo, aplicamos al elemento **body** tres imágenes de fondo. También podemos controlar la repetición y la posición de cada una de estas imágenes de forma independiente haciendo uso

de la propiedad **background-repeat** y **background-position** en el mismo orden en el cual declaramos las URL de las imágenes.

Un punto importante a tener en cuenta es que las imágenes de fondo van a quedar superpuestas, a menos que cambiemos la posición de cada una de ellas. En ese caso, la que esté declarada más cerca de la propiedad será la que estará más arriba en la pila de imágenes.

Fondos degradados

Este es otro de los cambios significativos de CSS3. Antes debíamos usar imágenes para poder crear este tipo de fondos; ahora lo podremos hacer de forma mucho más simple y fácil de editar mediante la propiedad **background-image** y sus nuevos valores para crear distintos tipos de degradados. Veamos un ejemplo:

```
div {background-image : linear-gradient(red, green); }
```

En este caso, tenemos un degradado lineal simple de color rojo a verde. También podemos determinar otros parámetros, como la orientación del degradado y la ubicación de los colores.

Veamos otros posibles valores:

```
//Gradientes lineales
linear-gradient(yellow, blue);
linear-gradient(to bottom, yellow, blue);
linear-gradient(180deg, yellow, blue);
linear-gradient(to top, blue, yellow);
linear-gradient(to bottom, yellow 0%, blue 100%);

//Gradientes lineales de más de dos colores
linear-gradient(135deg, yellow, blue);
linear-gradient(-45deg, blue, yellow);
linear-gradient(to top right, red, white, blue)
linear-gradient(yellow, blue 20%, #0f0);

//Gradientes elípticos y circulares
```

```
radial-gradient(yellow, green);
radial-gradient(ellipse at center, yellow 0%, green 100%);
radial-gradient(farthest-corner at 50% 50%, yellow, green);
radial-gradient(circle, yellow, green);
radial-gradient(red, yellow, green);
radial-gradient(farthest-side at left bottom, red, yellow 50px, green);
radial-gradient(closest-side at 20px 30px, red, yellow, green);
radial-gradient(20px 30px at 20px 30px, red, yellow, green);
radial-gradient(closest-side circle at 20px 30px, red, yellow, green);
radial-gradient(20px 20px at 20px 30px, red, yellow, green);

//Gradientes repetitivos
repeating-linear-gradient(red, blue 20px, red 40px)
repeating-radial-gradient(red, blue 20px, red 40px)
repeating-radial-gradient(circle closest-side at 20px 30px, red, yellow,
      green 100%, yellow 150%, red 200%)
```

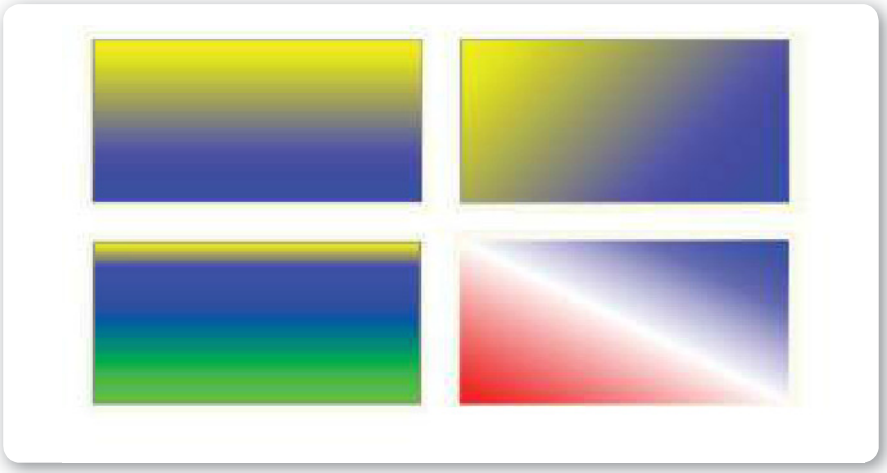


Figura 5. Ejemplos de fondos con degradados lineales.

También podemos indicar "frenos" en los colores –conocidos como **Color Stops**– para lograr distintos efectos degradados:

```
linear-gradient(red, white 20%, blue)
linear-gradient(red 0%, white 20%, blue 100%)
```

```
linear-gradient(red 40%, white, black, blue)
linear-gradient(red 40%, white 60%, black 80%, blue 100%)
linear-gradient(red -50%, white, blue)
linear-gradient(red -50%, white 25%, blue 100%)
linear-gradient(red 80px, white 0px, black, blue 100px)
linear-gradient(red 80px, white 80px, black 90px, blue 100px)
```

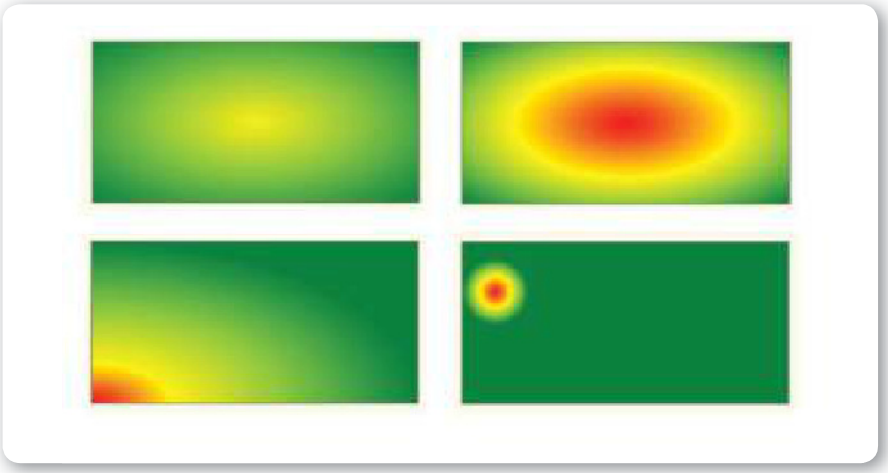


Figura 6. Ejemplos de fondos degradados.

Para crear un fondo degradado, el código suele ser un tanto complejo y engorroso, ya que esta propiedad fue implementada de distinta forma en cada navegador. Veamos un ejemplo de uso:

```
div {
background-image: url(images/fallback-gradient.png);

/* Safari 4+, Chrome 1-9 */
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%, from(#2F2727),
        to(#1a82f7));

/* Safari 5.1+, Mobile Safari, Chrome 10+ */
background-image: -webkit-linear-gradient(top, #2F2727, #1a82f7);

/* Firefox 3.6+ */
background-image: -moz-linear-gradient(top, #2F2727, #1a82f7);
```

```
/* IE 10+ */
background-image: -ms-linear-gradient(top, #2F2727, #1a82f7);

/* Opera 11.10+ */
background-image: -o-linear-gradient(top, #2F2727, #1a82f7); }
```

Por suerte, existen herramientas gratuitas y fáciles de usar a la hora de crear fondos con gradientes elaborados. Entre estas, podemos encontrar el generador **ColorZilla** (**www.colorzilla.com/gradient-editor**), que nos permitirá crear gran cantidad de degradados compatibles con todos los navegadores de uso frecuente.

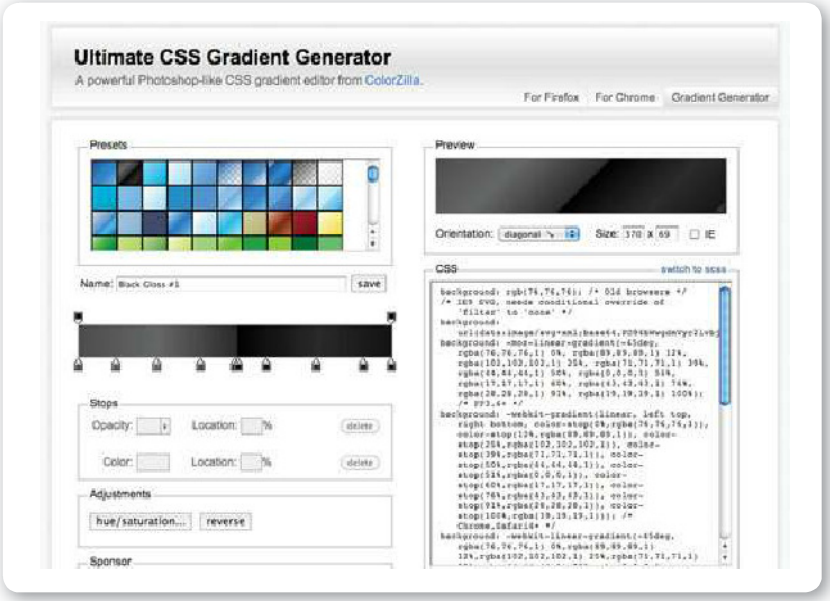


Figura 7. **ColorZilla** permite crear fácilmente degradados online, simplificando la creación del código.

COMPATIBILIDAD CON INTERNET EXPLORER

Si bien esta propiedad se implementó en la versión 9 de IE, hay trucos para aplicar un fondo degradado. Uno de los filtros de Microsoft permite lograr un efecto similar al de las propiedades estándar de CSS. Para ello, usaremos un código como: **filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#1e5799', endColorstr='#7db9e8',GradientType=0)**;. Este filtro es compatible con IE 6 y superiores.

Bordes

En el módulo de bordes también se han incorporado mejoras significativas, como las nuevas propiedades **border-image** y **border-radius**, que nos permiten lograr efectos visuales muy interesantes. Observemos estas propiedades en detalle.

Border-radius

Esta es, probablemente, una de las propiedades más conocidas y usadas de CSS3. Todos aquellos que tuvieron que crear un ** ** con los bordes redondeados la conocerán. Ya no tenemos que complicarnos para hacer códigos complejos y poco semánticos, pues **border-radius** permite redondear los bordes de un elemento HTML, inclusive de una imagen. Veamos la sintaxis:

```
div {border-radius:30px}
```

En este caso, los bordes del elemento quedarán redondeados por **30px**, todos por igual. Esto se debe a que **border-radius** es shorthand (modo abreviado) para todos los valores del borde. Esta regla de estilo equivale a:

```
div {   border-top-left-radius:30px;
        border-top-right-radius:30px;
        border-bottom-left-radius:30px;
        border-bottom-left-radius:30px;}
```

Siempre es recomendable utilizar las formas abreviadas de escritura para hacer más compacto el archivo CSS. Si quisiéramos crear un ** ** cuyos extremos tengan distinto radio de curvatura cada uno, también podríamos usar la escritura abreviada de la misma forma en que lo hacemos con las propiedades **margin** o **padding**:

```
div{border-radius:30px 10px 0 5px;}
```

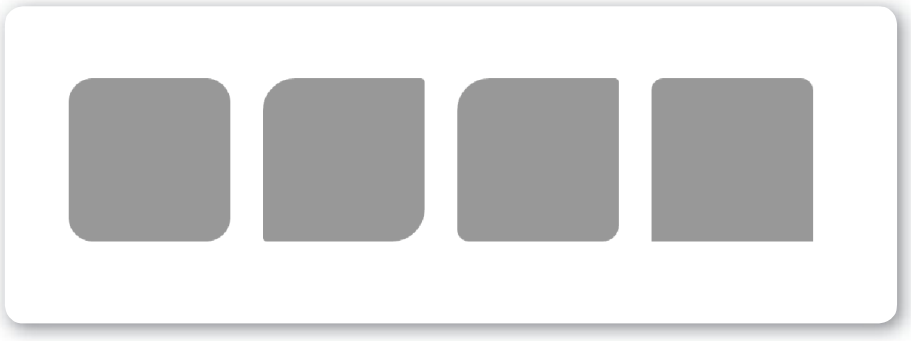


Figura 8. Ejemplos de cajas con **`border-radius`**.

También podemos definir de forma independiente el radio de curvatura del **eje X** con respecto al radio de curvatura del **eje Y** para cada una de las aristas de la caja. En este caso, la sintaxis sería:

```
div {border-radius:30px / 5px;}
```

En este ejemplo, la caja tendría los cuatro bordes iguales con un radio de curvatura de 30 px de ancho y 5 px de alto.

Figura 9. Ejemplos de cajas con **`border-radius`** manipulando la curvatura de los ejes X e Y.

Border-image

Esta es otra de las propiedades más interesantes de CSS3, con la cual podemos crear estilos de bordes de caja utilizando imágenes que definen marcos que se ajustan al tamaño de la caja. Esta propiedad nos permite utilizar una única imagen como borde. Lo importante de esta propiedad es cómo diseñamos la imagen. Veamos su sintaxis:

```
div {border-image: url("border.png") 27 round stretch; }
```

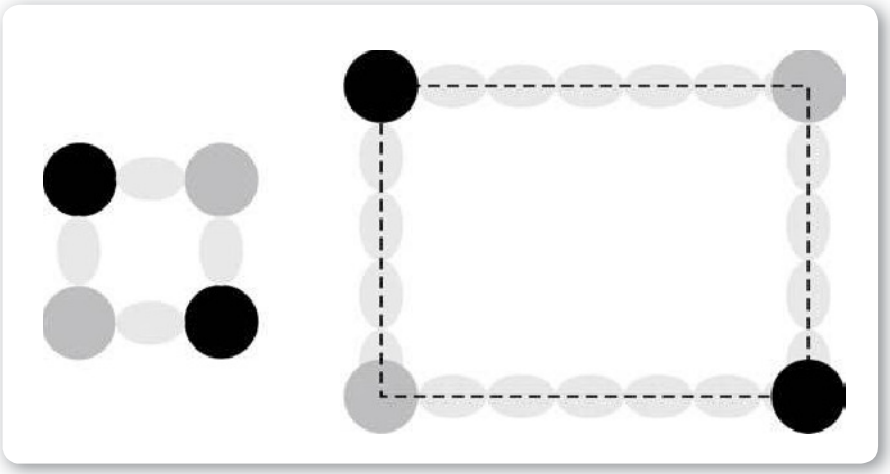


Figura 10. Imagen original y aplicada como imagen de borde.

En este ejemplo, se está utilizando el shorthand de la propiedad **border-image** para hacer más acotado el archivo CSS. Veamos las propiedades que podemos utilizar:

- **border-image-source**: define la URL de la imagen del borde.
- **border-image-slice**: determina la porción de imagen que se va a utilizar.
- **border-image-width**: fija el ancho del borde.
- **border-image-repeat**: especifica el tipo de repetición de la imagen de slice.
- **border-image-outset**: establece el outset del borde.

La forma abreviada de esta propiedad es la que muestra la siguiente sintaxis:

```
border-image: source slice width outset repeat;
```

Border-image-slice

Esta propiedad está relacionada con el diseño de la imagen que usaremos para el borde de la caja. Es muy importante entender cómo crear la regla de estilo. El **slice** es la porción de la imagen que se

tomará para crear el borde y las esquinas. Por esta razón, debemos usar como valor el tamaño según la imagen que utilizaremos. El valor se puede determinar mediante valores absolutos o relativos.

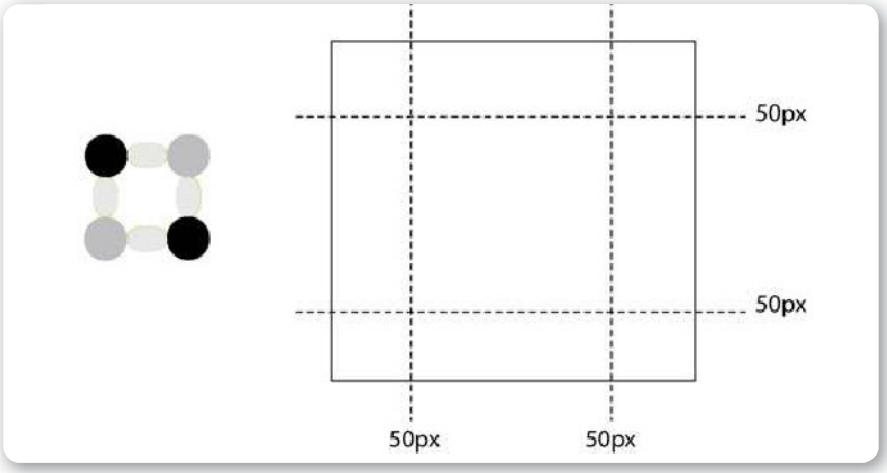


Figura 11. Imagen original y gráfico explicativo del slice.

Border-image-repeat

La propiedad **border-image-repeat** permite crear distintos efectos para los bordes, ya que controla el modo de repetición que va a tener la porción de imagen que se ubicará en los lados de la caja. Esta propiedad admite cuatro valores: **repeat**, **round**, **stretch** y **space.**

> LA PROPIEDAD BORDER-IMAGE NO ES BIEN SOPORTADA POR TODOS LOS NAVEGADORES

- **repeat**: este valor repetirá la porción de imagen en los lados de la caja, dado que el valor del slice y el ancho o alto de la caja pueden no ser proporcionales. Es posible que las últimas repeticiones del slice se vean recortadas.
- **round**: al igual que **repeat**, este valor repite la porción de imagen pero con un número entero. De esta forma, ajusta el tamaño de la imagen para que todas las repeticiones queden enteras.
- **space**: del mismo modo que **round**, el valor **space** repite la porción de imagen con un número entero, pero no deforma la imagen. En su lugar, agrega espacios para completar el ancho y el alto de la caja.
- **stretch**: este valor estira el **slice** para que se ajuste al ancho o al alto de la caja. Con este valor, la imagen siempre quedará deformada.

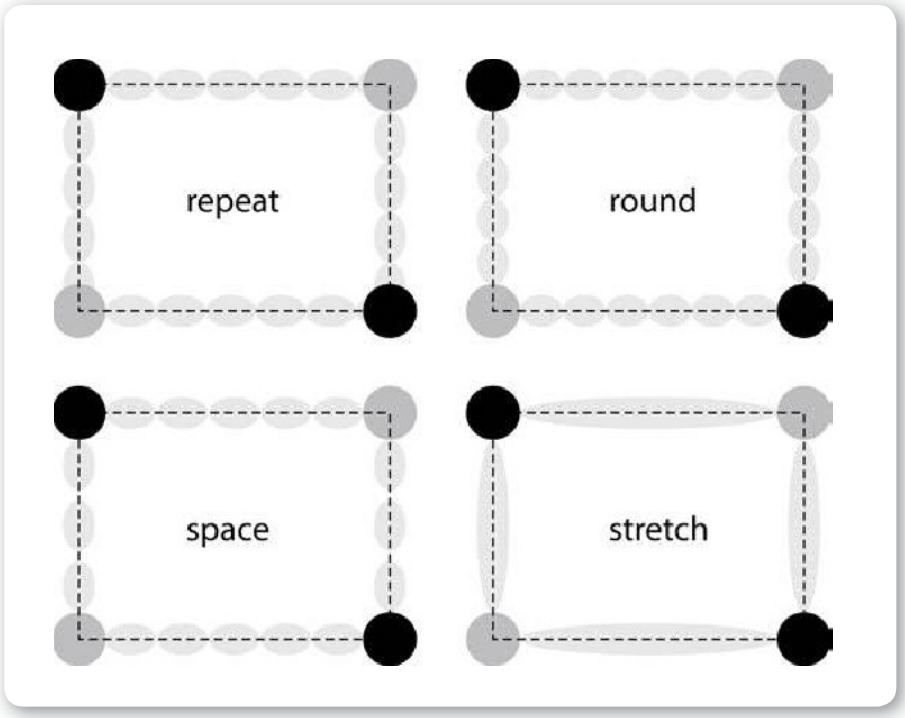


Figura 12. Cajas con las mismas imágenes de borde y distintos valores.

Sombras

Otra de las capacidades destacadas de CSS3 consiste en crear sombras para los elementos. Antes estábamos obligados a agregar muchas imágenes para crear efectos de sombra, tornando más lenta la descarga de nuestro sitio web y más complejo el mantenimiento.

Ahora podemos crear sombras de modo simple mediante código y lograr efectos muy atractivos, de bajo peso y muy fáciles de editar.

HERRAMIENTA ONLINE PARA BORDER-IMAGE

Ya aprendimos cómo trabajar con la propiedad **border-image**; ahora haremos nuestra vida más simple. En el sitio web **http://border-image.com** encontraremos un generador de código muy interesante para trabajar con esta propiedad en particular: bastará con subir la imagen que queremos usar y jugar con los controles hasta lograr el borde deseado. Por último, habrá que copiar y pegar el código en un archivo de estilo.

Podemos aplicar sombras a las cajas mediante la propiedad **box-shadow**, y a los textos, mediante **text-shadow.**

Box-shadow

Esta propiedad permite agregar efectos de sombra a los elementos de HTML y posee una serie de parámetros que nos permitirán crear distintos tipos de sombras. Los parámetros que podemos controlar son: desplazamiento horizontal, desplazamiento vertical, desenfoque de la sombra, color, posición y radio de difusión. Veamos un ejemplo:

```
div {box-shadow:3px 5px 20px #000;}
```

En este ejemplo, usamos los parámetros mínimos y necesarios para esta propiedad: la sombra tendrá un desplazamiento en el eje X de 3 px y un desplazamiento en el eje Y de 5 px. Además, mostará un difuminado de 20 px y será de color negro. Es importante utilizar la escritura abreviada de esta propiedad, ya que, de lo contrario, estaremos usando muchas líneas de código innecesarias en nuestro archivo de estilo.

```
div {box-shadow:3px 5px 20px 10px #000 inset;}
```

En este ejemplo, usamos todos los parámetros posibles y le hemos sumado un radio de difusión de 10 px. Este valor se utiliza para determinar si el tamaño de la sombra es igual o menor al del elemento. Además, usamos el valor **inset** para determinar que la sombra es interior. También podemos aplicar más de una sombra para crear distintos efectos creativos. En ese caso, la sintaxis sería:

```
div {box-shadow:3px 5px 20px #000 , -10px -10px 5px #333 , 13px 15px 5px #ccc;}
```

En este ejemplo, el elemento posee tres sombras cuyos valores van separados por comas. También es importante recordar que los parámetros numéricos pueden usar tanto valores positivos como negativos.

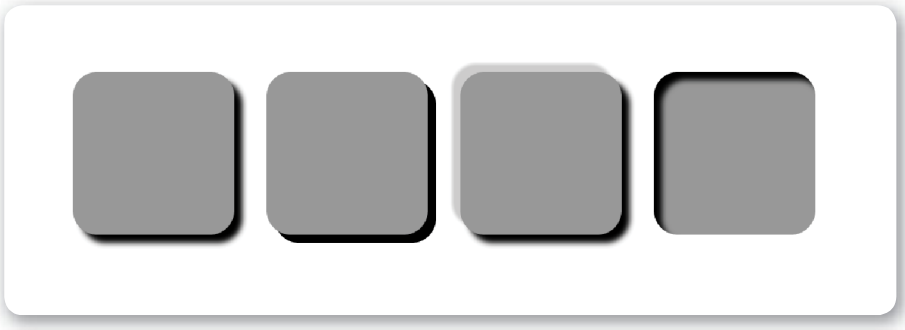


Figura 13. Distintos tipos de sombra para las cajas.

Text-shadow

La propiedad **text-shadow** es muy similar a la que acabamos de ver, pero su uso es específico para elementos de texto. También posee menos parámetros que **box-shadow**. Veamos la sintaxis:

```
h1{text-shadow:3px 5px 20px #000;}
```

En este ejemplo, estamos usando todos los parámetros posibles para la propiedad **text-shadow**. El título **h1** poseerá una sombra con un desplazamiento horizontal de 3 px, un desplazamiento vertical de 5 px, y un difuminado de 20 px de color negro. También es posible agregar más de una sombra para lograr distintos efectos visuales; en ese caso, deberíamos separar los valores con comas, tal como se indica en el siguiente ejemplo:

```
h1{text-shadow:3px 5px 20px #000 , 13px 15px 5px #f50 ;}
```

HACER MÁS FÁCIL CSS3

Si bien escribir código CSS3 es bastante sencillo, hoy en día contamos con herramientas web que nos permiten hacer nuestro trabajo mucho más fácil y rápido. En el sitio **http://css3generator.com** encontramos una poderosa herramienta gratuita que nos permite crear reglas de estilo de manera muy simple y con un entorno visual que nos da la posibilidad de visualizar en tiempo real cómo se verá el estilo creado.

Figura 14. Distintos tipos de sombra aplicados a los textos.

Propiedades de texto

Existe un grupo de nuevas propiedades para utilizar en los elementos de texto, que permiten tener mayor control sobre la maquetación y brindar una mejor experiencia al usuario. Veamos en detalle cada una de estas propiedades.

Word-wrap

Esta propiedad permite controlar el corte de palabras para que el texto se ajuste a un contenedor determinado. Supongamos que tenemos un contenedor con un tamaño específico y, dentro de este, tenemos una palabra que es demasiado larga. Dado que el navegador no posee la capacidad de separar las palabras en sílabas, esta palabra quedará por fuera de su contenedor. Para resolver este problema, podemos usar la propiedad **word-wrap** con el valor **break-word**. De esta manera, el texto siempre se ajustará al contenedor. Veamos cómo sería la sintaxis:

> TEXT-OVERFLOW NOS PERMITE CONTROLAR LOS TEXTOS QUE SUPERAN EL TAMAÑO DE LA CAJA

```
p {word-wrap:break-word;}
```

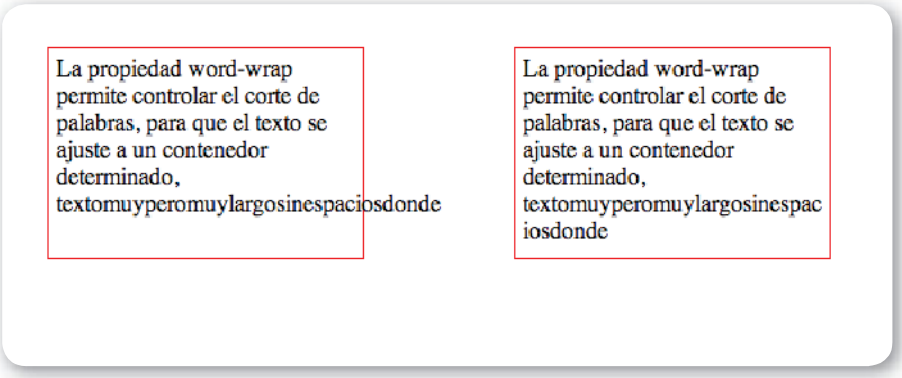


Figura 15. Bloque de texto controlado mediante la propiedad **word-wrap**.

Text-overflow

La propiedad **text-overflow** permite cortar un texto y agregarle de forma automática puntos suspensivos para indicar que el texto continúa. Esto es muy útil cuando tenemos contenedores de tamaño fijo cuyos contenidos pueden variar la cantidad de caracteres que poseen. Veamos la sintaxis de esta propiedad:

```
div {text-overflow:ellipsis;}
```

Los posibles valores para esta propiedad son:

- **ellipsis**: agrega puntos suspensivos para indicar que el texto continúa.
- **clip**: corta y oculta el texto que no entra en el contenedor.
- **string**: es el valor por defecto y no aplicará ningún efecto.

Para que la propiedad **text-overflow** funcione correctamente, debe ser combinada con otras propiedades:

```
div {
white-space:nowrap;
text-overflow:ellipsis;
width:300px;
overflow:hidden;
```

```
word-wrap:normal;
}
```

La propiedad **overflow** debe tener un valor distinto al de la propiedad visible. La propiedad **white-space** debe tener el valor **nowrap** o **per**, y la etiqueta debe tener una ancho determinado.

Aunque no es estrictamente necesario, es recomendable agregar la propiedad **word-wrap** con el valor normal, ya que si el valor para esta propiedad fuera **break-word**, no funcionaría en IE.

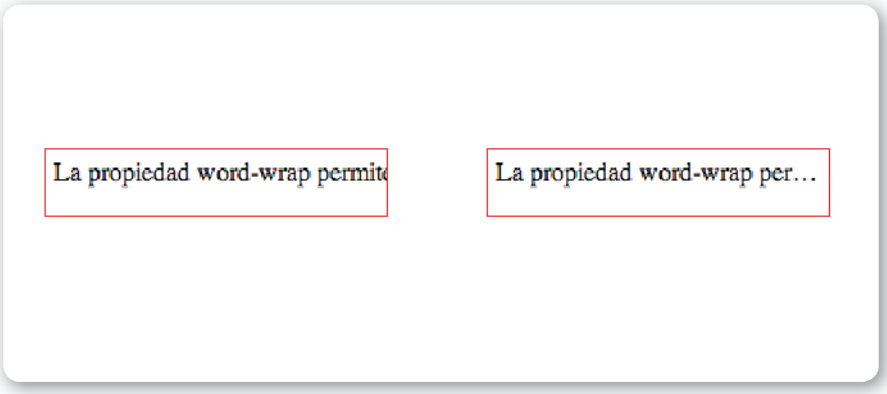


Figura 16. En este caso, podemos ver dos bloques de texto con la propiedad **`text-overflow`**.

Word-break

La propiedad **word-break** se utiliza para controlar la separación en sílabas; es especialmente útil cuando el sitio posee múltiples idiomas, dado que podemos establecer qué tipo de separación utilizará cada uno. Veamos su sintaxis:

OTRAS PROPIEDADES DE TEXTO

Existen algunas otras propiedades de texto que aún no han sido implementadas por ningún navegador y cuya especificación todavía se encuentra en estadio de borrador. Tendremos que esperar algún tiempo más para saber el futuro de estas propiedades y poder comenzar a usarlas. Algunas de ellas son: **text-align-last**, **hanging-punctuation**, **punctuation-trim**, **text-emphasis**, **text-justify** y **text-outline.**

```
div { word-break: keep-all }
```

Los valores posibles que admite son: **normal**, **break-all** y **keep-all**; cuando se usa el valor **normal**, el corte de palabras se realiza según las reglas normales. El **valorbreak-all** podría agregar un guión entre dos letras cualquiera, mientras que el **valor keep-all** nunca agregará un guión entre pares de letras.

Columnas de texto

Una capacidad importante que incorpora CSS3 es la posibilidad de agregar columnas en los textos para lograr diseños editoriales más simples de maquetar. Para ello, vamos a contar con nuevas propiedades, que enumeramos a continuación y veremos en detalle en las páginas siguientes:

column-count
column-width
column-gap
column-rule
column-rule-color
column-rule-style
column-rule-width
column-fill
column-span
columns

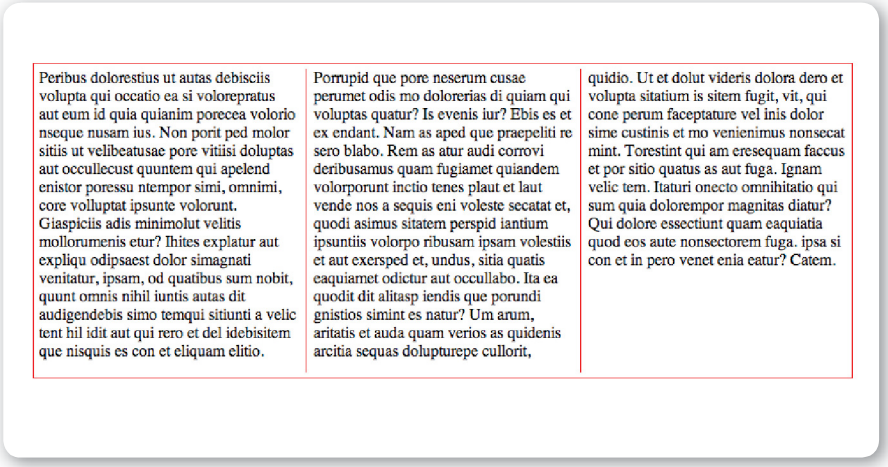


Figura 17. Con las propiedades para las columnas podemos crear diseños editoriales fáciles de editar.

Column-count

Se utiliza para determinar la cantidad de columnas que queremos crear:

```
div {
column-count:3;
}
```

Column-width

Esta propiedad se utiliza para determinar el ancho de las columnas:

```
div{
column-width:200px;
}
```

Al determinar el ancho de la columna, el navegador calculará la cantidad de columnas que debe crear.

Column-gap

Esta propiedad sirve para determinar el ancho de la calle (separación) entre una columna y otra.

```
div {
column-gap:20px;
}
```

Column-rule

Su función es crear una línea divisoria entre una columna y otra:

```
div {
column-rule:2px solid red;
}
```

Este propiedad es el shorthand para las propiedades **column-rule-color**, **column-rule-width** y **column-rule-style**. Es recomendable su uso para reducir la cantidad de líneas de código en el archivo de estilo.

Column-fill

Esta propiedad se utiliza para indicar cómo serán llenadas las columnas. Los posibles valores son **balance** y **auto**:

```
div {
column-fill:balance;
}
```

Si el valor utilizado es **balance**, el navegador deberá minimizar la variación en el ancho de las columnas, logrando que queden balanceadas. Por el contrario, si el valor es **auto**, el ancho dependerá de su contenido.

Column-span

Esta propiedad se utiliza para determinar si los elementos de texto, como los títulos, se ajustan a la primera columna o ignoran las columnas de texto. Los posibles valores para esta propiedad son **all** y **1**:

```
h2{
column-span:all
}
```

Columns

Esta propiedad es el modo abreviado para las propiedades **column-count** y **column-width**. Veamos su sintaxis con un ejemplo:

```
div {
columns: 100px 4;
}
```

Propiedades de Interfaz de Usuario (UI)

Existe un grupo de propiedades de CSS3 a las cuales se las conoce como "propiedades de interfaz de usuario". Por medio de ellas se pueden crear estilos que **enriquecen la interfaz y permiten brindar una mejor experiencia** al usuario. Vamos a encontrar propiedades como **resize**, **box-sizing** y **outline-offset**, entre otras.

Resize

Esta propiedad permite que el usuario pueda redimensionar el ancho y/o el alto de un elemento HTML. Se utiliza del siguiente modo:

```
div {
resize:both;
overflow:auto;
}
```

En este caso, el usuario podrá modificar tanto en ancho como el alto del ** **. Esta propiedad debe usarse en conjunto con la propiedad **overflow**. También es posible configurar solo uno de los lados del elemento y que el otro se establezca automáticamente:

```
div
{
resize:vertical;
overflow:auto;
}


div
{
resize:horizontal;
overflow:auto;
}
```

Box-sizing

Esta propiedad permite manipular el modelo de caja estándar de CSS. De esta forma, podemos determinar si el área de padding o del borde va por dentro del área del contenido, a diferencia del modelo de caja estándar, donde ambos van por fuera. Esta propiedad es muy útil cuando trabajamos con diseños adaptables cuyas medidas deben ser establecidas en porcentajes. Veamos la sintaxis:

```
div {box-sizing:border-box;
        width:100%;
        padding:5%;}
```

En este caso, el padding del 5% estará por dentro de la caja; de no usarse esta propiedad, la caja tendría un ancho real de 110%, ya que el padding se sumaría al ancho de la caja.

Los posibles valores que admite esta propiedad son: **border-box** y **content-box**. El valor **content-box** es el valor por defecto y hace que el padding y el border se calculen por fuera del elemento.

Outline-offset

Esta propiedad se utiliza en conjunto con la propiedad **outline** y permite determinar el desplazamiento. Nos da la posibilidad de crear un desfasaje visual entre el límite físico de la caja y la línea del outline, y se utiliza para crear únicamente efectos visuales en los elementos de HTML.

Veamos cómo trabaja:

DISEÑOS FLUIDOS O LÍQUIDOS

Así se denomina a los diseños que poseen la capacidad de adaptarse al ancho de la ventana del navegador por sí solos, pero no es exactamente lo mismo que los diseños adaptables o responsives. El adaptable requiere que determinemos cuáles son las medidas de quiebre en el diseño en función de los tamaños de pantalla que se quieren optimizar. El diseño fluido es más simple: trabaja directamente con valores relativos, para que se ajusten directamente al ancho del navegador.

```
div
{
border:1px dotted black;
outline:3px solid red;
outline-offset:25px;
}
```

Appearance

Esta propiedad permite modificar la apariencia con la cual se muestra un elemento. Por ejemplo, un ** ** podría mostrarse con la apariencia de un botón:

```
section { appearance:button;}
```

En este caso, la etiqueta **<section>** será renderizada por el navegador con la apariencia de un botón. Los valores posibles para esta propiedad son: **normal**, **icon**, **window**, **button**, **menu** y **field.**

RESUMEN

En este capítulo conocimos las nuevas posibilidades que ofrece CSS3 para jugar con la apariencia de los elementos. Vimos cómo trabajar con los colores, asignando saturaciones, cambiando la transparencia y hasta creando gradientes. También aprendimos cómo cambiar la apariencia de las cajas, incorporando sombreados, bordes y textos en columnas. Y, además, nos introdujimos en algunas de las novedades de CSS3 para mejorar la interfaz de usuario.

Actividades

TEST DE AUTOEVALUACIÓN

1. Si queremos hacer que una imagen de fondo ocupe siempre el ancho de la ventana del navegador, ¿cuál es la propiedad y el valor que debemos usar?
2. Enuncie los posibles valores para la propiedad **box-shadow** y comente para qué sirve cada uno de ellos.
3. Si queremos lograr que el fondo de un ** ** sea translúcido, pero que los contenidos de texto que posee sean opacos, ¿qué propiedad deberíamos implementar?
4. ¿Cuál es la propiedad que debemos usar para crear columnas de texto en una etiqueta **<section>**?
5. Explique cuáles son y para qué se utilizan los prefijos.
6. ¿Cuál es la propiedad que debemos usar para aplicar un color degradado en una etiqueta **<aside>**?
7. ¿Cuál es la propiedad que debe aplicarse para lograr un efecto de sombra en un texto?
8. Si queremos aplicar bordes redondeados a una imagen, ¿qué propiedad tendremos que utilizar?
9. Explique para qué se utiliza la propiedad **box-sizing**.
10. Explique para qué sirve la propiedad **resize**.

EJERCICIOS PRÁCTICOS

1. Descargue el archivo **tp_cap_8.zip**.
2. Realice la maquetación según el archivo de referencia. El archivo resultante deberá ser compatible con la mayoría de los navegadores.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

09

Interfaz de usuario avanzada

En este capítulo conoceremos algunos elementos avanzados de HTML y CSS para mejorar la experiencia del usuario tanto a nivel visual como interactivo. Aprenderemos a utilizar tipografías especiales, a crear animaciones y transiciones con CSS3 y a "arrastrar y soltar" elementos en la pantalla. Por último, crearemos sitios preparados para ser navegados con los dedos, trabajando con los eventos touch de JavaScript.

- Tipografías especiales.....270
- Transformaciones 275
- Transiciones 282
- Animaciones.....284
- Usar la API para Drag&Drop 287
- Eventos touch.....297
- Resumen.....301
- Actividades.....302

↗↗↗

Tipografías especiales

Una de las grandes limitaciones que teníamos a la hora de diseñar un sitio era la reducida cantidad de tipografías que podíamos usar. Los navegadores utilizaban las fuentes instaladas en cada computadora, lo que hacía que el sitio web se viera distinto en cada equipo. Con CSS3 ya no tendremos ese problema: en estos últimos años se ha hecho masivo el uso de la propiedad **@font-face**, que ofrece la posibilidad de implementar casi cualquier tipografía que deseemos.

La propiedad **@font-face** sirve para indicar la ruta donde el navegador deberá buscar los archivos tipográficos que se utilizarán en el sitio. Esta propiedad es compatible con todos los navegadores, incluso con ediciones viejas de Internet Explorer; de hecho, IE6 fue el primer navegador en implementar la propiedad **@font-face.**

Antes de empezar, hay un dato importante que debemos tener en cuenta: el formato de las tipografías que vamos a utilizar. Si bien son formatos específicos para la Web, lamentablemente no todos los navegadores utilizan el mismo. En consecuencia, deberemos incluir varios archivos tipográficos para contemplar todos los navegadores.

El formato tipográfico que la W3C estableció recientemete como estándar para la Web es **WOFF** (siglas en inglés de *Web Open Font Format*, que en castellano significa **Formato de Fuente Abierto para la Web**). Se trata de un formato especialmente creado para internet, que la mayoría de los navegadores modernos soporta. Sin embargo, no es recomendable utilizar únicamente este.

Los otros formatos que debemos usar son **EOT** (*Embed Open Type*), que es el único que soporta Internet Explorer, **TTF** (*True Type Font*), y **SVG** (*Scalable Vector Graphics*), que se utiliza principalmente para iPhone y iPad.

Implementación de tipografías

En primer lugar, definiremos en nuestro archivo de estilo la propiedad **@font-face** para indicar el nombre y la ruta al archivo de fuente:

```
@font-face {
    font-family: 'miFuente';
```

```
src: url(mifuente.woff);
font-weight:normal;
}
```

Mediante el atributo **src** indicamos la ruta al archivo de fuente, que puede ser relativa o absoluta. Usamos la propiedad **font-family** para declarar el nombre con el cual la tipografía será identificada por el navegador y referenciada desde el archivo de estilo. La propiedad **@font-face** permite llamar a una sola fuente. Si quisiéramos usar más de una tipografía especial, deberíamos volver a definir la propiedad **@font-face**, llamando a la nueva ruta.

Ahora nos falta asignar esta fuente especial a un elemento de HTML. Para ello, solo tenemos que llamarla normalmente mediante la propiedad **font-family**, usando el nombre que declaramos antes mediante la propiedad **@font-face**:

```
h1 {font-family:'miFuente'}
```

Bastante fácil, ¿no? Veamos ahora cómo hacer que nuestro código sea compatible para todos los navegadores:

```
@font-face {
font-family: 'miFuente';
src: url('webfont.eot'); /* IE9 */
src: url('webfont.eot?#iefix') format('embedded-opentype'), /* IE6-IE8 */
url('webfont.woff') format('woff'), /* Navegadores modernos */
url('webfont.ttf') format('truetype'), /* Safari, Android, iOS */
url('webfont.svg#svgFontName') format('svg'); /* iOSantiguos */
}
```

En este ejemplo podemos ver que se agregan, mediante una segunda propiedad **src**, todos los posibles formatos tipográficos, separados por coma. También, que es necesario utilizar la propiedad **format**, declarando el formato tipográfico para que el navegador pueda comprenderlo correctamente. En este ejemplo,

además, podemos ver algunos hacks utilizados para mejorar la compatibilidad con algunos navegadores.

Uso de fuentes locales

Es importante entender que, usando la propiedad **@font-face**, el navegador siempre descargará la tipografía, incluso si el usuario la tuviera instalada en su computadora. En ese caso, estaríamos forzando a algunos usuarios a realizar una descarga totalmente innecesaria. Para mejorar este aspecto, podemos agregar una línea más de código para indicar al navegador que verifique la existencia de la tipografía antes de descargar el archivo y que, de ser así, no realice la descarga.

```
@font-face {
font-family: 'miFuente';
src: local(Gentium), /* utilizar la tipografía local */
url(Gentium.ttf); /* de lo contrario, descargar la tipografia */
}
```

Podemos ver, en este ejemplo, que se agrega la propiedad **local()**, cuya función es indicar el nombre local de la tipografía para que el navegador verifique su existencia antes de proceder con la descarga del archivo.

Obtener tipografías

Es bastante común que no contemos con todos los formatos de fuente necesarios para cada uno de los navegadores, por lo que debemos recurrir a un generador de tipografías online. Uno de los más usados y conocidos es **Font Squirrel** (**www.fontsquirrel.com**), que cuenta con un amplio catálogo de fuentes gratuitas listas para ser usadas, con todos los formatos tipográficos necesarios y el código de ejemplo para copiar y pegar en nuestro archivo de estilo.

Este sitio también nos brinda un generador de tipografías gratuito, mediante el cual podremos convertir nuestro archivo de tipografías a los distintos formatos para la Web. Es importante recordar que, si la fuente es un archivo licenciado, debemos contar con la licencia de uso para poder utilizar el generador de tipografías.

Figura 1. **Font Squirrel** permite descargar tipografías y convertir nuestras tipografías a otros formatos para la Web.

Otra posibilidad, más fácil aún, es utilizar el catálogo tipográfico gratuito **Google Web Fonts** (**www.google.com/fonts**), que nos provee de una gran cantidad de tipografías de uso libre y muy buena calidad.

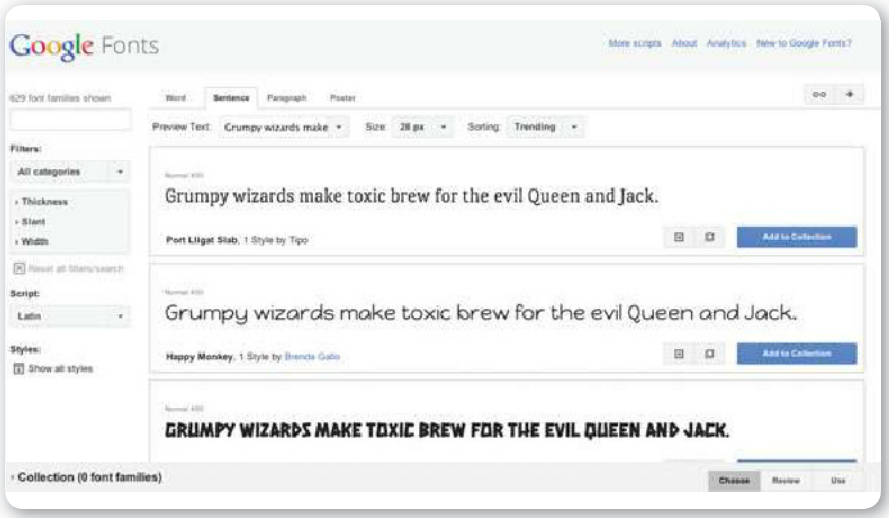


Figura 2. Catálogo tipográfico gratuito de Google.

El código fuente que obtendremos usando la API de Google es exactamente el mismo que acabamos de ver con la propiedad **@font-face**. Google ofrece varias alternativas para hacer uso de su API, las cuales están claramente explicadas en el sitio web.

La opción más básica es llamar a la API usando la etiqueta **link** en el encabezado del documento HTML. Veamos el ejemplo con la tipografía **New Rocker** de Google Web Fonts:

```
<link href='http://fonts.googleapis.com/css?family=New+Rocker' rel='stylesheet'
      type='text/css'>
```

También podemos importar la hoja de estilo desde el archivo de estilo de nuestra página web:

```
@importurl(http://fonts.googleapis.com/css?family=New+Rocker);
```

Una tercera opción consiste en llamar a la tipografía utilizando JavaScript:

```
<script type="text/javascript">
  WebFontConfig = {
    google: { families: [ 'New+Rocker::latin' ] }
  };
  (function() {
    varwf = document.createElement('script');
    wf.src = ('https:' == document.location.protocol ?'https' : 'http') +
      '://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js';
    wf.type = 'text/javascript';
    wf.async = 'true';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(wf, s);
  })(); </script>
```

Para conocer funciones más avanzadas sobre el uso de las tipografías de Google, es recomendable leer detenidamente la información disponible en el sitio web. Al igual que el catálogo tipográfico de Google, existen muchas otras alternativas y empresas que nos proveen de servicios similares: la mayoría de ellos son servicios pagos que vale la pena investigar antes de utilizarlos.

Transformaciones

Una de las nuevas capacidades de CSS3 es la posibilidad de aplicar efectos 2D y 3D a los elementos de HTML mediante la propiedad **transform**, que nos abre un nuevo universo de posibilidades, ya que permite crear diversos efectos que antes necesariamente debíamos crear usando imágenes o JavaScript.

La propiedad **transform** se utiliza comúnmente junto a las de **transition** y **animation,** dos nuevas propiedades de CSS3 que veremos más adelante en este capítulo.

Esta propiedad posee una sintaxis un tanto diferente a las que estamos acostumbrados, ya que recibe como valor un método, que a su vez recibe un parámetro. Los métodos que admite son:

LISTADO DE MÉTODOS 2D Y 3D

TRANSFORMACIÓN 2D	TRANSFORMACIÓN 3D
*rotate	*rotateX
*translate	*rotateY
*translatex	*rotateZ
*translatey	*translateZ
*scale	*scaleZ
*scalex	*translate3d
*scaley	*scale3d
*skew	*rotate3d
*skewx	*perspective
*skewy	
*matrix	

Tabla 1. Métodos para transformaciones en dos y tres dimensiones.

Rotate

El método **rotate** sirve para girar en 2D un elemento la cantidad de grados que deseemos. Se utiliza de la siguiente manera:

```
img { transform: rotate(45deg); }
```

En este caso, la imagen rota 45 grados en sentido horario. Si queremos que el elemento rote en sentido opuesto, entonces debemos usar un valor negativo.

Figura 3. Imagen rotada mediante la propiedad **`transform`** y el método **`rotate`**.

RotateX y rotateY

Estos métodos permiten rotar un elemento en 3D en los ejes **X** (horizontal) e **Y** (vertical), respectivamente.

La sintaxis es igual a la utilizada para el método **rotate**:

```
img { transform: rotateX(180deg); }
img { transform: rotateY(180deg); }
```

COMPATIBILIDAD CON INTERNET EXPLORER

IE es el navegador con menos soporte para esta propiedades y métodos; recién en su versión 10 incorpora soporte para algunas de estas capacidades de CSS3 y, en su versión 11, mejora la compatibilidad.

Estos métodos permiten crear efectos de espejo para los elementos, y si los combinamos con las propiedades de animación y transición, podemos crear efectos más que interesantes.

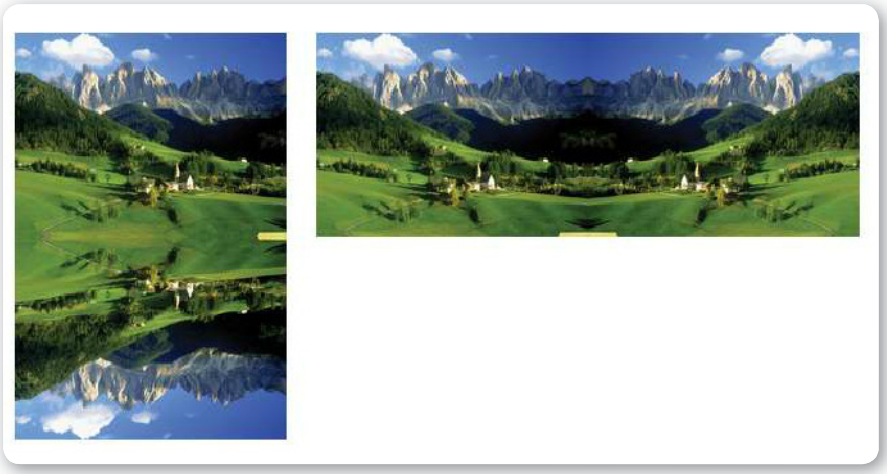


Figura 4. Efectos creados mediante la propiedad **`transform`** y los métodos **`rotateX`** y **`rotateY`**.

Scale, scalex y scaley

El método **scale** se utiliza para cambiar el ancho y el alto de un elemento a una proporción mayor o menor al tamaño real del elemento, es decir, **escalar**. Veamos su sintaxis:

```
img { transform: scalex(2); }
```

En este ejemplo, el ancho de la imagen se duplicará, ya que el método **scalex** permite manipular el ancho del elemento.

```
img { transform: scaley(2); }
```

En este caso, el alto de la imagen se duplicará, ya que el método **scaley** permite manipular el alto del elemento.

```
img { transform: scale(2); }
```

Aquí se duplicará tanto el ancho como el alto de la imagen. Mediante el método **scale** podemos manipular el ancho y el alto con el mismo valor; o bien, podemos controlarlos de forma independiente separando los valores con una coma, como se puede observar a continuación:

```
img { transform: scale(2,3); }
```

En este caso, se duplicará el ancho de la imagen, mientras que su alto se triplicará. Los métodos de escala también pueden utilizar valores decimales:

```
img { transform: scale(2.3 , 0.5); }
```

Figura 5. Imagen con su tamaño original y con una escala aplicada de 1.5, mediante la propiedad **`transform`** y el método **`scale`**.

Skew, skewx y skewy

Estos métodos permiten inclinar el elemento, en los ejes **X** e **Y**, la cantidad de grados que creamos necesaria. Vemos los ejemplos:

```
img { transform: skewx(45deg); }
```

En este caso, la imagen se inclinará 45 grados en el eje **X.**

```
img { transform: skewy(45deg); }
```

Con este código, la imagen se inclinará 45 grados en el eje **Y.**

```
img { transform: skew(45deg); }
```

Con el método **skew**, podemos controlar ambos lados con el mismo valor, o bien de forma independiente usando una coma para separar los valores.

```
img { transform: skew(45deg , 65deg); }
```

Figura 6. Imagen inclinada mediante el método **skew**.

Translate, translatex y translatey

Los métodos **translate**, **translatex** y **translatey** permiten trasladar un elemento de su posición original, logrando un efecto similar al que conseguimos usando la propiedad **position**. Veamos su sintaxis:

```
img { transform: translate(200px , 100px); }
```

En este caso, la imagen se desplazará 200 px a la derecha y 100 px hacia abajo de su posición original. Este método es muy útil cuando lo combinamos con animaciones y transiciones.

```
img { transform: translatex(200px); }
```

En este ejemplo, la imagen se desplaza 200 px hacia la derecha, ya que este método permite controlar el desplazamiento horizontal.

```
img { transform: translateY(50px); }
```

Aquí la imagen se desplaza 50 px hacia la abajo, ya que este método permite controlar el desplazamiento vertical.

Sumar transformaciones

Existen dos maneras de aplicar más de una transformación al mismo elemento. La primera consiste en agregar más de un método y separarlo por espacios, como vemos a continuación:

```
img { transform: rotate(50deg) scale(2)  skew(20deg); }
```

En este caso, hemos aplicado tres transformaciones simultáneas al mismo elemento, pero también podemos conseguir el mismo efecto usando el método **matrix** que veremos a continuación.

Matrix

El método **matrix** permite combinar todos los métodos de transformación 2D en una sola línea de código. Este método usa seis parámetros que utilizan funciones matemáticas mediante las cuales se puede rotar, escalar, inclinar y mover el elemento. La sintaxis es la siguiente:

```
img { transform:matrix(0.866,0.5,-0.5,0.866,0,0);}
```

Los valores para este método son un tanto complejos de calcular, pero afortunadamente podemos encontrar en la Web algunas herramientas de uso libre que nos permiten simplificar esta tarea. Una de ellas es **http://peterned.home.xs4all.nl/matrices**, que ofrece la posibilidad de copiar el código y pegarlo en nuestra hoja de estilo.

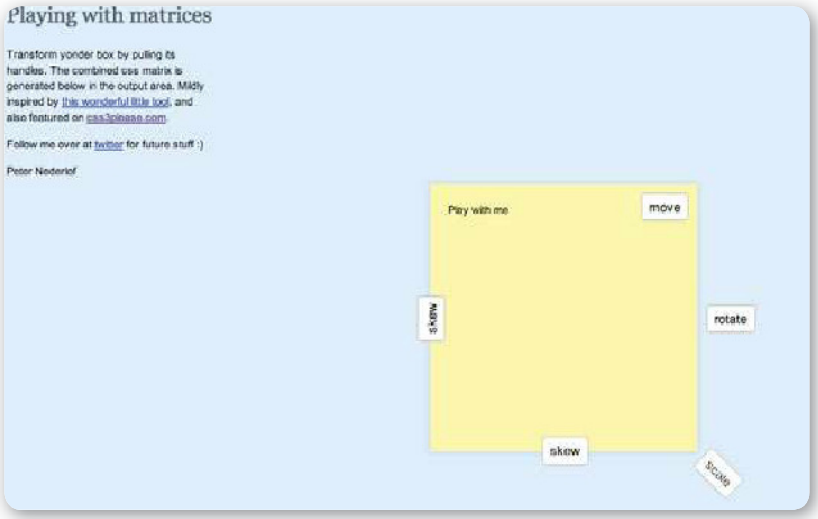


Figura 7. Generador de matrices online.

Transform-origin

Esta propiedad permite modificar el punto de origen de la transformación. Como habrán notado, todas las transformaciones se realizan desde el centro del elemento, pero en algunos casos ese punto de origen puede no ser útil. Con esta propiedad podemos modificar ese punto por otro que necesitemos. Veamos su sintaxis:

```
img { transform: rotate(50deg);
      transform-origin:top left;
}
```

De esta forma, la rotación se realiza desde el vértice superior izquierdo. Los valores posibles son:

- **left**
- **center**
- **right**

- **top**
- **bottom**
- valor en porcentaje %
- valor numérico

Transiciones

Una de las novedades que presenta CSS3 es la de crear transiciones entre reglas de estilos para lograr cambios elegantes cuando el navegador pasa de una regla de estilo a otra por efecto de una acción del usuario. Mediante la propiedad **transition** se pueden lograr efectos de interfaz muy estéticos y funcionales. Comencemos con un ejemplo simple: supongamos que tenemos un título y queremos que el texto cambie de color al pasarle por arriba el cursor. Veamos cómo sería el código:

```
h1{color:red}
h1:hover{color:white}
```

Así, al pasar por arriba del texto del **h1**, el color cambiará, de forma abrupta, de rojo a blanco. Ahora vayamos un paso más allá: podemos usar la propiedad **transition** para lograr que el texto pase de rojo a blanco de una forma elegante, estética y con distintos efectos que podemos configurar. Veamos qué cambios podemos hacer en nuestro código CSS:

```
h1{color:red; transition:2s color;}
h1:hover{color:white}
```

En este ejemplo, el texto del **h1** sufrirá una transición de 2 segundos para realizar el cambio de color de texto. La propiedad **transition** es el shorthand del siguiente grupo de propiedades:

- **transition-property**: especifica una o más propiedades a las que se les aplicará la transición. Su valor puede ser cualquier propiedad de CSS o bien más de una.

- **transition-duration**: determina el tiempo durante el cual se producirá la transición. Su valor se determina en segundos o décimas de segundo.
- **transition-timing-function**: indica el tipo de animación que usará la transición. Los posibles valores son: **ease**, **ease-in**, **ease-in-out**, **ease-out**, **linear** y **cubic-bezier.**
- **transition-delay**: establece el tiempo de retraso (delay) que tendrá la transición. Su valor se establece en segundos o décimas de segundo.

Todas estas propiedades pueden ser simplificadas en una línea, de la siguiente forma:

```
h1{transition:2s color linear 1s;}
```

Si queremos aplicar la transición a más de una propiedad, tenemos dos posibilidades: declarar cada una de las propiedades y sus valores separados por comas, o bien utilizar el valor **all**, para aplicar los mismos valores a todas las propiedades. Veamos ambos ejemplos:

```
h1{transition:2s color, background 1s, border 3s;}
```

En este caso, se aplicará la transición para cada propiedad en el tiempo especificado para cada una. Un caso bastante común es usar el mismo tiempo para todas las propiedades, lo que se aconseja realizar de la siguiente manera:

```
h1{ transition:          all 2s;
                         color:red;}

h1:hover {               color:blue;
                         border:1px solid red;
                         background:3px solid red;}
```

De esta forma, la transición se aplicará a todas las propiedades que estén en la regla de estilo para el estado **hover** del elemento.

Animaciones

Una de las capacidades más famosas de CSS3 es la posibilidad de crear animaciones usando solamente código CSS. Estas animaciones son realmente fáciles de crear, y lo más importante es que son reutilizables, ya que podemos definir una y aplicársela a tantos elementos como queramos, sin necesidad de volver a declarar la animación. De hecho, los parámetros de la animación no se establecen en la animación en sí, sino en la asignación de esta al elemento del HTML. Esta característica las vuelve aún más reutilizables, ya que podemos tener la misma animación aplicada a distintos elementos de HTML y que cada uno de ellos utilice distintos parámetros.

A diferencia de las transiciones, las animaciones no necesitan un evento que las ejecute, sino que se activan al cargar la página. De igual modo, también podemos decidir que estén pausadas y se activen mediante un evento del usuario.

La propiedad que utilizaremos para crear una animación es **@keyframes** y luego **animation** para asignarle esta animación a un elemento de HTML. A continuación, crearemos una animación simple de un texto que cambie de color:

```
@keyframesnombredeanimacion {

        /*reglas de estilo que componen la animación*/

}
```

El selector se compone por el atributo **@keyframes**, que indica que crearemos una animación. Acto seguido, debemos declarar un nombre de animación. Podemos crear tantas animaciones como nos resulte necesario; cada una de ella deberá tener la misma estructura.

Dentro de las llaves de esta regla de estilo debemos anidar las reglas de estilo de cada uno de los cuadros de nuestra animación. Veamos cómo hacer una animación de dos pasos:

```
@keyframestitulo {
```

```
        from {color:red;}
        to {color: grey;}
}
```

En este ejemplo, elegimos una animación que tiene dos estados, **inicial** y **final**, lo mínimo necesario para poder crearla. En este caso, usamos los selectores **from** y **to** para indicar el estado inicial y final de la animación.

Si queremos crear una animación con más pasos, tendremos que usar selectores numéricos en lugar de **from** y **to**. Veamos cómo sería la sintaxis:

```
@keyframestitulo {
        0% {color:red;}
        30% {color: grey;}
        50% {color: blue;}
        70% {color: green;}
        100% {color: white;}
}
```

En este caso, tenemos una animación que posee cinco estados. Podemos ir variando el valor del selector en función de la duración que deseamos para la animación. Es importante saber que el valor del selector representa un porcentaje del tiempo total de la animación, que asignaremos luego en la regla de estilo del elemento de HTML.

Ahora bien, esta animación no será visible en tanto no la asignemos a un elemento de HTML. Para ello, necesitamos usar la propiedad **animation** en la regla de estilo del elemento. Veamos cómo debe ser la sintaxis para aplicar esta regla de estilo a un elemento **h1**:

```
h1{
animation-name: titulo;
animation-duration: 5s;
animation-timing-function: linear;
animation-delay: 3s;
animation-iteration-count: infinite;
```

```
animation-direction: alternate;
animation-play-state: running; }
```

También podemos utilizar el shorthand para reducir las líneas de código:

```
h1{ animation: titulo 5s linear 3s infinite alternate running; }
```

Veamos una a una las posibles propiedades y sus valores:

- **animation-name**: define el nombre de la animación. Admite cualquier valor alfanumérico.
- **animation-duration**: determina la duración total de la animación. Este valor debe ser definido en segundos o milésimas de segundos.
- **animation-timing-function**: especifica el tipo de animación; los posibles valores son: **ease**, **ease-in**, **ease-in-out**, **ease-out**, **linear** y **cubic-bezier.**
- **animation-delay**: determina el tiempo de espera para comenzar la animación. Su valor se determina en segundos; si la animación es infinita, esta espera se aplicará solo en el primer ciclo de la animación.
- **animation-iteration-count**: establece si la animación es infinita o no. El valor posible es: **infinite.**
- **animation-direction**: define si la animación se ejecuta de forma alternada o siempre en la misma dirección. El valor posible es: **alternate**. Esta propiedad solo se utiliza si la animación es infinita.
- **animation-play-state**: especifica si la animación está andando o en pausa. Los valores posibles son: **running** y **paused.**

Como ya habíamos mencionado, las animaciones son reutilizables. Esto significa que podemos crear una animación y usarla en múltiples elementos y, lo que es mejor aún, cada elemento podrá usar distintos parámetros para la misma animación. Veamos un ejemplo usando una animación que creamos anteriormente:

```
h1{ animation: titulo 5s linear 3s infinite alternate running; }
h5{ animation: titulo 2s ease-in}
```

Usar la API para Drag&Drop

El concepto de **arrastrar** (*drag*) y **soltar** (*drop*) está en nuestras mentes y en nuestra forma habitual de utilizar la computadora. Pero ¿cómo llevamos esta característica a la Web?

Actualmente hay elementos que, al arrastrarlos y soltarlos, tienen ya comportamientos predefinidos. Por ejemplo, si posicionamos el cursor del mouse sobre una imagen, lo mantenemos presionado y lo arrastramos a una nueva pestaña del navegador, esta abrirá la nueva pestaña con la imagen como URL.

Ahora supongamos que queremos, dentro de la misma página, armar un listado de ítems y una papelera, y que cada vez que arrastremos un ítem a la papelera, este último se elimine de la pantalla, logrando un efecto impresionante y brindando al usuario la sensación de estar utilizando una aplicación de escritorio.

Para poder realizar esto vamos a necesitar una imagen de una papelera y una lista desordenada (**<ul>**). Combinando estos elementos con un poco de JavaScript lograremos el efecto deseado.

Antes de empezar

Debemos definir qué elementos serán "arrastrables" (**draggable**) y qué elementos serán encargados de recibir (**droppable**) los ítems. Para hacer esto, en HTML5 disponemos de ciertos eventos y atributos.

Para que un elemento sea "arrastrable" debe tener el atributo **draggable** con el valor **true** y debe escuchar los siguientes eventos: **drag** y **dragstart**. Luego disponemos de los siguientes eventos para controlar la acción de arrastrar y soltar: **dragenter**, **dragleave**, **dragover**, **drop** y **dragend**. Veamos una descripción de cada evento:

EXPERIENCIA DE USUARIO ENRIQUECIDA

Con las novedades de CSS3 y las nuevas APIs de HTML5 podremos lograr que la experiencia del usuario al utilizar un sitio web sea igual o incluso más atractiva que al usar una aplicación de escritorio. De hecho, es posible conseguir efectos similares a los que hallamos en programas, como barras de menús y los efectos de arrastrar y soltar elementos.

LISTADO DE EVENTOS

EVENTO	DESCRIPCIÓN
Dragstart	Ocurre cuando el usuario se posiciona sobre el elemento, hace clic y comienza la acción de arrastrar.
Dragenter	Ocurre cuando el elemento arrastrado se posiciona por primera vez sobre otro elemento. Debemos escuchar este evente si queremos indicar visualmente que se puede soltar un elemento en ese lugar.
Dragover	Se dispara constantemente cuando el elemento está sobre otro elemento.
Dragleave	Se dispara cuando el cursor sale de un elemento cuando está ocurriendo la acción de arrastrar. En este evento deberíamos quitar todas las propiedades del elemento que pusimos para indicar que se podían soltar elementos en él.
Drag	Es el evento que ocurre constantemente cuando el usuario está arrastrando el elemento.
Drop	Se dispara cuando el usuario suelta un elemento sobre otro que a su vez permite recibir otros elementos (droppable).
Dragend	Se produce cuando el usuario suelta el botón del mouse para dejar un elemento.

Tabla 2. Descripción de eventos de la API **Drag & Drop**.

Entonces, una secuencia normal de eventos para el arrastre de un elemento es: **Dragstart** -> **Drag** -> **Dragenter** -> **Dragover** -> **Dragend** -> **Drop.** Veamos nuestro ejemplo:

```
<!doctype html>
<html>
<head>
</head>
<style>
        section {
```

```
                width:200px;
                float:left;

        }
        #listado {
                list-style:none;
        }

</style>
<body>
<section>
        <ul id="listado">
                <li draggable="true" class="item">Item 1
                <li draggable="true" class="item">Item 2
                <li draggable="true" class="item">Item 3
        </ul>
</section>
<section id="cesta">
        <img src="img/vacio.png"/>
</section>

</body>
<script type="text/javascript">
//Aquí pondremos nuestro código que escuchará los eventos de arrastre
</script>
</html>
```

DRAG & DROP: EVENTOS BÁSICOS

Para utilizar la API de Drag&Drop necesitaremos escuchar como mínimo tres eventos. Para empezar, el evento **dragstart**, que se disparará al comenzar la acción. Luego, el evento **dragover**, que se disparará al pasar por el elemento que recibirá el ítem arrastrado. Por último, escucharemos el evento **dragend**, que se ejecutará cuando la acción finalice. Para aprender más acerca del uso del sistema de arrastrar y soltar de HTML5 y ver un ejemplo de uso, es recomendable visitar la página **www.html5demos.com/drag**.

Agregándole el atributo **draggable** con el valor **true** a los elementos **<li>**, hicimos que fueran elementos arrastrables.

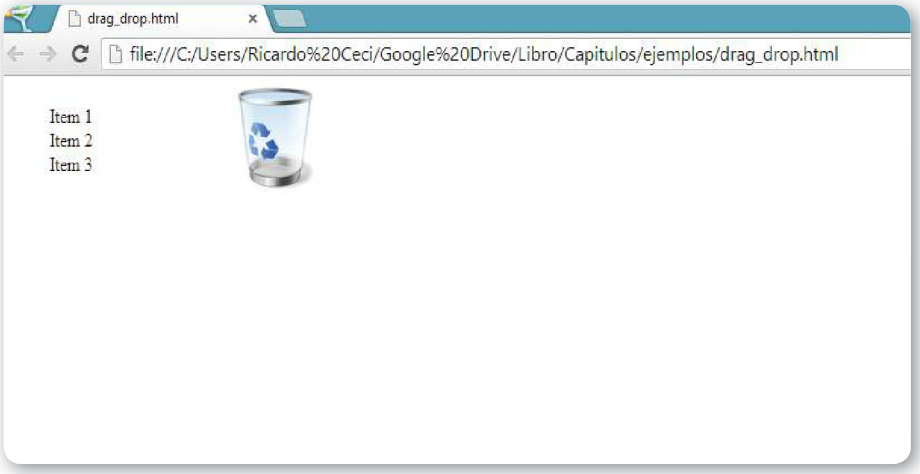


Figura 8. Así se ve nuestro ejercicio.

Repasando la secuencia de eventos, el primer evento que deberíamos escuchar es el evento **dragstart.**

```
<script type="text/javascript">
        window.onload=function(){
                var items= document.querySelectorAll('.item');
                for(i=0;i<items.length;i++){
        items[i].addEventListener('dragstart',function(event){
                        console.log(event.target.innerHTML+" ha comenzado a
arrastrarse");
                this.style.opacity = 0.4;
                        });
                }
}
</script>
```

En el código indicamos que, al cargar la ventana, se busquen todos los elementos con la clase **ítem** y que a cada uno se le asocie un **eventlistener** (mediante **addEventListener**) para que escuche el evento **dragstart**. El evento es pasado como parámetro a la función que se va a ejecutar cuando se dispare el **dragstart.**

Con la propiedad **target** del evento (**event.target**), o bien con **this**, accedemos al elemento HTML sobre el cual se está ejecutando la acción, mientras la propiedad **innerHTML** nos permitirá obtener el contenido HTML del elemento. Entonces, cuando empecemos a arrastrar el elemento, veremos en consola cuál es el ítem que estamos arrastrando y, además, le setearemos al elemento la propiedad **opacity** en 0.4.

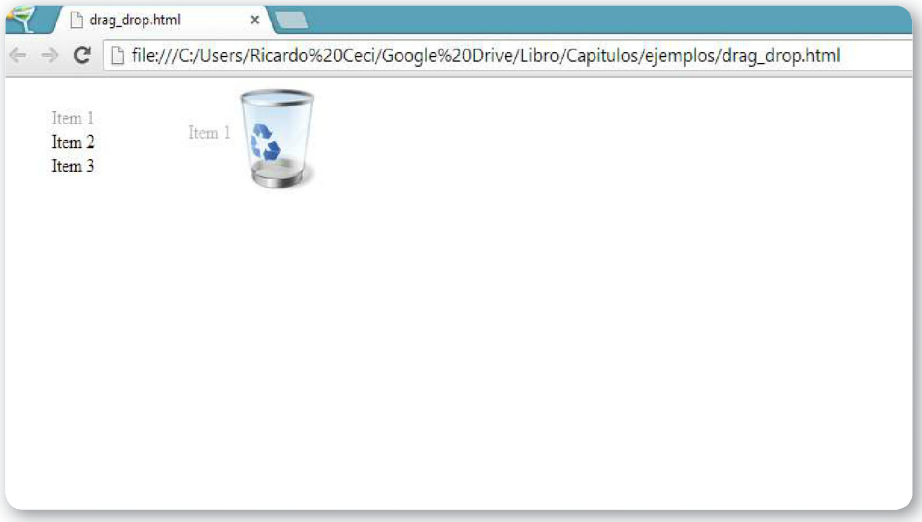


Figura 9. Escuchando el evento **`dragstart`**.

Cuando arrastramos, hay diferentes acciones que podemos realizar con el elemento: copiar (**copy**), mover (**move**) o vincular (**link**). Nosotros podremos definir qué acciones estarán permitidas con el arrastre; esto no quiere decir que la acción se realizará. Supongamos, por ejemplo, que un elemento tiene permitido copiar y algunos elementos que permiten depositar elementos aceptan el efecto "copia" y otros no. Entonces, solo se podrá depositar el elemento en aquellos otros que acepten dicho efecto.

DRAG & DROP ANTES DE HTML5

El efecto de arrastrar y soltar en la Web era, hasta HTML5, una tarea de librerías externas: debíamos utilizar algún plugin de JavaScript, como jQuery UI o Dojo. Estas librerías, mediante simples modificaciones, nos permitían lograr el efecto Drag&Drop a costa de una excesiva sobrecarga del documento y disminuciones de rendimiento que podían ser aprovechadas para otras cosas.

Para configurar el efecto se utiliza la propiedad **evento.dataTransfer.effectAllowed** en el elemento que se arrastra y **evento.dataTransfer.dropEffect** en el elemento que recibe. Por defecto, todos los efectos están permitidos en ambos casos.

```
<script type="text/javascript">
        window.onload=function(){
                var items= document.querySelectorAll('.item');
                for(i=0;i<items.length;i++){
        items[i].addEventListener('dragstart',function(event){
                        console.log(event.target.innerHTML+" ha comenzado a
arrastrarse");
                this.style.opacity = 0.4;
                event.dataTransfer.effectAllowed = 'copyMove'
                        });
                }
}
</script>
```

Siguiendo esta secuencia, podríamos escuchar el evento **drag** para hacer un seguimiento del arrastre o escuchar el evento **dragOver.** Para esto, le pondremos a nuestra papelera de reciclaje un borde que aparecerá cuando un elemento se posicione sobre ella.

```
<style>
        section {
                width:200px;
                float:left;
                        }
        #listado {
                list-style:none;
        }
        .cesta_activa {
                border: 5px dashed red;
                width:100px;
```

```
}
</style>
…
<script type="text/javascript">
window.onload=function(){
                    var items= document.querySelectorAll('.item');
                    for(i=0;i<items.length;i++){
        items[i].addEventListener('dragstart',function(event){
console.log(event.target.innerHTML+" ha comenzado a arrastrarse");
                    this.style.opacity = 0.4;
                    event.dataTransfer.effectAllowed = 'copyMove'
                              });
                    }
        var cesta = document.getElementById('cesta');
        cesta.addEventListener('dragover',function(evento){
                    this.classList.add('cesta_activa');
                    evento.preventDefault();
});
}
</script>
```

Finalmente, con estas líneas de código logramos el siguiente efecto:

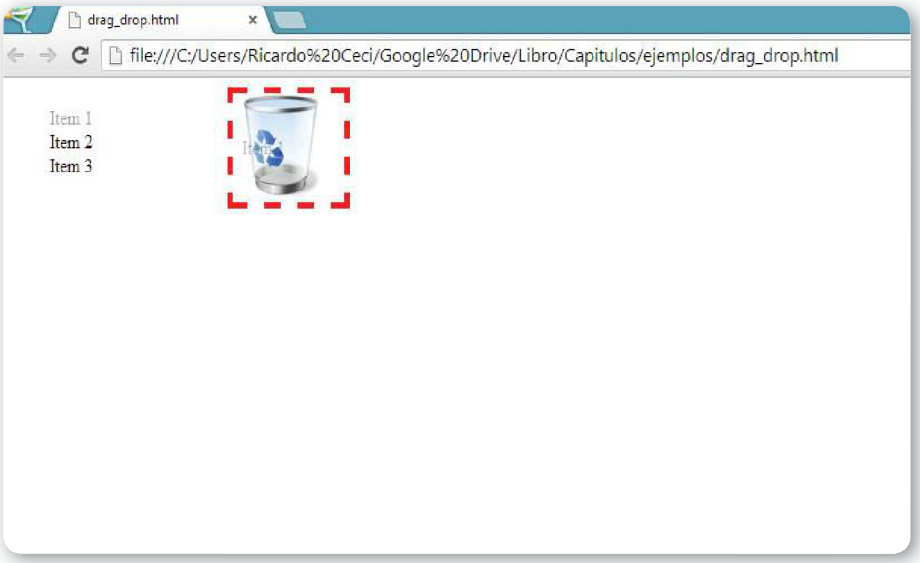


Figura 10. Demostración del efecto **dragover**.

Ahora nos queda recibir el elemento y borrarlo de la lista de la izquierda para pasarlo a la derecha. Para esto, vamos a tener que hacer unas pequeñas modificaciones:

```
…
items[i].addEventListener('dragstart',function(event){
        console.log(event.target.innerHTML+" ha comenzado a arrastrarse");
                this.style.opacity = 0.4;
                event.dataTransfer.effectAllowed = 'copyMove';
        evento.dataTransfer.setData('id',this.id);
});
…
```

Comenzamos a utilizar el objeto **dataTransfer** con funcionalidades avanzadas, ya que se encarga de llevar la información de un elemento draggable a un elemento droppable. Para esto, disponemos de dos métodos: **setData()** y **getData()**. Para el primero, debemos especificar un par clave-valor donde indicaremos el tipo de dato a pasar o, si en la clave no definimos el tipo de dato, podemos utilizarla como referencia para luego recuperarla a través del método **getData()**. Ahora escuchamos el evento **drop**:

```
…
cesta.addEventListener('drop',function(evento){
                var id = evento.dataTransfer.getData('id');
                var item = document.getElementById(id);
                item.parentNode.removeChild(item);
                this.innerHTML = '<img src="img/lleno.jpg">';
                this.classList.remove('cesta_activa');
        });
…
```

Con el método **getData** obtenemos lo que haya en la clave **ID**, que en este caso es el ID del elemento que se está arrastrando. Luego, mediante el método **getElementById**, obtenemos el elemento, lo guardamos en la variable **item** y con **item.parentNode.removeChild(item)** lo eliminamos.

Cambiamos la imagen de la cesta vacía por la cesta llena con **this.inerHTML = '<img src="img/lleno.jpg">'**. Y con **this.classList.remove('cesta_activa')** le quitamos el borde rojo a la cesta.

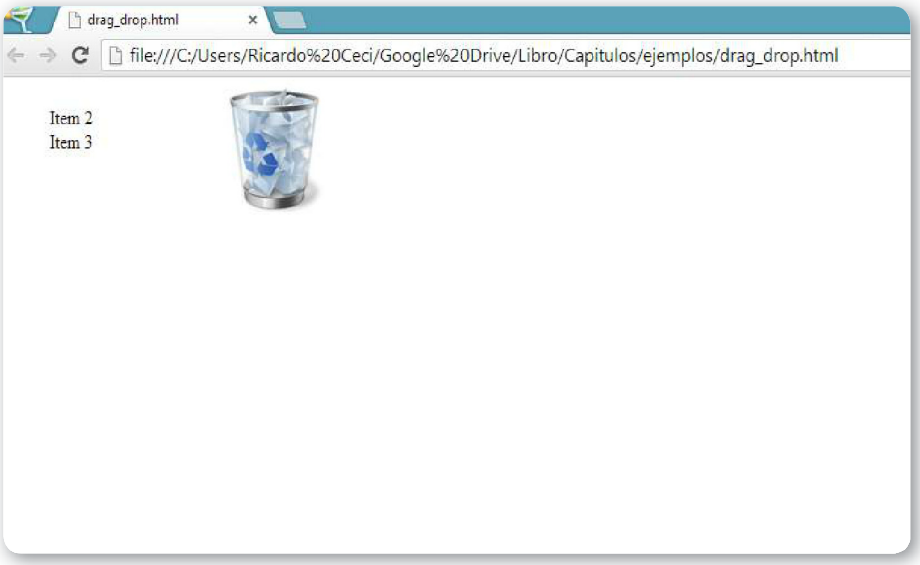


Figura 11. Demostración del efecto logrado.

Ahora bien, todo funcionará perfecto siempre y cuando el usuario seleccione un ítem, lo arrastre sobre la cesta y lo deje ahí. Supongamos que el usuario arrastra el ítem, pasa sobre la cesta, pero lo deja en cualquier otro lado no permitido; observemos, entonces, qué sucede:

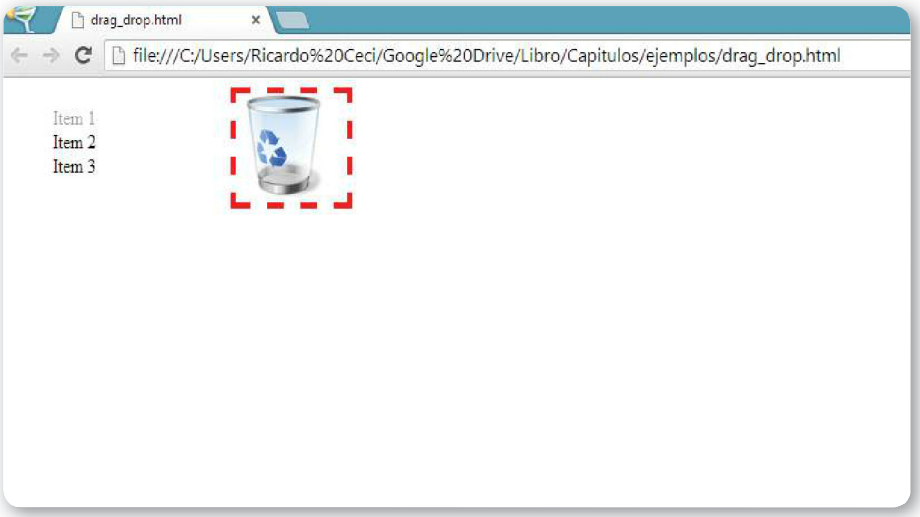


Figura 12. Si el usuario arrastra y deja en otro lugar el elemento, nuestro efecto queda inconcluso.

Lo que observamos es que, si bien se dispararon todos los eventos, el usuario no completó la acción, motivo por el cual el efecto quedó a la mitad. Para evitar esto, debemos escuchar los eventos **dragleave** y **dragend** y, con un pequeño cambio en nuestro código, podremos apuntar ambos efectos a la misma función. A continuación, veremos cómo hacerlo:

```
window.onload = function() {
…
for(i=0;i<items.length;i++){
…
        items[i].addEventListener('dragend',limpiar);
}

cesta.addEventListener('dragleave',limpiar);

}

var limpiar = function(evento){
        evento.target.style.opacity = 1;
        evento.classList.remove('cesta_activa');
        evento.preventDefault();
}
```

> DEBEMOS CONTROLAR TODOS LOS MOVIMIENTOS QUE PUEDA LLEGAR A HACER EL USUARIO

De esta manera lograremos que, en caso de que el usuario no deje el elemento donde nosotros queríamos, vuelva la propiedad **opacity** a 1 y quite el recuadro **cesta_activa**, dejando todo tal como estaba cuando se inició la operación.

Cuando llamamos al método **preventDefault()**, estamos indicándole al navegador que evite realizar cualquier acción por defecto o comportamiento ya establecido y que realice únicamente lo que nosotros queremos. Esto va a servir, por ejemplo, cuando arrastremos imágenes, para que el navegador no abra una nueva pestaña con la imagen. Este comportamiento ya lo anallizamos al principio de este apartado.

Eventos touch

Con el inminente arribo de los dispositivos móviles y las pantallas táctiles al mundo de la Web, tenemos una posibilidad adicional para mejorar la experiencia: permitirle al usuario "tocar" nuestro sitio.

Hasta el momento, la Web solo soportaba clics en vínculos o en botones, pero a fin de cuentas el evento que se escuchaba era el clic. HTML5 nos trae un conjunto de **eventos touch**, que se disparan solamente en dispositivos que cuenten con pantallas táctiles.
Los nuevos eventos que incorpora la API de HTML5 son:

LISTADO DE EVENTOS TOUCH

EVENTO	DESCRIPCIÓN
Touchstart	Ocurre cuando se apoya un dedo sobre un elemento del DOM.
Touchmove	Se arrastra un dedo a lo largo de un elemento del DOM.
Touchend	Se quita el dedo.

Tabla 3. Listado de eventos touch.

Cada evento touch incorpora tres propiedades:

- **Touches**: es una lista de todos los dedos con los que se está tocando la pantalla.
- **targetTouches**: es una lista de los dedos que están sobre el elemento del DOM.
- **changedTouches**: indica qué dedos cambiaron al ejecutarse el evento.

TOCAR NO ES LO MISMO QUE HACER CLIC

Al arribar los dispositivos móviles como herramientas para navegar en internet, se agregaron nuevos eventos que permiten controlar lo que realmente hacen los usuarios con estos equipos, ya que, en lugar de hacer clic, los tocan. En **http://mobiforge.com/design-development/html5-mobile-web-touch-events** se propone un ejemplo simple para comprender su uso.

¿Cómo probar los eventos touch en la computadora?

Para probar estos ejercicios debemos probar nuestro sitio en un dispositivo móvil, o bien activar los eventos touch de la consola del desarrollador de Google Chrome. Para esto, pulsamos la tecla **F12** en el navegador, para que se abra la consola, y pulsamos el botón representado con una tuerca.

Luego, tildamos la opción **Show 'Emulation' view in console drawer**; posteriormente, presionamos la tecla **ESC** para volver a la consola y presionamos la pestaña **Emulation**. En el menú desplegable, podremos elegir un teléfono o tablet que emulará el browser.

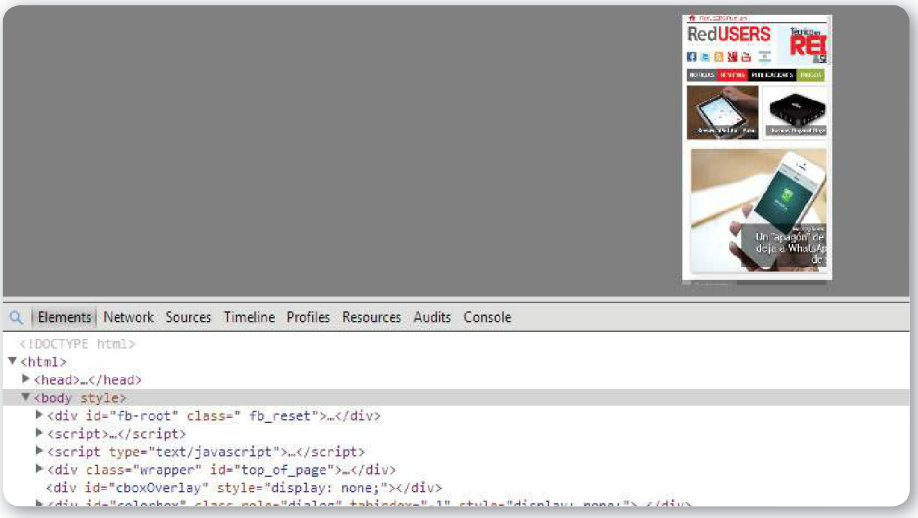


Figura 13. Configuración de la consola de desarrollador.

Mover elementos

Vamos a realizar un ejercicio que moverá un elemento del DOM siguiendo nuestro dedo. Primero escucharemos el evento **touchstart** para identificar cuando el usuario toca nuestro elemento, e indicarle que comience a arrastrar.

```
<style>
        #miCaja {
                width:150px;
                height:150px;
```

```
                background-color:orange;
                position:absolute;
                }
</style>
…
<div id="miCaja">Toque aquí</div>
…
<script>
        var miCaja = document.getElementById('miCaja');
        miCaja.addEventListener('touchstart',function(e)
        {
                this.innerHTML = 'Ahora arrastre...';
        });
</script>
```

Luego, podemos escuchar el evento de finalización de toque **touchend**, para identificar cuando el usuario levanta el dedo, y volver a su valor inicial el valor de nuestra caja.

```
miCaja.addEventListener('touchend',function(e){
                this.innerHTML = 'Toque aquí';
        });
```

Cuando el cliente levante el dedo, veremos que el contenido de nuestra caja vuelve al estado inicial.

HERRAMIENTAS DEL DESARROLLADOR

La consola del desarrollador de Google Chrome no solo nos va a permitir inspeccionar los elementos HTML que se encuentran en nuestra página, sino que, además, nos da la posibilidad de emular ciertos navegadores de dispositivos móviles y los eventos que pueden ser ejecutados en ellos, como por ejemplo los eventos táctiles. Esto nos permitirá una prueba preliminar en pantallas de tablets y móviles antes de abrir el sitio web en uno de estos equipos.

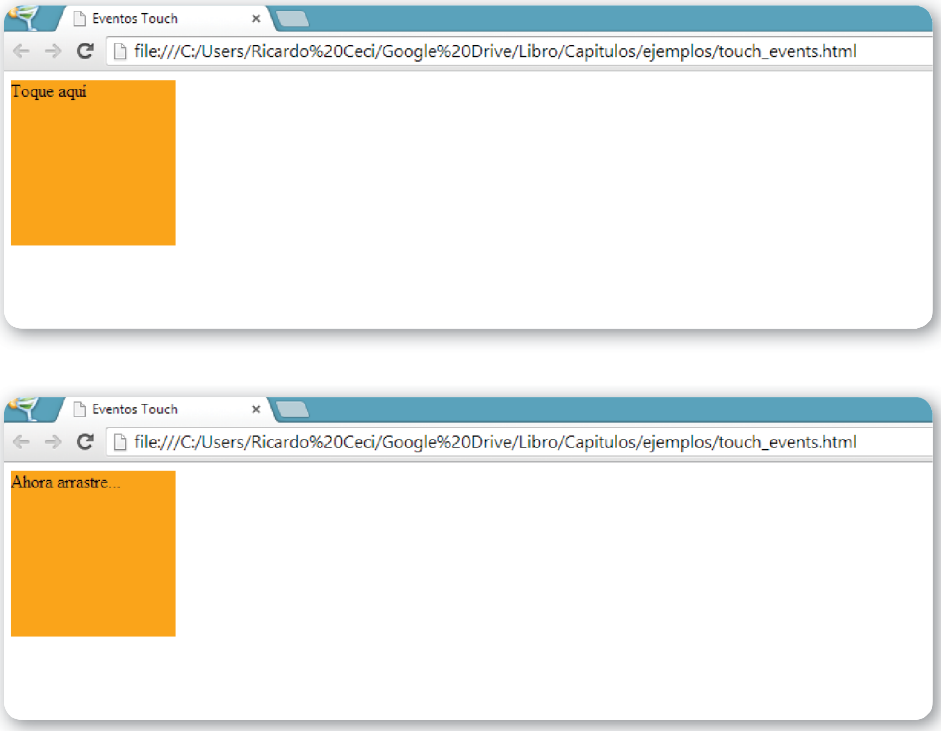


Figura 14. Al tocar el elemento cambiamos el texto; si levantamos el dedo, volvemos al texto original.

Ahora solo nos queda hacer el seguimiento del movimiento para desplazar el elemento en pantalla.

```
miCaja.addEventListener('touchmove',function(e){
        e.preventDefault();
        this.innerHTML = 'Desplazando...';
                if(e.targetTouches.length == 1){
                        var touch = event.targetTouches[0];
                        this.style.left = touch.pageX + 'px';
                        this.style.top = touch.pageY + 'px';
                }

        });
```

En este caso, con el comando **e.preventDefault()** prevenimos el comportamiento por defecto que el dispositivo le haya asignado al gesto o movimiento que hagamos con el dedo para que realice

nuestra acción. Luego contamos la cantidad de dedos que tocaron el elemento (**targetTouches.length**): en caso de ser 1, guardamos las propiedades del toque en una variable llamada **touch** y continuamos. De esta manera, podemos saber con cuántos dedos tocó el usuario la pantalla. Después, solo queda modificar las propiedades **left** y **top** de nuestro elemento con los valores **pageX** y **pageY** del toque realizado.

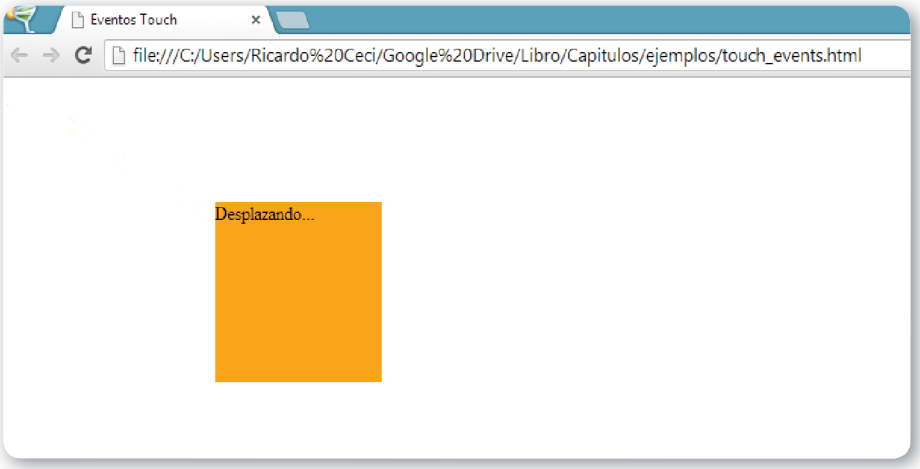


Figura 15. Desplazamiento de nuestro elemento.

RESUMEN

En este capítulo aprendimos a mejorar la calidad visual de nuestros diseños, utilizando tipografías especiales en la Web, y cómo crear efectos de interfaz mediante el uso de las propiedades **transition**, **animation** y **transform**. Además, trabajamos con dos APIs de JavaScript para mover elementos en pantalla, utilizamos la API de Drag&Drop para definir fácilmente elementos que podrán ser arrastrados y elementos que actuarán como receptores de otros. Luego, aprovechamos las nuevas características de los dispositivos móviles y táctiles y aprendimos a trabajar con los eventos touch para identificar los distintos toques del usuario.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son los valores que debemos usar si queremos crear una animación que nunca termine y que su ciclo se muestre de forma alternada?
2. ¿Cuáles son los formatos de tipografías necesarios para cada uno de los navegadores?
3. Explique para qué se utiliza el método **matrix**.
4. ¿Cuál es la propiedad y el método que permite rotar una imagen 45 grados?
5. ¿Qué propiedad debemos agregarle a un elemento HTML para que sea "arrastrable"? ¿Qué valor debemos asignarle?
6. Nombre tres eventos que debemos controlar en JavaScript para una acción de **Drag & Drop**.
7. ¿En qué momento se dispara el evento **touchstart**?
8. ¿De qué manera evitamos el comportamiento por defecto de un evento?
9. ¿Cómo comprobamos la cantidad de dedos con los que un usuario tocó un elemento?

EJERCICIOS PRÁCTICOS

1. Descargue el archivo **tp1.zip**.
2. Arme una página donde haya cuatro productos y, al arrastrarlos a un carrito de compras, este vaya listando debajo de él los productos agregados.
3. En una página simple, cambie el color de fondo de acuerdo con la cantidad de dedos con los que se presiona en pantalla (deberá probarlo en un teléfono o tablet).

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*10

Diseño adaptable con media queries

En este capítulo veremos los conceptos fundamentales del diseño adaptable y cuáles son sus principales características. También aprenderemos en detalle qué son y para qué se utilizan las media queries y cómo implementarlas en diseños adaptables.

- **El diseño adaptable.....304**
- **El mundo de las media queries.....305**
  - Antepasados de las media queries..305
  - Anatomía de las media queries.....306
  - Posibilidades de las media queries.307
- **El atributo viewport.....311**
  - Valores posibles de viewport.....312
  - El futuro de la etiqueta meta viewport.....314
- **Unidades de medida adaptables.....315**
  - El uso de em y rem.....315
- **Resumen.....317**
- **Actividades.....318**

El diseño adaptable

Diseño adaptable o *responsive design* –como fue bautizado originalmente por Ethan Marcotte– es la técnica de diseño y maquetación que se utiliza para crear sitios que se adapten a la pantalla con la cual accede el usuario, ya sea una computadora de escritorio, un celular, una tablet, etcétera.

> LO MÁS IMPORTANTE DE UN SITIO ES QUE QUE SE ADAPTE A LA PANTALLA DEL USUARIO

Es muy importante entender que cuando hablamos de diseño adaptable **no hablamos de ver el mismo diseño en tamaño pequeño**, sino justamente de **mostrar una interfaz completamente distinta en cada caso**, adaptada a las características específicas de cada dispositivo.

El punto más importante que debemos tener en cuenta es que crear un diseño adaptable no requiere solo de codificación, sino también de un gran trabajo de diseño de interfaz y –sobre todo– cambiar nuestra forma de pensar.

Figura 1. Distintos diseños de interfaz, que dependen del dispositivo con el que accede el usuario.

EL ORIGEN DEL DISEÑO ADAPTABLE

Si nos interesa conocer las fuentes que le dieron origen al diseño adaptable, no deberíamos dejar de leer un interesantísimo artículo escrito por Ethan Marcotte en el año 2010, donde podremos observar en detalle las características de un diseño adaptable mediante un ejemplo de diseño. La URL para acceder al artículo es **www.alistapart.com/article/responsive-web-design**.

El mundo de las media queries

La incorporación de **media queries** marca un salto cualitativo en el lenguaje CSS. Hasta ahora, si queríamos aplicar estilos dependiendo de las características del navegador, nos veíamos obligados a incluir librerías o funciones de JavaScript para poder aplicar un archivo de estilo para cada tipo de pantalla.

En CSS 2.1 ya contábamos con el atributo **media** para aplicar archivos de estilos diferenciados, pero su uso era muy limitado, ya que solo nos permitía detectar el tipo de medio con el cual accedía el usuario, pero no nos ofrecía la posibilidad de comprobar otras características importantes, como la resolución de pantalla o la orientación del dispositivo.

CSS3 introduce las media queries, que nos permiten detectar características propias de los navegadores, como la resolución de pantalla, la profundidad de color, la orientación, etcétera. De este modo, podremos crear una hoja de estilo específica para cada tipo de pantalla o de dispositivo.

Antepasados de las media queries

Como mencionamos anteriormente, desde CSS 2.1 ya contábamos con el atributo **media**, que nos permite detectar el tipo de medio con el cual accede el usuario, mediante el atributo **media** en la etiqueta **link** o bien con la propiedad **@media** en el archivo de estilo. Veamos ejemplos usando las etiquetas **link** y la propiedad **@media**:

```
<link rel="stylesheet" type="text/css" media="screen" href="sans-serif.css">

<link rel="stylesheet" type="text/css" media="print" href="serif.css">

@media screen {
        body { font-family: sans-serif }
}
```

Si bien los **media type** de uso más frecuente son **screen** y **sprint**, también tenemos **braille**, **handheld**, **projection**, **tty**, **tv**, **all**, **embossed** y **speech**. Cada uno de ellos se refiere a un agente de uso distinto.

Anatomía de las media queries

Las media queries son expresiones lógicas que responden a verdadero o falso. Se componen del **media type** y una o más expresiones que permiten detectar características específicas del navegador. Pueden ser usadas de varias maneras:

- Usando la etiqueta **link**:

```
<link rel="stylesheet" media="screen and (color)" href="estilo1.css" >
```

- Usando el comando **import**:

```
@import url(color.css) screen and (color) { ... }
```

- En el archivo de estilo, mediante **@media**:

```
@media screen and (color)  { ... }
```

En los tres ejemplos anteriores, la hoja de estilo se aplicará a los agentes de uso **screen** que además cumplan con la condición de ser pantallas de color. Esto significa que cualquier agente de uso que no cumpla con estas condiciones, ignorará este archivo de estilo.

Si bien todos estos ejemplos son perfectamente válidos, generalmente elegimos trabajar con la propiedad **@media** desde el archivo de estilo externo para mejorar la performance del proyecto. También podemos usar más de una condición en la declaración de la media query, separando con coma cada una de ellas:

```
@media screen and (color), projection and (color) { ... }
```

En este caso, los estilos se aplicarán tanto a los agentes de uso **screen** color, como a **projection** color. Las condiciones también pueden ser expresadas mediante la negación:

```
@media not screen and (color) { ..... }
```

En este ejemplo, el archivo de estilo se aplicará en todos los agentes de uso, con la excepción del que está declarado en la condición que expresa la media query.

Posibilidades de las media queries

Las media queries pueden detectar ciertas características de cada agente de uso. A continuación, repasamos las más utilizadas por los diseñadores web.

Width, max-width y min-width

El atributo **width** y sus prefijos **max** y **min** son de uso frecuente, pues permiten detectar el ancho mínimo y/o máximo de la ventana del navegador:

```
@media screen and (min-width: 900px) { ... }

@media screen and (min-width: 320px) and (max-width: 768px) { ... }
```

En estos casos, el archivo de estilo se aplicará solo si el navegador cumple con la condición de ajustarse a los tamaños establecidos por esta

DISEÑOS ADAPTABLES INSPIRADORES

Para iniciarnos en el mundo de diseño adaptable, podemos recurrir a una interesante fuente de inspiración visual: el sitio web **http://mediaqueri.es** nos ofrece una amplia selección de sitios cuyas interfaces son adaptables.

media query. Estos atributos **no admiten valores negativos**, ya que sería imposible tener una pantalla de -500 px.

Height, min-height, max-height

Al igual que **width**, el atributo **height** y sus prefijos **min** y **max** permiten detectar el alto, alto mínimo y/o máximo de la pantalla. Sus valores no pueden ser negativos, y el uso de estos atributos no es tan frecuente como los atributos de ancho.

```
@media screen and (min-height: 480px) { ... }
```

Device-width, min-device-width, max-device-width

Al igual que el atributo **width**, este también permite detectar el ancho, pero, en este caso, de la ventana del dispositivo.

```
@media screen and (min-device-width: 320px) { ... }
```

Utiliza los prefijos **min** y **max** para detectar el ancho máximo y mínimo de la pantalla del dispositivo. La diferencia entre **width** y **device-width** es que este último se aplicará únicamente a dispositivos móviles, ya que lo que detecta es el ancho de la pantalla y no de la ventana del navegador.

Device-height, min-device-height, max-device-height

Este grupo de atributos se comportan exactamente igual que los que acabamos de ver, con la única diferencia de que detectan el alto del dispositivo. Lógicamente, no admiten valores negativos. Utiliza los prefijos **min** y **max** para detectar el alto máximo y el alto mínimo de la pantalla del dispositivo.

```
@media screen and (min-device-height: 480px) { ... }
```

Orientation

Este atributo permite detectar si el dispositivo está el posición vertical u horizontal. Los valores posibles son **portrait** y **landscape.**

```
@media all and (orientation:portrait) { ... }

@media all and (orientation:landscape) { ... }
```

Aspect-ratio

Este atributo sirve para determinar la proporción entre el ancho y el alto de la ventana del navegador. Veamos dos ejemplos de uso:

```
@media screen and (aspect-ratio: 16/9) { ... }

@media screen and (aspect-ratio: 32/18) { ... }
```

Device-aspect-ratio

Esta propiedad se comporta exactamente igual a la anterior, con la diferencia de que se aplica exclusivamente a dispositivos móviles.

```
@media screen and (device-aspect-ratio: 16/9) { ... }

@media screen and (device-aspect-ratio: 32/18) { ... }
```

Color

Este atributo permite detectar la cantidad de bits de color de una pantalla o dispositivo. Admite los prefijos **min** y **max**, y su valor nunca puede ser negativo:

```
@media all and (color) { ... }

@media all and (min-color: 1) { ... }

@media all and (max-color: 1) { ... }
```

Color-index

Este atributo describe la cantidad de colores de la tabla de color del dispositivo de salida. Por ejemplo, si el dispositivo fuera monocromático, el valor sería igual a cero. Veamos cómo utilizarlo:

```
@media all and (color-index) { ... }
@media all and (min-color-index: 256) { ... }
```

Monochrome

Mediante este atributo podemos detectar la cantidad de bits por pixel en una pantalla monocromática, como la de un lector de libros electrónicos (e-reader). Su valor nunca puede ser negativo.

```
@media all and (min-monochrome: 2) { ... }
```

Resolution, min-resolution y max-resolution

Este atributo detecta la resolución de la pantalla del usuario. Admite los prefijos **min** y **max**, y su valor nunca puede ser negativo.

```
@media print and (resolution: 300dpi) { ... }
@media print and (min-resolution: 300dpi) { ... }
```

Scan

Este atributo se utiliza para TV, ya que describe el modo de barrido (escaneo) de la pantalla del televisor:

```
@media tv and (scan: progressive) { ... }
```

Grid

Este atributo permite determinar si el dispositivo de salida es

del tipo **grid** o del tipo **bitmap**. Si el dispositivo de salida es grid, como por ejemplo una terminal tty o un celular con un único tamaño tipográfico, entonces el valor será 1. De lo contrario, será igual a 0.

```
@media handheld and (grid) and (max-width: 15em) { ... }
@media handheld and (grid) and (device-max-height: 7em) { ... }
```

El atributo viewport

Uno de los puntos más importantes en el desarrollo de un proyecto responsive es controlar el **viewport** desde el encabezado del archivo HTML; de lo contrario, el navegador de un celular o de una tablet aplicará un zoom a nuestro sitio web, ignorando el trabajo realizado en la hoja de estilo mediante las media queries.

Supongamos que creamos un archivo HTML que contiene una imagen de 320 px y luego lo visualizamos en un iPhone, cuyo ancho de pantalla es justamente 320 px. En este caso, debemos asumir que la imagen ocupará todo el ancho de la ventana. No obstante, al visualizarlo podemos comprobar que la imagen solo ocupa una pequeña parte de la pantalla.

Figura 2. Imagen de 320 px en un dispositivo cuyo ancho de pantalla es de 320 px.

Este comportamiento se debe a que el navegador del dispositivo (en este caso, de iPhone) supondrá que nuestro archivo posee un ancho de pantalla de 980 px y le aplicará un zoom para ajustarlo al tamaño real de la ventana. Si creamos un sitio adaptable, este comportamiento nativo del navegador estropeará nuestro trabajo. Para que esto no ocurra, debemos entender el comportamiento del viewport.

Hace algunos años, la empresa Apple introdujo un nuevo valor para la etiqueta **meta**, que permite controlar el modo de visualización de un sitio por parte del navegador. Esta propiedad fue rápidamente adoptada por la mayoría de los fabricantes de dispositivos móviles y se ha incorporado a la especificación de HTML5. Veamos cómo usarla:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

En este ejemplo, la etiqueta le indica al navegador que no aplique zoom sino que utilice el ancho del dispositivo y que la escala inicial sea 1. Si bien este es el caso de uso típico, podemos usar diversos valores en esta etiqueta, según lo que necesitemos indicar al navegador.

Valores posibles de viewport

La etiqueta **meta viewport** posee una serie de atributos y valores mediante los cuales controlar distintos aspectos de la visualización en un dispositivo móvil, como el ancho, el alto, la escala inicial, las escalas máxima y mínima y la escala del usuario. Veamos un ejemplo:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
      maximum-scale=1">
```

Como observamos en este ejemplo, podemos asignarle al atributo **content** más de un parámetro y su correspondiente valor, separando unos de otros mediante comas.

Width y device-width

Con estos atributos podemos indicar el ancho que tomará el sitio en el dispositivo. De esta manera, podemos indicarle un tamaño

exacto en pixeles, aunque esto no es lo más conveniente, ya que deberíamos crear un archivo de estilo para cada ancho que exista. Por ejemplo:

```
<meta name="viewport" content=" width=320">
```

Lo más recomendable es indicarle que utilice el ancho propio del dispositivo, como se ve en el ejemplo a continuación:

```
<meta name="viewport" content="width=device-width">
```

Height y device-height

Estos atributos funcionan exactamente igual que los que acabamos de ver, pero tomando como referencia el alto del navegador y/o del dispositivo. Su uso es realmente muy poco frecuente, ya que generalmente basamos nuestros diseños en el ancho y no en el alto del navegador. Veamos un ejemplo de uso:

```
<meta name="viewport" content=" height=480">
```

Al igual que el atributo **width**, admite valores absolutos o bien el alto propio del dispositivo.

Inicial-scale

Mediante este atributo podemos determinar la escala inicial con la cual el navegador visualizará nuestro sitio web. Lo recomendable es establecer el valor en 1:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

El navegador Opera Mini suele aplicar un zoom inicial mayor a 1. Al cargar la página, esa característica quedará corregida con este atributo.

Maximum-scale y minimum-scale

Estos dos atributos nos permitirán establecer el zoom máximo y mínimo que le permitiremos realizar al usuario en nuestro sitio web.

```
<meta name="viewport" content="width=device-width, initial-scale=1,
      maximum-scale=3, minimum-scale=1">
```

User-scalable

Este atributo permite determinar si el usuario puede o no hacer zoom sobre nuestro sitio web. El valor por defecto es **yes** (sí), de forma tal que si queremos evitar que el usuario escale nuestro sitio debemos usar este atributo con el valor **no**. Veamos cómo usarlo:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
      user-scalable=no>
```

En términos de accesibilidad o de experiencia de usuario, no es recomendable utilizar este atributo, ya que estaríamos quitándole al usuario la libertad de agrandar el sitio para verlo con más claridad o detalle si lo necesitara.

El futuro de la etiqueta meta viewport

Debido a la gran utilidad de esta etiqueta, algunos navegadores como Internet Explorer y Opera han comenzado con la implementación de la propiedad **@viewport** en el archivo de estilo para lograr el mismo efecto sin tener que agregar líneas en el archivo HTML. El código que probablemente utilicemos con mucha frecuencia en el futuro será algo así como:

```
@-webkit-viewport { width: device-width; }
@-moz-viewport { width: device-width; }
@-ms-viewport { width: device-width; }
@-o-viewport { width: device-width; }
@viewport{ width: device-width; }
```

Unidades de medida adaptables

Un punto muy importante a tener en cuenta cuando desarrollamos un proyecto adaptable es el uso de las unidades tipográficas. Si bien podemos usar distintas unidades de medida, debemos erradicar el uso de los pixeles u otras medidas absolutas y, en su lugar, utilizar medidas relativas y accesibles, como los **em** o la nueva unidad **rem**, acerca de las cuales trataremos a continuación.

El uso de em y rem

La unidad de medida **em** es una unidad relativa basada en el tamaño de **m** (la letra de mayor superficie del alfabeto). La importancia del uso de esta medida tipográfica radica en que **permite al usuario controlar el tamaño tipográfico** (agrandarlo o achicarlo) desde las herramientas propias del navegador. Esta unidad de medida es la que debemos usar también si deseamos crear un sitio accesible, dado que es la recomendada en las pautas de accesibilidad establecidas por el consorcio W3C.

Una característica importante de esta unidad de medida consiste en que, al modificar el tamaño de tipografía de la etiqueta **body**, todos los tamaños del resto de las etiquetas definidos mediante **em** se modificarán automáticamente sin ningún tipo de esfuerzo adicional.

Repasemos brevemente la lógica de funcionamiento para esta unidad de medida. Supongamos que tenemos un sitio web cuya hoja de estilo posee las siguientes reglas de estilo:

```
body {font-size:10px;}

h1 {font-size:2em}

a {font-size:1.5em}
```

Si pensamos cuál sería la correlatividad en pixeles –dado que, como seres humanos, nos resulta muy complejo pensar en medidas relativas–

entonces diríamos que la etiqueta **<h1>** posee un tamaño de fuente de 20 px y que la etiqueta **<a>** posee un tamaño tipográfico de 15 px. Esto se cumplirá si todos los elementos están anidados directamente dentro de la etiqueta **<body>**. Asumamos que tenemos el siguiente código HTML:

```
<h1>La casa de la <a href="#">pradera</a></h1>
```

En este caso, debemos saber que el tamaño tipográfico del hipervínculo que está dentro del título será de 35 px, es decir 20 px por 1,5 px, dado que la unidad tipográfica **em** hereda el valor de su padre directo. Esta característica puede, en ocasiones, ser un problema, ya que nos veremos obligados a crear muchas reglas de estilo para definir el tamaño tipográfico de los elementos según su contexto.

Para resolver este obstáculo, la especificación de CSS3 incluye una nueva unidad de medida denominada **rem** (abreviatura para **rootem**), que trabaja de la misma forma que **em**, pero con una diferencia significativa: **no hereda de su padre directo sino del elemento raíz**. De este modo, no importa el contexto donde se utilice: siempre aplicará un múltiplo del tamaño del **body** o del elemento HTML.

Veamos cómo sería en el ejemplo anterior:

```
body {font-size:10px;}

h1 {font-size:2rem}

a {font-size:1.5rem}
```

En esta caso, el elemento **h1** tendrá un tamaño tipográfico de 20 px y el elemento **<a>** tendrá un tamaño de 15 px, sin importar el contexto en el que la etiqueta se encuentre. Estas características hacen de esta nueva unidad de medida un aliado ideal para el desarrollo de un proyecto responsive. La unidad de medida **rem** es soportada por casi todos los navegadores modernos, incluyendo el IE9. Cuando utilicemos **rem** en un proyecto, es importante **no olvidar resetear el tamaño tipográfico en el archivo de estilo** en la etiqueta **<html>** y no en la etiqueta **<body>**, incorporando la siguiente regla en nuestro archivo de estilo:

```
html { font-size:100%; }
```

De esta forma, estaremos utilizando el tamaño tipográfico por defecto de cada dispositivo o navegador. Luego podremos utilizar media queries para establecer el tamaño tipográfico que deseamos en cada caso.

Veamos un ejemplo simple en el que podemos determinar el tamaño tipográfico que queremos para cada tipo de pantalla o dispositivo, con el fin de mejorar la legibilidad y, con ello, la experiencia del usuario:

```
@media (max-width: 640px) {
        body {font-size:1.2rem;}
}

@media (min-width: 640px) {
        body {font-size:1rem;}
}

@media (min-width:960px) {
        body {font-size:1.3rem;}
}

@media (min-width:1100px) {
        body {font-size:1.5rem;}
}
```

RESUMEN

En este capítulo conocimos las principales características del diseño adaptable y vimos en profundidad las posibilidades que nos brindan las media queries para aplicar distintas reglas de estilo dependiendo de la configuración de pantalla del navegador o del dispositivo. También aprendimos la importancia de utilizar la etiqueta **viewport** para un proyecto responsive y cómo utilizar las unidades de medida **em** y **rem** para trabajar con el tamaño de las fuentes.

Actividades

TEST DE AUTOEVALUACIÓN

1. Explique para qué se utiliza la etiqueta **meta** con el valor **viewport**.
2. Explique cuáles son los elementos que componen una **media query**.
3. ¿Cuál es el valor que debemos usar si no queremos que el usuario pueda agrandar o achicar nuestro sitio web cuando es visualizado desde un celular?
4. Explique la diferencia entre los valores **max-width** y **max-device-width**.
5. ¿Cuál es tipo de media que debo utilizar para aplicar un archivo de estilo para impresión?
6. Explique la diferencia entre las unidades de medida **em** y **rem**.
7. En un proyecto responsive, ¿cuál es la medida tipográfica más recomendable para usar? Fundamente su respuesta.
8. Explique brevemente cómo deben ser trabajadas las imágenes y los videos en un proyecto responsive.
9. Explique para qué se utiliza el atributo **user-scalable**.
10. Explique para qué se utilizan los atributos **maximum-scale**, **mínimum-scale** e **initial-scale**.

EJERCICIOS PRÁCTICOS

1. Descargue el archivo **tp-responsive.zip**.
2. Maquete el diseño de referencia utilizando media queries para obtener un layout distinto para un celular, una tablet y una computadora de escritorio.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

11

Las APIs de HTML5

En este capítulo, aprovecharemos las ventajas de HTML5 para acceder a las características del dispositivo que está visitando nuestro sitio web, como la ubicación física del usuario a través de la API de geolocalización. También permitiremos que los sitios funcionen offline con AppCache, almacenaremos datos de forma persistente con WebStorage, haremos una introducción a los WebWorkers y aprenderemos a enviar notificaciones al usuario mediante el navegador.

- Dónde está el usuario 320
- Trabajar sin conexión.....327
- Datos persistentes 331
- Las APIs avanzadas 341
- Notificaciones de HTML5 347
- Resumen.....351
- Actividades.....352

Dónde está el usuario

Con la llegada de HTML5, la tarea de ubicar geográficamente al usuario se ha vuelto muy sencilla. La API que utilizaremos para esto es la **API de Geolocalización** (**Geolocation API**). Pero, ¿para qué nos puede servir ubicar al usuario?

Por ejemplo, podremos mostrar en un mapa su ubicación y, utilizando servicios de terceros –como la API de Google Maps–, indicarle el camino que debe recorrer caminando o con su auto hacia alguna sucursal de nuestro negocio. También, con conocimientos de PHP o algún otro lenguaje del lado del servidor, podremos realizar alguna operación matemática para decirle qué sucursales se encuentran cerca de su ubicación.

A la acción de ubicar al usuario en algún punto de la tierra se la conoce como **geolocalizar**. Para poder hacerlo, existen diversas formas, que veremos a continuación:

- **Dirección IP de la conexión**: en la Argentina, por citar un ejemplo, este dato no es muy preciso, ya que hay muchas conexiones con IP dinámicas y, la mayoría, al consultar su lugar de origen, informa que está radicada en el domicilio del proveedor de acceso a internet (ISP). Lo mismo ocurre con las IP fijas.
- **Por triangulación de celulares**: cuando nos conectamos a internet a través de un celular, el equipo se registra en una o más antenas para obtener señal y transmitir datos. Existe un registro de todas las antenas de conectividad móvil con ubicación geográfica; por lo tanto, mediante una triangulación de esta información es fácil obtener la ubicación aproximada del dispositivo.
- **Triangulación de antenas Wi-Fi**: el dispositivo detecta qué conexiones Wi-Fi hay cerca, las compara con un patrón donde

NAVEGADORES CON GEOLOCALIZACIÓN

Según el sitio **CanIUse.com**, podremos ubicar a un usuario con las siguientes versiones de navegadores o sus posteriores: Internet Explorer 9, Firefox 3.5+, Safari 5.0+, Chrome 5.0+, Opera 10.6+ (a excepción de la versión 15), iOS Safari 3.2+, Android Browser 2.1+, Opera Mobile 11+, Chrome for Android 29+, Firefox for Android 24+, BlackBerry Browser 7.0+ e IE Mobile 10+.

están listadas geográficamente y, así, nos permite obtener una ubicación precisa del dispositivo.

- **GPS**: los equipos que tengan incorporado hardware GPS podrán proporcionarnos su ubicación precisa.

Manos a la obra

Para poder acceder a esta característica, vamos a utilizar la propiedad **geolocation** del objeto **navigator**. Lo primero que debemos hacer es preguntar si está disponible esta propiedad en el navegador.

```
<script>
if(navigator.geolocation){
                console.log('Geolocalización soportada');
        }
        else {
                console.log('La geolocalización no está soportada');
                // Utilizar algún polyfill
        }
</script>
```

Una vez que sabemos que la geolocalización está soportada, tenemos que obtener la posición del usuario. Para esto, utilizaremos el método **getCurrentPosition()**.

```
...
var mostrarUbicacion = function(position){
```

GEOLOCALIZACIÓN EN INTERNET EXPLORER 7

Debido a la gran popularidad de las viejas ediciones de Windows (especialmente, XP), hay muchos usuarios que cuentan con versiones anteriores de Internet Explorer. En estos casos, existe un **polyfill** para añadirle esta característica a las versiones 7 y 8 de este navegador, que se puede encontrar en **WebShim Libs** (**http://afarkas.github.io/webshim/demos/**), o bien utilizar la librería **geo.js**.

```
        latitud = position.coords.latitude;
        longitud = position.coords.longitude;
        console.log(position);
        console.log('Latitud: '+ latitud);
        console.log('Longitud: '+ longitud);

}
var error = function(e){
        console.log(e);
}
        if(navigator.geolocation){
                console.log('Geolocalización soportada');
                //Obtengo la ubicación
                navigator.geolocation.getCurrentPosition(mostrarUbicacion,error);
        }
...
```

Lo que hemos hecho es crear **dos funciones**: una, para realizar una acción en caso de que se encontrara la ubicación, y la otra, en caso de que no se hallara o se produjera un error. Si estamos probando nuestro archivo en Google Chrome, es muy probable que, si hacemos doble clic sobre el archivo y lo abrimos con el método **getCurrentPosition** por defecto, arroje un error. Esto se debe a que las políticas de seguridad de Google Chrome impiden el acceso a esta información desde archivos abiertos de esta forma. Para poder probarlo con este navegador, deberemos subir nuestro sitio a internet y acceder a él mediante la URL (por ejemplo, **http://midominio.com/geo.html**) o ejecutando el navegador con el modificador **--allow-file-access-from-files.**

CONFIDENCIALIDAD

La API de geolocalización de HTML5 es muy poderosa, pero, para utilizarla, el usuario nos tiene que dar permiso y compartir su ubicación. Los browsers son muy cuidadosos y respetuosos de la privacidad del usuario, ya que ningún sitio web puede obtener nuestra ubicación sin haber sido autorizado. Para quitar el acceso a nuestra geolocalización, bastará con ir a la configuración de seguridad de nuestro navegador.

Para hacer estas pruebas vamos a utilizar el navegador Firefox. En este navegador, al abrir el archivo se nos presenta lo siguiente:

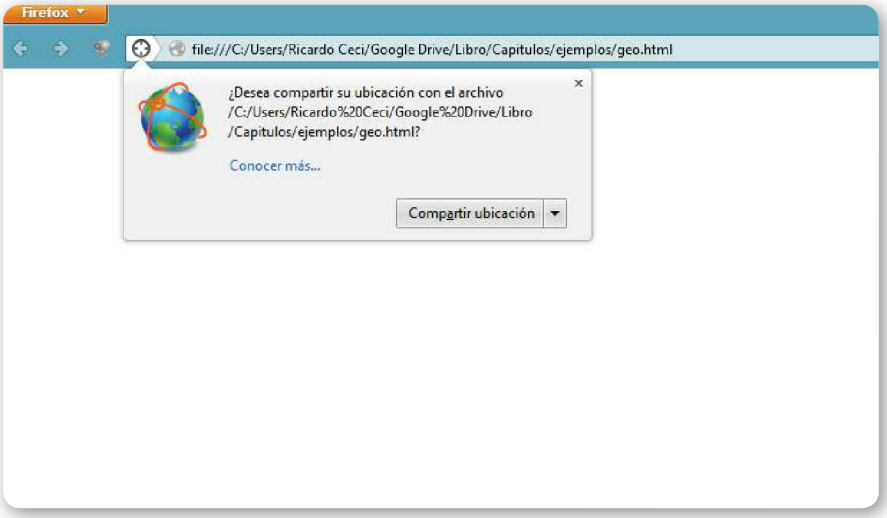


Figura 1. Firefox nos pregunta si deseamos compartir nuestra ubicación.

La geolocalización del usuario es algo que obtendremos si el usuario nos da permiso. De otra manera no podremos obtenerlo. El usuario puede no darnos la ubicación, darnos la ubicación solo una vez, o darnos la ubicación y recordar la acción para que el navegador no vuelva a preguntarle.

En el caso de que el usuario no nos ofrezca su ubicación, automáticamente se disparará la función **error**, y entonces tendremos que indicarle que debe darnos su ubicación para poder realizar su consulta o pedirle que ingrese su ubicación de forma manual para poder brindarle el servicio.

Si el usuario nos da la ubicación, comienza el proceso de ubicarlo. Cuando el navegador encuentra la posición, ejecuta la función **mostrarPosicion** y envía como parámetro un objeto **position**, el cual posee las propiedades que se indican en la **Tabla 1**, a continuación:

PROPIEDADES DEL OBJETO POSITION

PROPIEDAD	DESCRIPCIÓN
coords.latitude	Latitud en grados en formato decimal.
coords.longitude	Longitud en grados en formato decimal.

PROPIEDADES DEL OBJETO POSITION	
coords.altitude	Altitud en metros.
coords.accuracy	Precisión en metros.
coords.altitudeAccuracy	Precisión de la altitud en metros.
coords.heading	Grados en sentido horario desde el norte.
coords.speed	Metros / segundos.
Timestamp	Timestamp de la ubicación.

Tabla 1. Propiedades del objeto **`position`**.

Veamos qué nos devuelve nuestro ejercicio:

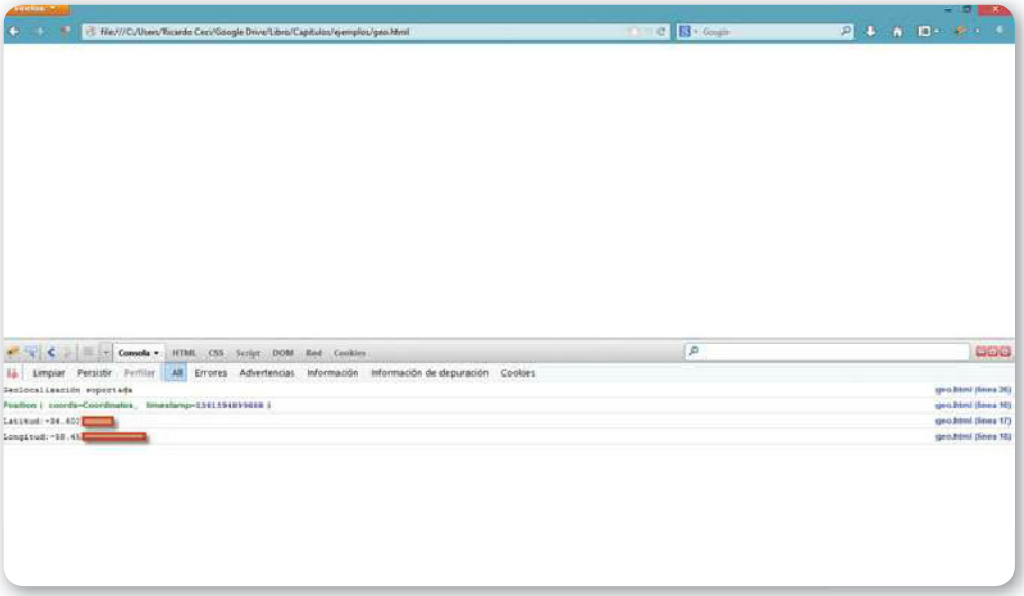


Figura 2. Vemos el resultado de nuestra ubicación.

Mostremos esta ubicación en un mapa. Para esto, utilizaremos un sistema de mapas estáticos de Google Maps y haremos el siguiente retoque a nuestro documento:

```
<!doctype html>
<html>
```

```
<head>
<title>Trabajando con Geolocalización</title>
</head>
<body>
        <div id="mapa"></div>
</body>
<script>
var mostrarUbicacion = function(position){
        latitud = position.coords.latitude;
        longitud = position.coords.longitude;
        console.log(position);
        console.log('Latitud:' + latitud);
        console.log('Longitud:' + longitud);
        latlng = latitud+','+longitud;
        imagen = 'http://maps.googleapis.com/maps/api/staticmap?center=
                '+latlng+'&zoom=14&size=500x500&sensor=false';
        var contenedorMapa = document.getElementById('mapa');
        contenedorMapa.innerHTML = '<img src="'+imagen+'" />';
}
var error = function(e){
        console.log(e);
}
        if(navigator.geolocation){
                console.log('Geolocalización soportada');
                //Obtengo la ubicación
                navigator.geolocation.getCurrentPosition(mostrarUbicacion,error);
        }
        else {
                console.log('La geolocalización no está soportada');
                //Utilizar algún polyfill
        }
</script>
</html>
```

El resultado es un mapa de Google Maps:

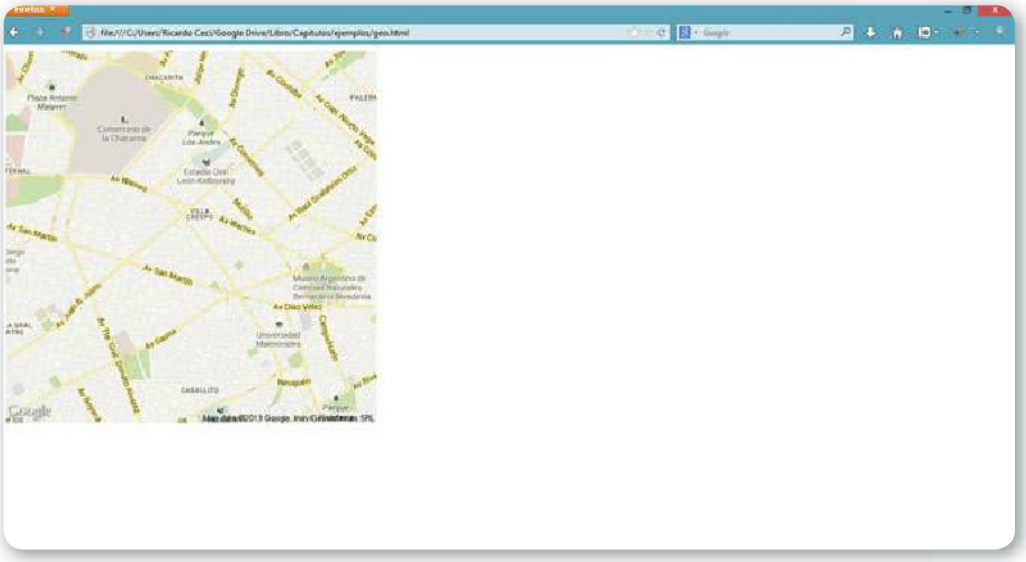


Figura 3. Mostramos en un mapa la ubicación del usuario.

Más precisión

Para obtener más precisión en la ubicación del usuario, disponemos de dos alternativas más. Es posible reemplazar **getCurrentPosition** por **watchPosition()**, que tiene los mismos argumentos pero, además, nos informa de los cambios de posición permanentemente. También devuelve un identificador que debemos guardar en una variable para luego eliminarlo mediante el método **clearWatch(id)** y, así, impedir que siga informándonos sobre la ubicación del usuario.

```
var watchId = navigator.geolocation.watchPosition(mostrarUbicacion,error);

clearWatch(watchId);
```

La segunda opción es llamar a **getCurrentPosition** o **watchPosition** pasando un tercer parámetro, indicándole que queremos la mayor precisión posible. Para esto, debemos activar **highAccuracy.**

```
navigator.geolocation.watchPosition(ok,error,{'enableHighAccuracy':true});
```

Al habilitar esta opción, el dispositivo tratará de obtener la mayor precisión posible. En la mayoría de los casos, utilizará el GPS, motivo por el cual el consumo de batería se verá incrementado. Y, si el usuario tiene el GPS apagado, es probable que se requiera su activación.

Trabajar sin conexión

Es un poco raro hablar de sitios web que funcionen sin estar conectados a internet o aplicaciones que realicen acciones sin estar conectadas a la red. Claro que no podremos acceder a recursos generados en el momento del lado del servidor cuando estamos desconectados, pero sí podremos indicarle al navegador qué recursos descargar para que estén disponibles cuando el usuario se desconecte y pueda navegar nuestro sitio normalmente.

Cuando navegamos habitualmente en internet, el navegador va descargando imágenes, hojas de estilo CSS, scripts de JavaScript y otros elementos. Luego, cuando otra página del mismo sitio llama a esos elementos, el navegador no vuelve a descargarlos, sino que los levanta de una **caché**.

HTML5 nos ofrece una herramienta llamada **AppCache**. Para utilizarla, debemos crear un archivo donde indicaremos qué elementos almacenar localmente para que estén disponibles cuando el usuario se desconecte. Este archivo es llamado **Manifiesto de caché** (**Cache Manifest**).

Se trata de un archivo de texto con una estructura determinada, al que debemos llamar desde todas las páginas de nuestro sitio que queramos que estén disponibles offline o aprovechen esta ventaja.

CONFIGURAR EL SERVIDOR

Para poder procesar los archivos **.manifest**, el servidor debe ser configurado previamente. Para ello, debemos solicitarle a nuestro proveedor de hosting que agregue el tipo de archivos **.manifest** a la lista de archivos soportados y aclararle que debe tratarlo como si fuera un archivo de texto. Algunos servicios de hosting nos permiten hacer esto a nosotros, simplemente agregando un archivo llamado **.htaccess** con el siguiente contenido: **AddType text/cache-manifest .manifest**.

Según el sitio **CanIUse.com**, podremos utilizar los manifiestos de caché con Internet Explorer a partir de la versión 10, Firefox 3.5+, Safari 4.0+, Chrome 5.0+, Opera 10.6, iOS Safari 3.2+, Android Browser 2.1+, Opera Mobile 11+, Chrome for Android 29+, Firefox for Android 24+, BlackBerry Browser 7.0+ e IE Mobile 10+.

Manos a la obra

Para vincular un archivo HTML a un manifiesto de caché, en primer lugar, debemos escribir lo siguiente en nuestro sitio web:

```
<!DOCTYPE html>
<html manifest="cache.manifest">
<body>
...
```

Debemos, además, configurar el servidor para que pueda "servir" este tipo de archivos. Esto se obtiene solicitándolo a nuestro proveedor o bien agregando un archivo **.htaccess** con la siguiente línea:

```
AddType text/cache-manifest .manifest
```

Para probar estos ejemplos, debemos subirlos todos a internet o correrlos sobre un servidor local. Veamos la estructura de un manifiesto de caché:

```
CACHE MANIFEST
/estilo.css
/script.js
/logo.jpg

# Esto es un comentario

NETWORK:
chat.php
```

```
FALLBACK:
/ /offline.html
```

La primera línea de todo archivo de manifiesto es la siguiente:

```
CACHE MANIFEST
```

En nuestro ejemplo, luego de escribir la primera línea, indicamos tres archivos: un **.css**, un **.js** y un **.jpg**. Esta sección es llamada **explícita**: todo lo incluido allí será descargado. Si nos desconectamos de internet y refrescamos la ventana, podremos comprobar que estos archivos estarán disponibles. Luego tenemos la sección **network**, que sirve para indicarle al navegador que hay archivos que no deberá descargar porque requieren, indefectiblemente, de una conexión a internet. Como ejemplo, supongamos que, mediante Ajax, enviamos consultas a un archivo **chat.php**: no servirá de nada si estamos offline. Existe una tercera sección llamada **fallback**, que funcionará en caso de que el recurso no esté disponible en la caché. En primer lugar, se escribe el recurso buscado y, en segundo lugar, se especifica qué mostrar en caso de no encontrarse. Las secciones pueden ir en cualquier orden dentro del archivo de manifiesto y se pueden repetir.

> LAS SECCIONES DEL MANIFIESTO DE CACHÉ SON: EXPLÍCITA, NETWORK Y FALLBACK

¿Cómo funciona?

Los archivos en caché, disponibles offline, lo estarán hasta tanto ocurra alguna de las siguientes cuestiones:

- El usuario borre el almacenamiento de datos del navegador;
- Se modifique el archivo manifiesto;
- Mediante programación, se modifique la caché.

El navegador nos **ofrece 5 MB de espacio como máximo** para almacenar los elementos que estarán disponibles offline.

A tener en cuenta

Si alguno de los recursos declarados en el manifiesto **no se puede almacenar en la caché**, todo el proceso de almacenamiento se interrumpirá y no se guardarán los datos. El navegador disparará un error, pero a simple vista no lo notaremos.

Supongamos que cambiamos el archivo **logo.jpg** del servidor, ¿cómo hacemos para que los navegadores vuelvan a descargar este elemento? Lo que debemos hacer es modificar el archivo manifiesto para que el navegador vuelva a descargar los elementos. El problema es que el nombre del archivo no cambió; entonces, nos encontramos ante un inconveniente de fácil solución: le agregamos una versión y, luego, la modificamos para que el navegador detecte el cambio y vuelva a descargar los elementos.

```
CACHE MANIFEST
#Version 1.0.0


CACHE MANIFEST
#Version 1.0.1
```

Otra forma que tenemos de actualizar la caché es con el objeto **window.applicationCache**:

```
var appCache = window.applicationCache

appCache.update();

if(appCache.status == window.applicationCache.UPDATEREADY) {
        //Intercambiamos la cache
        appCache.swapCache();
        if(confirm('Hay una nueva versión disponible de esta aplicación,
                desea  argarla?')){
        window.location.reload();
}
}
```

Lo que hicimos aquí es guardar dentro de una variable **appCache**, que es el objeto encargado de gestionar el manifiesto de caché; luego forzamos un update y, cuando finalizamos de descargar todo, le informamos al usuario que hay una nueva versión de la aplicación guardada en memoria (por ejemplo, el nuevo logo). Y si la persona lo confirma, recargamos la página para mostrar el nuevo contenido.

Datos persistentes

Según el diccionario, una de las definiciones de **persistencia** es duración, permanencia de una actividad o suceso. Pero llevado a la programación, es **un dato que permanece en el tiempo**. Ese tiempo se da cada vez que el usuario abre el navegador y visita nuestro sitio. Con HTML5, podemos hacer que los datos **persistan** en el tiempo.

Supongamos que nuestro sitio le pide al usuario, la primera vez que ingresa, que nos diga su nombre, su edad y su color favorito. Entonces, de esta forma, seleccionamos un fondo de color adecuado para él y le escribimos en la cabecera del sitio: "Hola **nombredelusuario**, ¿cómo estás pasando tus **edaddelusuario** años?".

Ahora, el mismo usuario vuelve a ingresar el día siguiente a nuestro sitio y (¡oh, casualidad!) le pregunta exactamente lo mismo. Puede hacer dos cosas: enfurecerse y cerrar nuestro sitio, o probar y divertirse poniendo otros datos, perdiendo nuestro objetivo inicial de hacerlo sentir confortable y con una experiencia personalizada.

Históricamente, la persistencia de este tipo de información en las aplicaciones de escritorio se realizaba con los archivos de configuración. Quienes trabajan con Windows habrán visto archivos **.INI**

LOCALSTORAGE EN EQUIPOS MÓVILES

WebStorage es utilizado generalmente en dispositivos móviles para almacenar contenido en el celular y luego consultarlo desde allí sin necesidad de solicitar una conexión adicional, o bien para guardar configuraciones específicas del usuario del dispositivo móvil y así reconocerlo al momento de abrir nuestro sitio. Borrar estos datos en el dispositivo móvil es aún más complejo.

o **.XML** donde hay determinados parámetros y configuraciones que los sistemas buscan al momento de ejecutarse.

En lo que a la programación web respecta, durante mucho tiempo se escuchaba hablar de las **cookies**, consistentes en pequeños archivos de texto que el sitio web dejaba en la computadora del usuario para brindarle información acorde a sus preferencias en sus próximas visitas.

El principal problema de las cookies, más allá del poco espacio de almacenamiento (4 KB), es el usuario en sí: las puede bloquear y no permitirlas, o bien puede borrarlas de manera sencilla. Además, viajan en cada pedido que el usuario hace al sitio, lo cual reduce el rendimiento y la velocidad de carga de la página.

HTML5 nos trae un nuevo concepto para lograr mantener estos datos persistentes, llamado **webStorage**. Nos ofrece 5 MB de espacio para almacenar datos de texto en el navegador del cliente, que estarán disponibles si se recarga la página o si se cierra y se vuelve a abrir el navegador. Además, se puede acceder bajo demanda, cuando el programador desee obtener algún dato.

Según el sitio **CanIUse.com**, **webStorage** es compatible a partir de las siguientes versiones de los navegadores más usados: Internet Explorer 8, Firefox 3.5, Safari 4.0, Chrome 4.0, Opera 10.5, iOS Safari 3.2, Android Browser 2.1, Opera Mobile 11, Chrome for Android 29, Firefox for Android 24, BlackBerry Browser 7.0 e IE Mobile 10.

Manos a la obra

Veamos el ejemplo mencionado al principio de este apartado:

```
<!doctype html>
<html>
<head>
<title>Trabajando con WebStorage</title>
<style>
        div {
                font-family:'Verdana'
        }
        .oculto {
                display:none;
```

```
        }
        .visible {
                display:block;
        }
</style>
</head>
<body>
        <div id="primeraVez" class="oculto">
                <h1>Bienvenido, aún no te conocemos, queremos conocerte</h1>
                <p>Para eso vamos a necesitar algunos datos tuyos
                <p>Nombre: <input type="text" id="txtNombre" />
                <p>Edad: <input type="number" id="numEdad" /> Años
                <p>Color Favorito: <input type="color" id="colorFavorito" />
                <input type="button" value="Guardar Preferencias" onClick=
                        "guardarPreferencias()"/>
        </div>
        <div id="recurrente" class="oculto"></div>
        <div id="generico" class="oculto">
                <h1>Bienvenido, aquí podrás encontrar toda la información
                        que  ecesitásn
                </div>
</body>
<script>
        if(window.localStorage){
                //chequeo que esté disponible localStorage en el navegador
                var hayDatosGuardados = window.localStorage.
                        getItem('hayDatos');
                if(hayDatosGuardados == 1){
                        //Muestro el div de visitante recurrente con datos
                configurarSitio();

                }
                else {
                        // Muestro el formuario para guardar los datos
                        document.getElementById('primeraVez').className =
                                'visible';
                }
        }
```

```
        else {
                // Muestro el div genérico
                document.getElementById('generico').className = 'visible';
        }
</script>
</html>
```

Lo que hicimos fue crear tres elementos ** **: el primero se mostrará si el usuario entró por primera vez; el segundo, si hay datos guardados; y el tercero, si no los hay.

La propiedad que vamos a utilizar para guardar datos persistentes es **localStorage**, para lo cual primero chequeamos que esté disponible. **localStorage** tiene tres métodos:

- **localStorage.getItem('clave')**: recupera el ítem cuya clave sea la que se pasó por parámetro. Devuelve **null** si no la encuentra.
- **localStorage.setItem('clave','valor')**: guarda en **localStorage** dentro de la clave **clave** el valor **valor.**
- **localStorage.removeItem('clave')**: elimina el ítem de memoria.

Como el ítem **hayDatos** no existe, el método **getItem()** devolverá **null** y el documento le pedirá los datos.

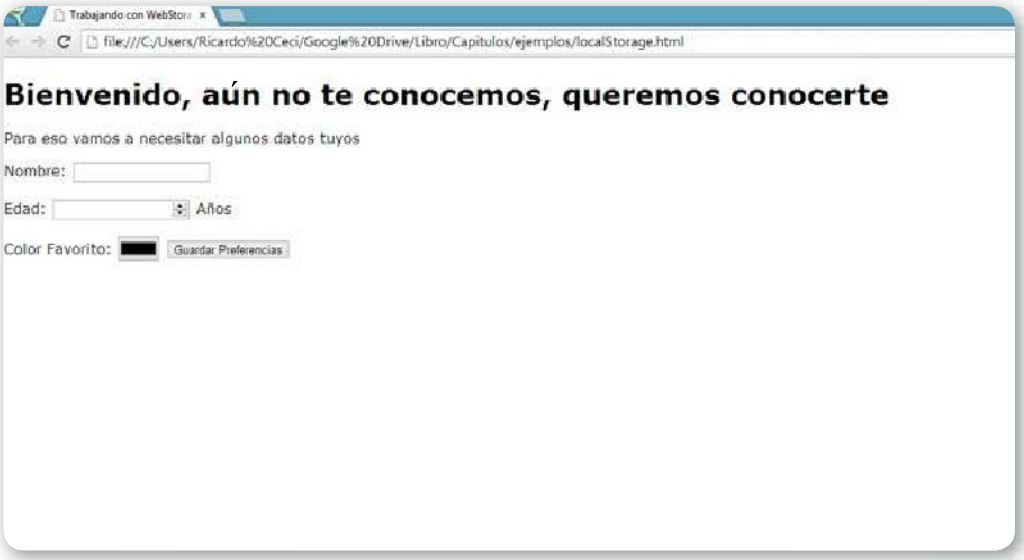


Figura 4. Pedimos los datos para configurar nuestro sitio.

```
function guardarPreferencias(){
        if(window.localStorage){
                var nombre = document.getElementById('txtNombre').value;
                var edad = document.getElementById('numEdad').value;
                var color = document.getElementById('colorFavorito').value;
                window.localStorage.setItem('nombre',nombre);
                window.localStorage.setItem('edad',edad);
                window.localStorage.setItem('color',color);
                document.getElementById('primeraVez').className = 'oculto';
                configurarSitio();
        }
}
```

Aquí lo que hacemos es guardar información en el navegador del usuario y llamamos a la función **configurarSitio()**, que se encargará de preparar el sitio a gusto del usuario. A continuación, debemos definir la función **configurarSitio()**, que será la encargada de ocultar el formulario, cambiar el fondo del sitio, llenar el **div** con el ID **recurrente** y mostrarlo:

```
function configurarSitio(){
        if(window.localStorage){
                var hayDatos = window.localStorage.getItem('hayDatos');
                if(hayDatos == 1){
                        var divRecurrente = document.
                                getElementById('recurrente');
                        var nombre = window.localStorage.getItem('nombre');
                        var edad = window.localStorage.getItem('edad');
                        var color = window.localStorage.getItem('color');
                        //Modifico el div
                        divRecurrente.innerHTML = '<h1>Hola '+nombre+',
        bienvenido a nuestro sitio, como estas pasando tus '+edad+' años?</h1>';
                        //Cambio el color de fondo del body
                        document.body.style.background = color;
                        //Oculto el formulario
                        document.getElementById('primeraVez').className =
                                'oculto';
```

```
                    //Muestro los datos del usuario
                    document.getElementById('recurrente').className =
                            'visible';
                    }
                    else            {
                     //No hay datos, devuelvo falso
                     return false;
                            }
            }
            else {
                    //LocalStorage no soportado
                    return    lse;     fa
            }
    }
```

Ahora sí, estamos listos para probarlo. Veamos cómo funciona:

Figura 5. Primer ingreso: como no se encuentra nada en **`localStorage`**, mostramos el formulario.

POLYFILLS PARA WEBSTORAGE

Existen polyfills para añadir la característica a los navegadores que no lo soportan, entre ellos **cupcake.js** (**www.rivindu.com/p/cupcakejs.html**) y **Storage Polyfill** (**https://gist.github.com/remy/350433**) de **Remy Sharp**. En ambos casos, los datos pasan a almacenarse en cookies y se les añaden los métodos para trabajar como si fueran datos almacenados en **localStorage**.

Figura 6. Completamos los datos del formulario.

Ahora, cuando presionemos **Guardar Preferencias**, inmediatamente se configurará nuestro sitio:

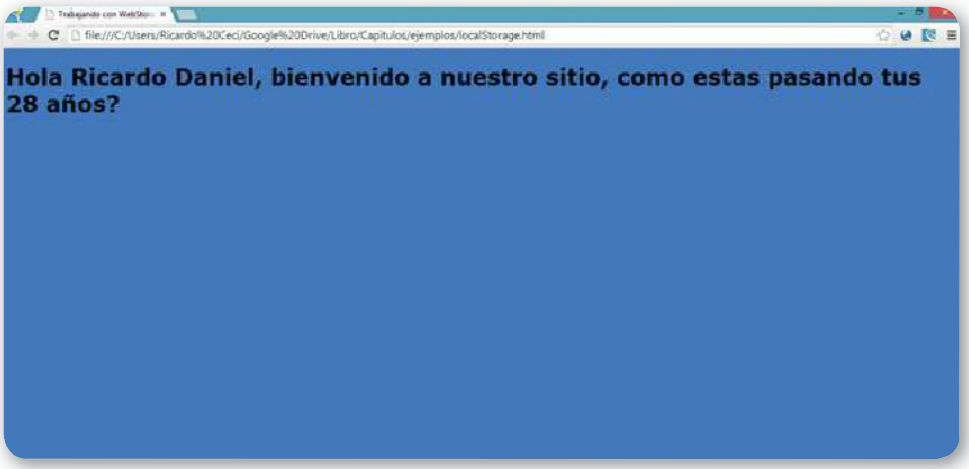


Figura 7. Nuestro sitio está configurado.

Ahora, cerremos el navegador, y volvamos a abrir el archivo: veremos que nuestro sitio se configura automáticamente. Hemos almacenado los datos y persistieron. Veamos el código completo:

```
<!doctype html>
<html>
<head>
<title>Trabajando con WebStorage</title>
<style>
```

```
        div {
                font-family:'Verdana'
        }
        .oculto {
                display:none;
        }
        .visible {
                display:block;
        }
    </style>
    </head>
    <body>
        <div id="primeraVez" class="oculto">
                <h1>Bienvenido, aún no te conocemos, queremos conocerte</h1>
                <p>Para eso vamos a necesitar algunos datos tuyos
                <p>Nombre: <input type="text" id="txtNombre" />
                <p>Edad: <input type="number" id="numEdad" /> Años
                <p>Color Favorito: <input type="color" id="colorFavorito" />
                <input type="button" value="Guardar Preferencias" onClick="gua
rdarPreferencias()"/>
        </div>
        <div id="recurrente" class="oculto"></div>
        <div id="generico" class="oculto">
                <h1>Bienvenido, aquí podrás encontrar toda la información que
necesitás
                </div>
    </body>
    <script>
        if(window.localStorage){
                //chequeo que esté disponible localStorage en el navegador
                var hayDatosGuardados = window.localStorage.
getItem('hayDatos');
                if(hayDatosGuardados == 1){
                        //Muestro el div de visitante recurrente con datos
                        configurarSitio();
                }
                else {
```

```
                    // Muestro el formuario para guardar los datos
                    document.getElementById('primeraVez').className =
                        'visible';
                }
        }
        else {
                // Muestro el div generico
                document.getElementById('generico').className = 'visible';
        }
function guardarPreferencias(){
        if(window.localStorage){
                var nombre = document.getElementById('txtNombre').value;
                var edad = document.getElementById('numEdad').value;
                var color = document.getElementById('colorFavorito').value;
                window.localStorage.setItem('nombre',nombre);
                window.localStorage.setItem('edad',edad);
                window.localStorage.setItem('color',color);
                window.localStorage.setItem('hayDatos',1);
                document.getElementById('primeraVez').className = 'oculto';
                configurarSitio(); // Funcion que cargará los datos en el div con ID
                        'recurrente'y cambiará el color de fondo
        }
}

function configurarSitio(){
        if(window.localStorage){
                var hayDatos = window.localStorage.getItem('hayDatos');
                if(hayDatos == 1){
                        var divRecurrente = document.
                                getElementById('recurrente');
                        var nombre = window.localStorage.getItem('nombre');
                        var edad = window.localStorage.getItem('edad');
                        var color = window.localStorage.getItem('color');
                        //Modifico el div
                        divRecurrente.innerHTML = '<h1>Hola '+nombre+',
                                bienvenido a nuestro sitio, ¿como estás
                                pasando  us '+edad+'       ?</h1>';
                        //Cambio el color de fondo del body
                        document.body.style.background = color;
```

```
                    //Oculto el formulario
                    document.getElementById('primeraVez').className =
                            'oculto';
                    //Muestro los datos del usuario
                    document.getElementById('recurrente').className =
                            'visible';
                    }
                    else {
                     //No hay datos, devuelvo falso
                     return false;
                            }
                }
                else {
                    //LocalStorage no soportado
                    return  lse;  fa
                }
        }
</script>
</html>
```

Un poco de debugging

Para ver qué hay guardado en **LocalStorage**, abrimos la consola de desarrolladores de Google Chrome y vamos a la solapa **Resources**. Allí veremos, entre otras cosas, la opción **LocalStorage**.

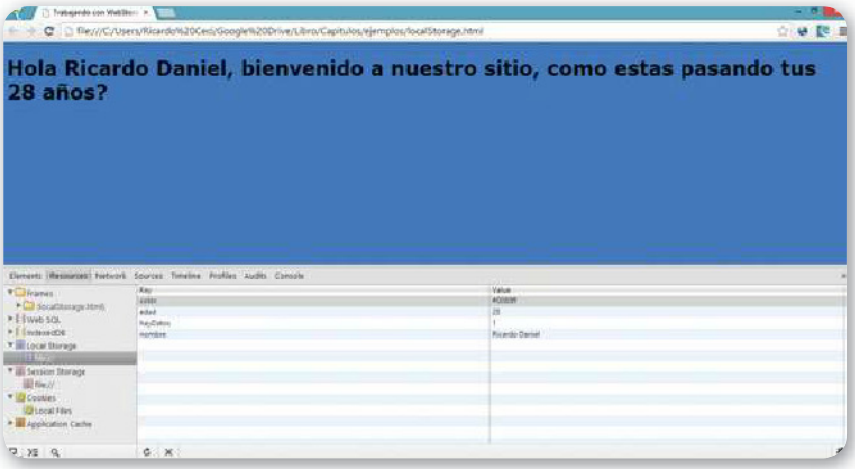


Figura 8. En la consola de Chrome vemos lo que hay en **localStorage**.

Desde allí podemos eliminar (presionando la X que se encuentra en el borde inferior del listado) y editar (haciendo doble clic sobre un valor) cualquier elemento del **LocalStorage**. Esta característica puede ser accesible por cualquier usuario que tenga conocimientos de programación. En consecuencia, hay que tener cuidado de no guardar información sensible (claves, por ejemplo) en este lugar, sino datos que puedan ser almacenados en la máquina del cliente y que él pueda ver sin problemas.

También debemos tener en cuenta que no todo el mundo conoce cómo ingresar a la consola ni su potencial. Tampoco se eliminarán los datos persistentes almacenados en **LocalStorage** si se borra el historial. Por este motivo se trata de una herramienta muy útil para almacenar información del usuario y que esta no sea fácilmente borrada.

Las APIs avanzadas

Si ya tenemos conocimiento y experiencia en el uso de JavaScript, con las APIs que enumeramos a continuación podremos trabajar de manera profesional con este lenguaje, utilizando los nuevos recursos que vienen de la mano de HTML5.

Trabajar en segundo plano

Uno de los problemas más comunes que tiene JavaScript es que es un lenguaje de proceso único; es decir, todas las tareas que se están ejecutando son realizadas por el mismo proceso. Hasta ahora no podíamos, en un proceso, manipular el DOM, mientras en otro proceso realizábamos operaciones matemáticas o peticiones de red.

GEOLOCALIZACIÓN Y GOOGLE MAPS

Utilizar las APIs de Google Maps con la API de geolocalización de HTML5 puede ser muy interesante. Podremos desarrollar buscadores de sucursales, marcar lugares cercanos, agregar puntos en los mapas y trazar recorridos, entre otras opciones. Para obtener más información sobre esta API, podemos ingresar a: **https://developers.google.com/maps/?hl=es**.

HTML5 nos trae a los **WebWorkers** para poder tener múltiples procesos a la vez. Con ellos podremos correr tareas de JavaScript en segundo plano y, por lo tanto, realizar diferentes tareas sin trabarle la página o el sitio al usuario que lo está navegando.

Supongamos que hay que efectuar alguna operación matemática compleja o acceder a registros en **LocalStorage**. Estas operaciones, si demoran, pueden llegar a congelarle la pantalla al usuario o disminuirle el rendimiento y hacer más dificultosa la navegación. Con los WebWorkers, mientras el usuario navega nuestro sitio, en segundo plano podremos correr estas tareas y no interrumpiremos el flujo normal de la experiencia en el sitio.

Los WebWorkers trabajan con un sistema de mensajes para comunicarse entre sí y así lograr el efecto de trabajar al mismo tiempo que el proceso principal. Un caso, por ejemplo, sería poner un cronómetro mientras el usuario está moviendo o desplazando un elemento en el DOM.

De acuerdo con el sitio **CanIUse.com**, los WebWorkers pueden ser utilizados a partir las siguientes versiones de los navegadores más populares: Internet Explorer 10, Firefox 3.5, Safari 4.0, Chrome 4.0, Opera 10.6, iOS Safari 4.0, Opera Mobile 11, Chrome for Android 29, Firefox for Android 24, BlackBerry Browser 7.0 e IE Mobile 10. En Android Browser solo son compatibles en la versión 2.1: en las nuevas versiones no están soportados.

Manos a la obra

Los WebWorkers se ejecutan en un proceso aislado. Para poder lograr esto, tendremos que trabajarlos en un archivo separado. Crearemos nuestro primer worker, que se encargará de contar en segundo plano cuando se lo indiquemos, mientras que en primer plano tendremos el ejercicio de los **TouchEvents** realizado en el **Capítulo 9**. Nuestro archivo **worker.js** deberá contener lo siguiente:

```
var contador = 0;
var timer;

function contar(){
```

```
        contador++;
        self.postMessage(contador);
        timer = setTimeout("contar()",500);
}

function parar() {
        if(timer){
        clearTimeout(timer);
        }
}
self.addEventListener('message',function(e){
        var datos = e.data;

        switch(datos.comando){
                case 'contar':
                        contar();
                        break;
                case 'parar':
                        parar();
                        break;
        }
});
```

Este worker contiene una variable llamada **contador** que está en **0**, una variable del tipo **timer** que será de referencia para el comando **timeOut** de la función **contar**. La función **contar()** incrementa la variable **contador** en 1 (**contador++**) y luego envía un mensaje al hilo que lo llamó (en este caso, el hilo principal) mediante el comando **self.postMessage(datos).**

Luego guarda en la variable **timer** la referencia al **timeout**. La función **parar()** elimina de la memoria al **timeOut** para que pare de contar.

A continuación, escuchamos el evento **message**. Cuando el evento se dispare, se ejecutará una función que recibe como parámetro este evento. Y en la propiedad **data**, recibimos un objeto con un comando, hacemos un switch para ver cuál es el comando enviado y ejecutamos una u otra función. Ahora, veamos cómo queda el hilo principal encargado de mostrar en pantalla lo que sucede:

```
<!doctype html>
<html>
<head>
<title>Eventos Touch</title>
<style>
        #miCaja {
                width:150px;
                height:150px;
                background-color:orange;
                position:absolute;
                }
        #resultado {
                width:50px;
                height:50px;
                background:green;
                color:white;
                }

</style>
</head>
<body>
<input type="button" value="Contar" onclick="contar()"/>
        <input type="button" value="Parar" onclick="parar()"; />
        <div id="resultado"></div>
        <div id="miCaja">Toque aquí</div>
        <div id="resultado"></div>

</body>

<script>
        …

        var worker = new Worker('worker.js');

 function contar() {
   worker.postMessage({'comando': 'contar'});
 }
```

```
function parar(){
        worker.postMessage({'comando': 'parar'});
}
 worker.addEventListener('message',function(e){
        var resultado = document.getElementById('resultado');
                resultado.innerHTML = e.data;

 });

</script>
</html>
```

El código de este ejercicio lo tomamos del **Capítulo 9** y le agregamos dos botones que ejecutarán las funciones **contar** y **parar**, que son las encargadas de enviarle los mensajes al worker. Luego, en un **div** con fondo verde y letras blancas, mostaremos el contador.

Para probar este ejemplo debemos ejecutarlo en nuestro hosting, ya que Google Chrome, por seguridad, impide la llamada a workers de archivos abiertos directamente a menos que lo ejecutemos de la forma mencionada en este capítulo, en el apartado de geolocalización.

> GOOGLE CHROME IMPIDE LA LLAMADA A WORKERS DE ARCHIVOS ABIERTOS DIRECTAMENTE

Luego creamos un nuevo worker mediante el comando **newWorker ('archivo.js')** y lo guardamos dentro de la variable **worker**. En las funciones **contar()** y **parar()** enviamos el mensaje al **worker** mediante el método **worker.postMessage({'comando':'valor'}).**

FUNCIONES DE LOS WEBWORKERS

Los webworkers pueden ser utilizados para múltiples fines. Por ejemplo, podremos acceder al objeto **navigator**, al objeto **location** (solo lectura) y podremos hacer uso del objeto **XMLHttpRequest** para trabajar con AJAX. También es posible generar otros webworkers y acceder a la caché de la aplicación. Así podremos, por ejemplo, realizar las solicitudes por AJAX con otro archivo y enviárselas al hilo principal cuando se hayan procesado a través de un mensaje.

Lo que resta es igual a lo que hicimos en el worker: escuchar un evento del tipo **message**, mediante el comando **worker.addEventListener ('message',function(e){...})**. Veamos cómo funciona:

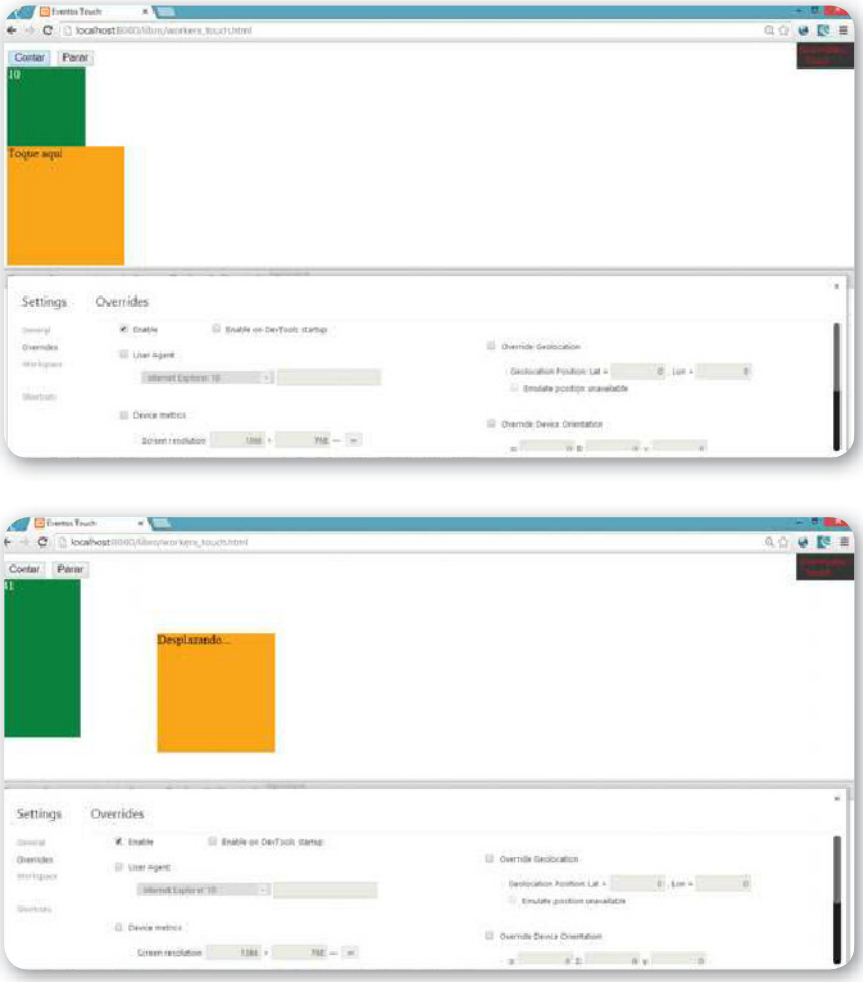


Figura 9. Arrancamos el contador y desplazamos el cuadro. Veremos que el contador sigue funcionando sin interrumpir el desplazamiento.

Como podemos observar, el contador sigue ejecutándose en segundo plano y el desplazamiento del objeto no se ve interrumpido ni impedido. De esta forma, separamos el proceso de ejecutar la cuenta y lo delegamos en un worker.

Notificaciones de HTML5

Las notificaciones nos sirven para alertar o mostrarle al usuario información que acontece en nuestro sitio, sin importar si lo tiene o no en pantalla. El sitio debe estar abierto, pero no es necesario que el usuario lo esté viendo. Muchas veces recibimos carteles de notificación cuando, por ejemplo, nos llega un e-mail o nos envían un mensaje de chat.

> LAS NOTIFICACIONES LE AVISAN AL USUARIO QUÉ SUCEDE CUANDO NO ESTÁ MIRANDO EL SITIO

Las notificaciones en HTML5 se llaman **WebNotifications** y podremos usarlas de manera muy sencilla. Utilizan el sistema de notificaciones por defecto que viene en el sistema operativo, llamándolo desde el navegador.

A diferencia del resto de las APIs que venimos estudiando, las WebNotifications están –al momento de escribir este libro– en proceso de elaboración, motivo por el cual hay muy pocos navegadores compatibles, no existen polyfills y algunos browsers lo implementan con un prefijo.

Según **CanIUse.com**, podremos aprovechar las WebNotifications a partir de las siguientes versiones de los navegadores más usados: Firefox 22, Chrome 22 (sin uso de prefijo), Safari 6.0 (usando el prefijo Webkit), Blackberry Browser 10 y Firefox for Android 24. En Opera 16 Mobile y Chrome for Android 29 también se pueden utilizar con el prefijo Webkit. El resto de los navegadores no mencionados, por ahora no soportan WebNotifications.

Al no ser un estándar y estar aún en formato borrador, los browsers lo trabajan de distinta forma y, al momento de escribir esta obra, la implementación final no está definida. El objeto que vamos a utilizar es el objeto **webkitNotifications** y, en el siguiente ejemplo, veremos los métodos necesarios y cómo se verá en Google Chrome.

```
<!doctype html>
<html>
<head>
<title>Notificaciones</title>
</head>
<body>
<input type="button" value="Pedir Permiso" id="btnPedirPermiso"
        onclick="pedirPermiso()" disabled/>
<input type="button" value="Enviar notificación" id="btnNotificarAhora"
        onclick="notificar()" />
<input type="button" value="Enviar notificación más tarde"
        id="btnNotificarLuego" onclick="notificarMasTarde()" />

</body>

<script>

window.addEventListener('load',function(e){
 if(window.webkitNotifications){
        chequear_permiso();
 }
});
function notificar(){

        if(window.webkitNotifications.checkPermission() == 0){
        console.log('Notificamos');
                var icon = 'img/html5logo.png';
var title = 'Notificacion HTML5';
var content = 'Esta es una notificación enviada desde HTML5';
                window.webkitNotifications.createNotification(icon, title, content).
                        show();

        }
        else {
                alert('Notificaciones No autorizadas');
        }
}

function pedirPermiso(){
```

```
function chequear_permiso(){
        switch(window.webkitNotifications.checkPermission()){
                case 0:
                //Tengo permiso, habilito los botones
                document.getElementById('btnPedirPermiso').disabled = true;
                document.getElementById('btnNotificarAhora').disabled =false;
                document.getElementById('btnNotificarLuego').disabled =false;
                break;
                case 1:
                //No tengo permiso, deshabilito los botones y habilito el de pedir
                permiso
                document.getElementById('btnPedirPermiso').disabled = false;
                document.getElementById('btnNotificarAhora').disabled =true;
                document.getElementById('btnNotificarLuego').disabled =true;
                break;
                case 2:
                //Permiso denegado, deshabilito los botones
                document.getElementById('btnPedirPermiso').disabled = false;
                document.getElementById('btnNotificarAhora').disabled =true;
                document.getElementById('btnNotificarLuego').disabled =true;
                alert("Permiso Denegado");
                break;
        }
}

</script>
</html>
        window.webkitNotifications.requestPermission(chequear_permiso);
}
```

OTRAS APIS PARA APROVECHAR

Además de las APIs mencionadas en el capítulo, podemos investigar otras. Una de ellas es **Device-Motion**, que permite utilizar el acelerómetro y el giroscopio de los equipos móviles. De esta forma, se pueden usar los datos de la aceleración, el ángulo de inclinación, la rotación y orientación del equipo. Otra de ellas es **FullScreen API**, para activar la pantalla completa del navegador.

Lo primero que hacemos es verificar que las notificaciones estén habilitadas y soportadas. Si están habilitadas, revisamos si el usuario nos autorizó; y, si aún no lo ha hecho, habilitamos el botón **Pedir permiso**:

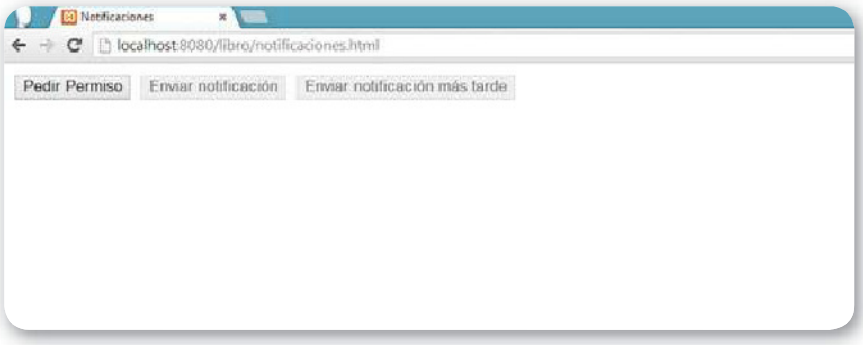


Figura 10. Habilitamos el botón de pedir permiso.

Cuando el usuario lo presiona, el navegador presenta un cuadro de diálogo para autorizar o rechazar el pedido.

Figura 11. Pedimos permiso para mostrar las notificaciones.

Si el usuario nos autoriza, habilitamos los botones **Notificar ahora** y **Enviar notificación más tarde.**

Figura 12. Se habilitan los botones para enviar las notificaciones.

Al presionar el botón **Enviar notificación**, veremos la notificación que hemos creado.

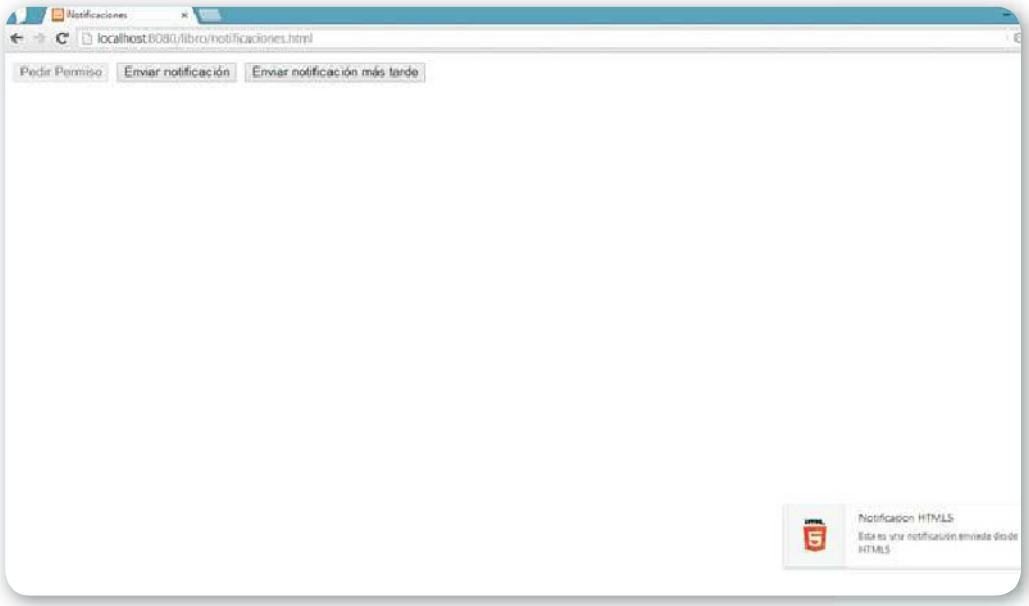


Figura 13. Así se ven las notificaciones en el escritorio de nuestra computadora.

RESUMEN

Hay mucho aún por descubrir en lo que respecta a las APIs de HTML5. En este capítulo, hemos hecho un recorrido por las más interesantes, como la de geolocalización, para ubicar al usuario geográficamente. También aprendimos a trabajar con contenidos sin conexión e introdujimos el concepto del manejo de datos persistentes con la API WebStorage. Luego incursionamos en el trabajo con hilos en JavaScript, mediante la introducción de los WebWorkers, y vimos cómo enviar notificaciones de escritorio al usuario. Esta última API, al estar en formato borrador, aún no tiene definida su implementación. Como esta, hay muchas más APIs que irán apareciendo, así que nuestra tarea será explorarlas a medida que las vayamos necesitando.

Actividades

TEST DE AUTOEVALUACIÓN

1. Mencione dos formas de geolocalizar a un usuario.
2. ¿Puedo geolocalizar al usuario aunque no me dé autorización?
3. ¿Qué es un **manifiesto de caché**?
4. ¿Qué hay que hacer para que los archivos **.manifest** funcionen?
5. Mencione dos secciones de un archivo del manifiesto de caché.
6. Mencione dos métodos del objeto **LocalStorage**.
7. ¿Para qué sirven los **WebWorkers**?
8. ¿Cuál es la función específica para enviar un mensaje entre hilos con WebWorkers?
9. ¿Puedo definir el contenido de un worker en el mismo archivo JavaScript del hilo principal?
10. ¿Para qué sirven las notificaciones de HTML5?

EJERCICIOS PRÁCTICOS

1. Ubique a un usuario con la API de geolocalización y luego muestre en un mapa el área donde se encuentra este mismo usuario.
2. Genere un manifiesto de caché donde se guarde el logo del sitio.
3. Con un webworker genere números al azar, mientras en el hilo principal el usuario completa una encuesta.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

Compatibilidad

En este apartado hablaremos de librerías y extensiones de terceros que harán que nuestros desarrollos en HTML5 lleguen a más dispositivos, sean compatibles con múltiples navegadores y, a la vez, nos simplifiquen el desarrollo. Comenzaremos por jQuery, un framework de JavaScript gratuito, liviano y muy potente. Luego aprenderemos los conceptos básicos de utilización de Modernizr, una librería para detectar compatibilidades de los navegadores.

- Pasar la prueba 2
- Introducción a jQuery 2
- Introducción a Modernizr 22
- Resumen..... 25
- Actividades..... 26

Pasar la prueba

Una pregunta que todos los programadores web hemos escuchado alguna vez es *¿Lo probaste en todos los navegadores?*. Muchas veces, nuestro código JavaScript no se ejecuta de la misma forma en Internet Explorer, en Firefox y en Chrome. Y, a veces, hasta deja de funcionar.

Es por eso que debemos prestar especial atención, al momento de escribir nuestro código, a pensar en desarrollar para todos los navegadores, sin discriminar a ninguno ni a ninguna versión de las más utilizadas en el mercado.

Una de las opciones que tenemos a la hora de resolver estos problemas es trabajar con algún entorno de trabajo o **framework** que incluya las funciones de chequeo de compatibilidades e implementación de estas.

¿Qué es un framework?

Para definir un framework podemos pensar en una librería o conjunto de librerías que contienen un set de soluciones a situaciones comunes que se plantean al momento de programar utilizando una tecnología en particular. Esto significa que, al utilizar estas librerías, podremos, de cierta manera, implementar soluciones planteadas por los desarrolladores del framework para validar campos, automatizar tareas, escuchar eventos, aplicar políticas de seguridad, etcétera.

Algunos frameworks son específicos para interfaz gráfica; otros, para trabajo del lado del servidor; y otros son un mix de ambos. Uno de los frameworks más populares para trabajar con JavaScript es **jQuery**.

Introducción a jQuery

jQuery se define como una librería "pequeña, rápida y con muchas características", que simplifica el trabajo con el DOM de HTML, el manejo de eventos, animaciones, el trabajo con AJAX y que, además, sirve para desarrollar para múltiples navegadores. Técnicamente, jQuery permitirá, de manera simple:

- Buscar elementos en el DOM de HTML.
- Cambiar el contenido de los elementos encontrados.
- Monitorear lo que el usuario está realizando con el sitio y reaccionar de manera correcta (escuchar eventos).
- Realizar animaciones.
- Realizar conexiones a través de la red para pedir datos (AJAX).

Primero debemos descargar la librería jQuery desde **www.jquery.com**. A la derecha, encontraremos el botón **Download jQuery**, donde se presentan dos versiones de esta librería para descargar.

A la fecha de redacción de esta obra, las versiones disponibles son 1.10.2 y 2.0.3. La diferencia principal entre la versión 1.x y la versión 2.x consiste en que esta última no soporta Internet Explorer 6, 7 y 8.

Ambas versiones (1.x y 2.x) son mantenidas. Si no pensamos dar soporte a IE 6, 7 y 8, la recomendación es utilizar la versión 2.x, ya que es más liviana que la 1.x y, por ende, ofrece un rendimiento superior. En cambio, si no podemos dejar de dar soporte a IE 6, 7 y 8, debemos utilizar la versión 1.x que es totalmente compatible y está actualizada.

A la vez, disponemos de dos versiones de cada una para descargar:

- **Production**: es una versión comprimida de jQuery para ambientes que están en producción (sitio en vivo).
- **Development**: es una versión sin comprimir con el código fuente de jQuery, por si queremos modificar algo del core (núcleo) del framework o hacer un debug específico.

Por lo pronto, descargaremos la versión 1.10.2 comprimida para producción. Hacemos clic derecho sobre el link, presionamos **Guardar destino como** y lo guardamos en una carpeta llamada **JS** dentro de nuestro sitio web. Solo nos resta incluirla en nuestro documento HTML y ya estamos listos para trabajar.

```
<!doctype html>
<html>
        <head>
                <meta charset=utf-8 />
                <title>Hola Mundo jQuery!</title>
```

```
        <script type="text/javascript" src="js/jquery-1.10.2.min.js"></script>
</head>
<body>
        <h1>Bienvenidos a mi sitio web con jQuery</h1>
<p>Herramienta que facilita el desarrollo</p>
</body>
</html>
```

Buscar elementos en el DOM

jQuery nos provee de una sintaxis específica para buscar y manipular elementos en nuestro documento HTML. Para trabajar con jQuery, utilizaremos un símbolo específico en nuestro código: el **$**. Si deseamos seleccionar algún elemento del DOM, simplemente debemos llamarlo de la siguiente forma: **$(selector)**, donde **selector** es un selector de CSS.

> JQUERY PERMITE SELECCIONAR ELEMENTOS DOM CON JAVASCRIPT, USANDO SELECTORES DE CSS

¿CSS? Sí, los creadores de jQuery unificaron la forma de seleccionar elementos en JavaScript de la misma manera en que lo hacemos en CSS, tornando más fácil y más corta la sintaxis de desarrollo. Veamos una tabla comparativa de selección de elementos por **ID**, por **clase** y por **etiqueta**:

NAVEGADOR / FORMATO		
SELECCIÓN	JAVASCRIPT	JQUERY
Clase	document.getElementsByClassName(clase)	$('.clase');
ID	document.getElementById(id)	$('#id');
Etiqueta	Document.getElementsByTagName(etiqueta)	$(etiqueta);

Tabla 1. Comparación entre la sintaxis de JavaScript y la de jQuery.

A continuación, veamos un ejemplo: tomemos como referencia el ejemplo introductorio a jQuery que vimos páginas atrás y cambiemos el contenido del **<h1>**.

```
<!doctype html>
<html>
	<head>
		<meta charset=utf-8 />
		<title>Hola Mundo jQuery!</title>
	<script type="text/javascript" src="js/jquery-1.10.2.min.js"></script>
<script type="text/javascript">
	$(document).ready(function(){
	$('h1').text('Hola Mundo jQuery');
});
</script>

</head>
<body>
	<h1>Bienvenidos a mi sitio web con jQuery</h1>
<p> Herramienta que facilita el desarrollo
</body>
</html>
```

Veamos qué fue lo que hicimos. Generalmente, para hacer una modificación de un elemento del DOM necesitamos esperar a que esté **cargado** o **listo** para ser modificado. Si intentamos modificar un elemento y el DOM aún no está cargado, lo más probable es que jQuery no encuentre el elemento y nuestro código falle.

Para esto, jQuery pone a disposición el siguiente comando: **$(document).ready(function(){})**. Este indica que, cuando el documento esté listo, se deberá ejecutar el cuerpo de la función que se pasa como parámetro dentro de **ready()**.

PLUGINS PARA JQUERY

Una de las características que hace que jQuery se utilice cada vez más, a nivel global, en el desarrollo de sitios web es la cantidad de plugins desarrollados para esta librería, que nos permitirán realizar gráficos y trabajar con los elementos del DOM. Por ejemplo, **nivoSlider** sirve para crear galerías de fotos profesionales con muy poco código. El sitio de jQuery posee una lista muy completa de plugins en **http://plugins.jquery.com**.

Luego, dentro de esta función, seleccionamos el elemento **<h1>** del DOM y con el método **.text('valor')** le cambiamos el valor: **$('h1').text('Hola Mundo jQuery');**. El resultado es el siguiente:

Figura 1. Nuestro primer ejemplo utilizando jQuery.

Veamos otro ejemplo:

```
<ul id="listadoItems">
        <li class="destacado">Primer ítem
        <li>Segundo ítem
        <li>Tercer ítem
</ul>

<ul id="otroListado">
        <li>Primero
        <li>Segundo
        <li>Tercero
</ul>
```

En este caso, si deseamos modificar el texto de algún **<li>**, lo primero que se nos ocurre es hacer algo como **$('li').text('nuevo texto');**. Este comando modificaría todo el contenido de texto de los **<li>** de todo

el documento. Entonces, por ejemplo, podemos cambiar el texto de todos los **<li>** dentro del **<ul>** cuyo ID sea **listadoItems**, de la siguiente manera: **$('#listadoItems li').text('Cambio de texto');**

```
<ul id="listadoItems">
        <li class="destacado">Primer ítem
        <li>Segundo ítem
        <li>Tercer ítem
</ul>

<ul id="otroListado">
        <li>Primero
        <li>Segundo
        <li>Tercero
</ul>
<script type="text/javascript">
        $(document).ready(function(){
        $('#listadoItems li').text('Cambio de texto');
});
</script>
```

De este modo, utilizamos la forma de seleccionar elementos de CSS.

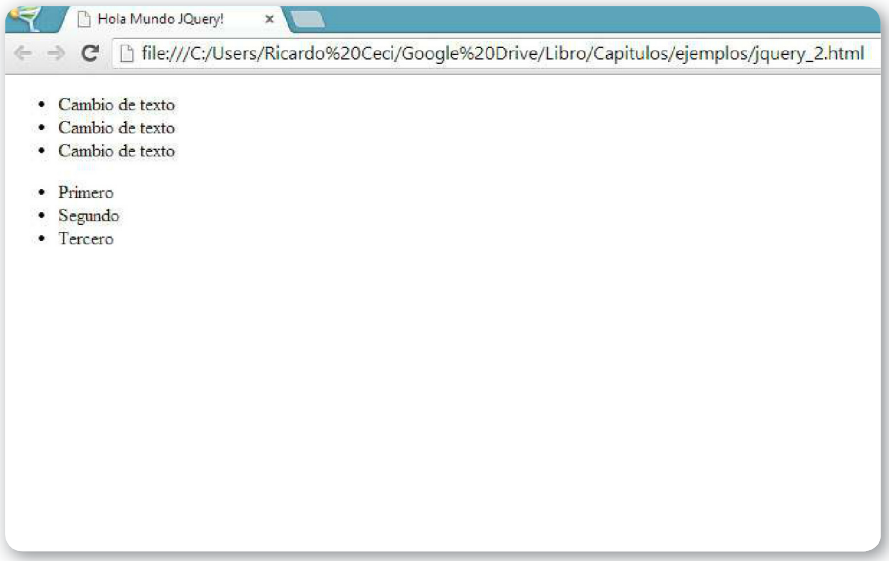


Figura 2. Modificación de elementos con jQuery.

Ahora, supongamos que queremos modificar el elemento "destacado" incluyendo dentro alguna etiqueta HTML. Probemos:

```
<ul id="listadoItems">
        <li class="destacado">Primer ítem
        <li>Segundo ítem
        <li>Tercer ítem
</ul>
<ul id="otroListado">
        <li>Primero
        <li>Segundo
        <li>Tercero
</ul>
<script type="text/javascript">
        $(document).ready(function(){
        $('#listadoItems li').text('Cambio de texto');
        $('#listadoItems .destacado').html('<b>Este elemento está destacado</b>');
});
</script>
```

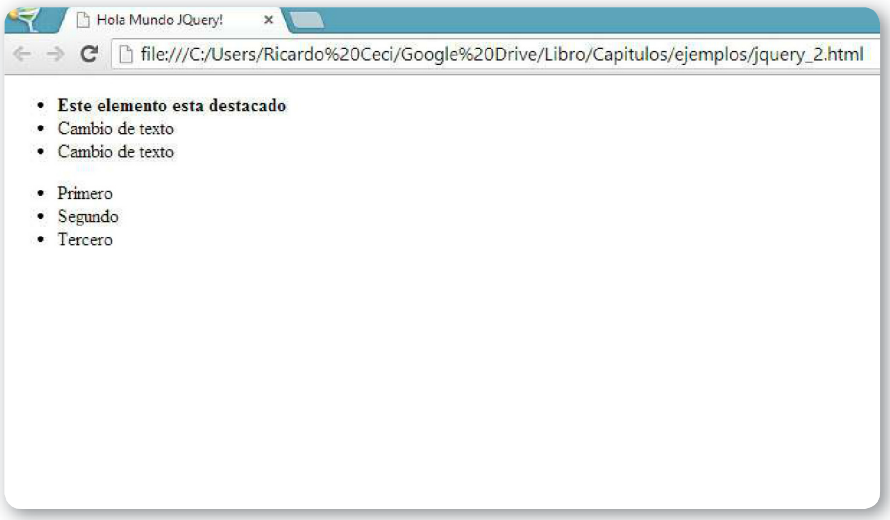


Figura 3. Cambio de elementos por clase.

Atravesar el DOM

jQuery, además, nos permite buscar dentro de los elementos seleccionados. Supongamos que queremos modificar el segundo

elemento de la lista cuyo ID es **otroListado**. Existen varias maneras de realizarlo con jQuery.

Una de ellas es combinar los métodos **.first()** y **.next()**. De esta forma, seleccionamos el primer elemento y, mediante **next**, pasamos al inmediato posterior que cumpla el mismo requisito:

```
$('#otroListado li').first().next().html('Hola Mundo');
```

La otra manera es hacerlo mediante **.eq(indice)**, que nos devolverá el elemento cuyo índice es el pasado por parámetro. El índice empieza desde **0.**

```
$('#otroListado li').eq(1).html('Hola Mundo');
```

De este modo, se nos hace mucho más fácil atravesar el DOM en busca de elementos.

Getters y Setters

La sintaxis de jQuery es muy simple, pero debe ser comprendida para saber cómo obtener (**get**) y cómo modificar o setear valores. Supongamos que queremos recuperar el contenido de un elemento **<li>** para luego hacer una validación. Lo haríamos de la siguiente forma:

```
var textoLi = $('#otroListado li').first().next().html();
```

Ahora, supongamos que queremos modificar el contenido de un elemento **<li>**:

```
$('#otroListado li').first().next().html('<b>Cambio de texto</b>');
```

En ambos casos, utilizamos el método **.html()**. En el primer caso, si no pasamos ningún parámetro, este método sirve para **recuperar** el valor, es decir, funciona como **getter**. Si pasamos algún valor como

parámetro, ahora este método actuará como **setter**, es decir, para **modificar** o **setear** el valor del elemento.

Trabajar con estilos

Con jQuery podemos modificar de manera muy sencilla los estilos CSS de nuestros elementos. Para esto, vamos a utilizar la función **.css()**, que recibe como parámetros la propiedad y el valor.

```
…
<style>
        .caja {
                width:500px;
                height:500px;
                background-color:red;
                float:left;
        }
        </style>
…
<body>
        <div class="caja"></div>
        <input type="button" onclick="accion()" value="Accion!"/>
</body>
<script type="text/javascript">
function accion(){
        $('.caja').css('backgroundColor','yellow');
}
</script>
```

UNA LIBRERÍA PARA MEJORAR LA INTERFAZ

jQuery posee una librería adicional que provee interacciones, widgets y efectos que podemos utilizar para darles a nuestras interfaces gráficas un estilo profesional. Se trata de **jQuery UI** y se puede descargar desde **www.jqueryui.com**. Además, posee una herramienta que se llama **Themeroller**, mediante la cual podremos poner la combinación de colores que nosotros necesitemos y descargar el paquete personalizado a nuestro gusto.

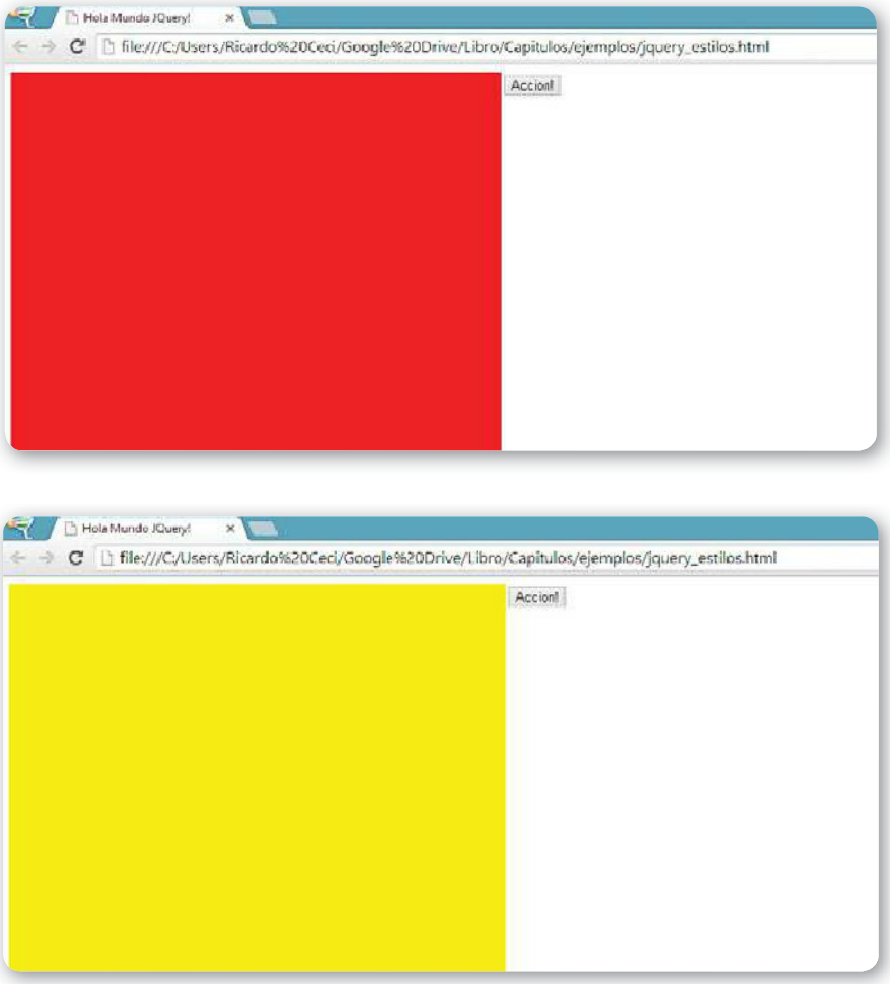


Figura 4. Aplicación de cambios de estilos dinámicamente.

Nótese que las propiedades que llevan **guión** (-), como en este caso **background-color**, deben ser escritas en formato **camelCase**, quedando, por ejemplo, **backgroundColor.**

Efectos

jQuery nos permite mostrar y ocultar elementos fácilmente y, a la vez, darles un efecto. Veamos este ejemplo:

```
...
        <style>
...     .oferta {
```

```
                font-family:'Arial';
                width:350px;
                padding:10px;
                min-height:150px;
                border:1px solid;
                border-radius:5px;
                float:left;
                text-align:center;
        }
        .precios {
                display:none;
                font-color:green;
                list-style:none;
        }
        .boton {
                display:block;
                border-radius:15px;
                width:250px;
                height:20px;
                background-color:green;
                color:white;
                margin:auto;
                font-weight:bold;
                text-decoration:none;
                text-align:center;
        }
        .promo {
                color:orange;
        }
…
        </style>
…
<body>
        <article class="oferta">
                <h2>Vacaciones en Cancún</h1>
                <p>Ideal para parejas all inclusive</p>
                <ul class="precios">
                        <li>7 noches <span class="promo">$ 10.500</span>
```

```
                        <li>10 noches $ 12.500
                        <li>15 noches $ 16.500
                </ul>
                <a href="#" class="boton" onclick="verPrecios()">Mostrar
                        Precios</a>
        </article>
</body>
<script type="text/javascript">
function verPrecios(){
        $('.precios').show();
        $('.boton').hide();
}
</script>
</html>
```

De esta forma, mostramos los precios con el método **show()** y ocultamos el botón con el método **hide()**.

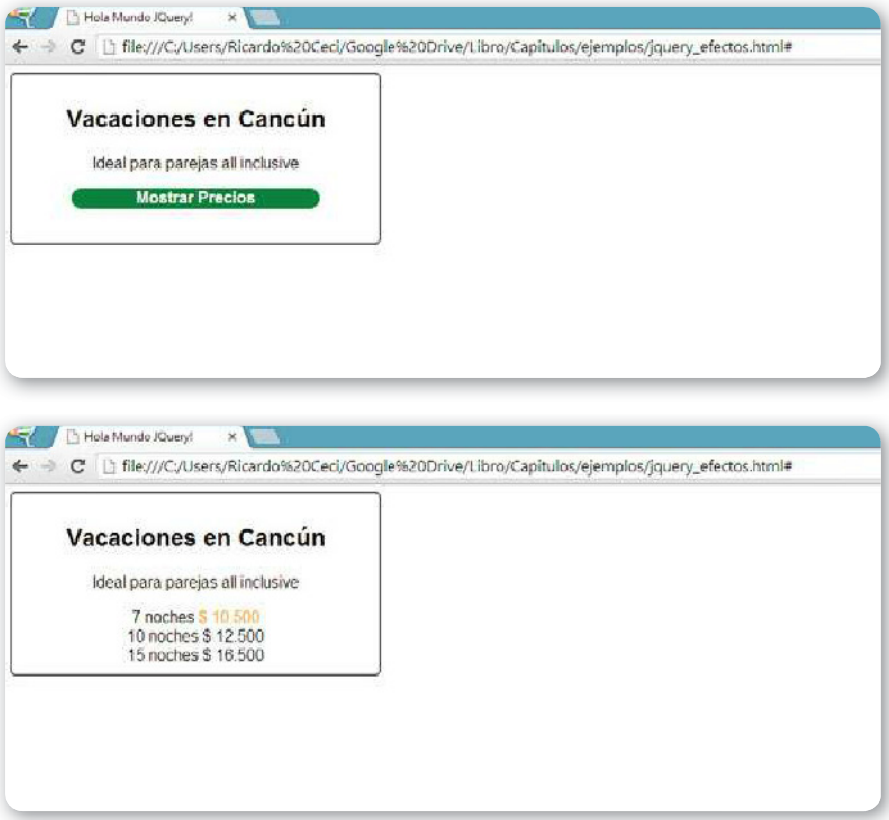


Figura 5. Aquí vemos cómo mostrar y ocultar elementos con jQuery.

Ahora, vamos a darle un efecto por medio de los métodos **fadeIn()** y **fadeOut()**. En el ejemplo anterior, reemplazamos los métodos **show** y **hide** por lo siguiente:

```
<script type="text/javascript">
function verPrecios(){
        $('.precios').fadeIn();
        $('.boton').fadeOut();
}
</script>
```

Veremos cómo –de manera sutil– los elementos aparecen con un efecto **fade in** y se retiran con un **fade out**. Si a **fadeIn()** y a **fadeOut()** les pasamos un valor entre paréntesis, estaremos indicando los milisegundos que durará el efecto.

Ahora supongamos que queremos que, cuando la persona presione el botón **Mostrar precios**, cambie el texto del botón por **Ocultar precios**, y que este efecto sea el inverso cuando se vuelva a hacer clic sobre este botón. Esto es muy sencillo gracias a las funciones **toggle()** y **fadeToggle()** de jQuery. Hagamos algunas modificaciones en nuestro código:

```
…
<a href="#" class="boton" onclick="togglePrecios()">Mostrar Precios</a>
…

<script type="text/javascript">
function togglePrecios(){
        $('.precios').fadeToggle();
        if($('.boton').html() == 'Mostrar Precios'){
                $('.boton').html('Ocultar Precios');
        }
        else {
                $('.boton').html('Mostrar Precios');
        }
}
</script>
```

Aquí combinamos los getters y setters **.html()** y, además, la función **fadeToggle()** para invertir la visiblidad del elemento cuya clase es **.precios**. El resultado es el siguiente:

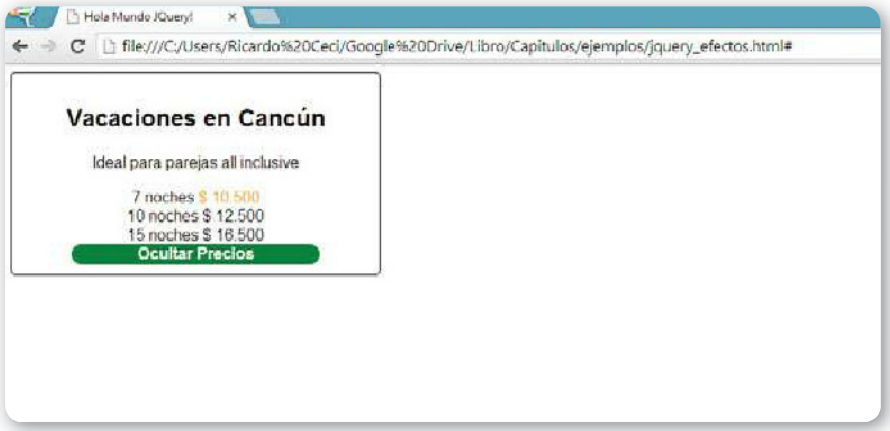


Figura 6. Cambiar el texto del botón y alternar visibilidad de elementos con **fadeToggle**.

Sintaxis encadenada

Sin darnos cuenta, hemos estado desarrollando con una forma especial de sintaxis que posee jQuery, que es la de **encadenar métodos**. ¿Cómo funciona? Volvamos al primer ejemplo de la caja a la cual le cambiamos el color de fondo:

```
…
<script type="text/javascript">
function accion(){
        //Encadenando métodos
        $('.caja').css('backgroundColor','yellow').fadeOut(1000).fadeIn(2000);
}
</script>
…
```

De esta manera, no tendremos que buscar múltiples veces el elemento en el DOM, sino que ejecutaremos las modificaciones de forma encadenada. Para hacerlo, simplemente debemos agregar, después del método utilizado, otro método.

Animaciones

Una de las características más poderosas de jQuery es que podremos crear animaciones de manera muy sencilla, mediante la modificación de propiedades CSS de un elemento. Por ejemplo:

```
<!doctype html>
<html>
	<head>
		<meta charset=utf-8 />
		<title>Hola Mundo jQuery!</title>
	<script type="text/javascript" src="js/jquery-1.10.2.min.js"></script>
	<style>
	.contenedor {
		position:absolute;
	}
	.caja {
		position:relative;
		width:100px;
		height:20px;
		background-color:green;
		color:white;
		text-align:center;
		border-radius:5px;
	}
	</style>

</head>
<body>
<body>
<input type="button" onclick="accion()" value="Accion!"/>
	<div class="contenedor">
	<div class="caja">JQuery</div>
	</div>

</body>
<script type="text/javascript">
function accion(){
	$('.caja').animate({
```

```
                width:'200px',
                height:'150px',
                fontSize:'40px',
                opacity:0.5,
                left:'200px',
                top:'30px'
        },1500);


}
</script>
</html>
```

En este caso, vemos cómo en el lapso de 1,5 segundos (1500 milisegundos) cambiamos el ancho, el alto, el tamaño de la letra y la opacidad del elemento, y lo movemos 200 px desde la izquierda y 30 px desde arriba.

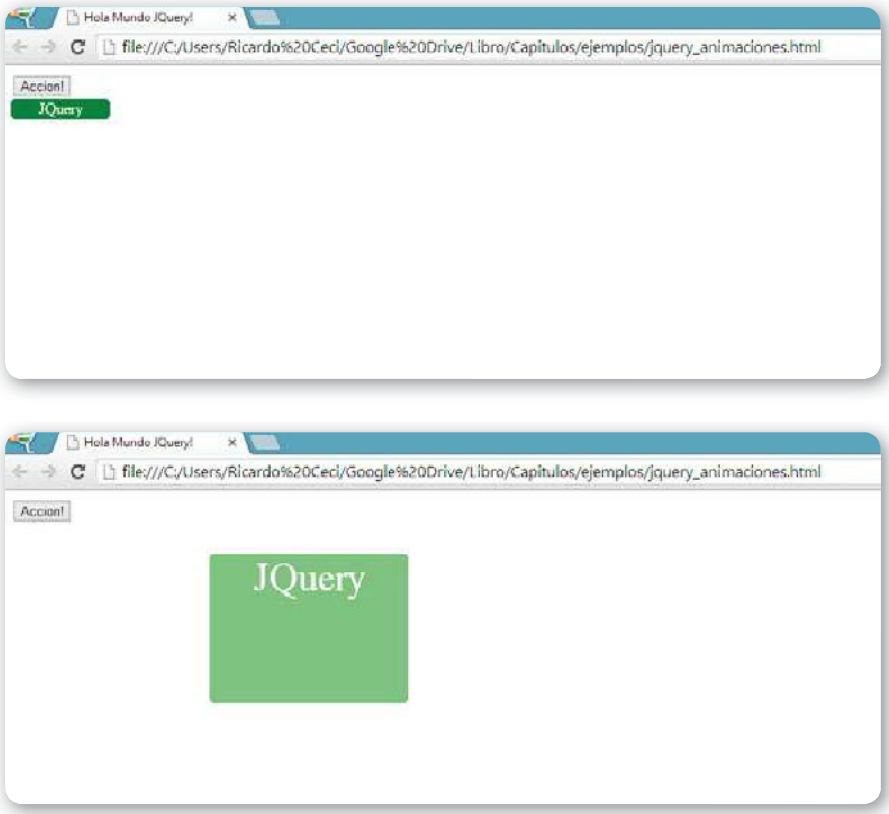


Figura 7. Nuestra primera animación con jQuery.

Encadenar animaciones

Supongamos que, una vez finalizada esta animación, queremos que ocurra otra cosa. Para esto, simplemente debemos encadenar métodos **.animate()**. Por ejemplo:

```
<script type="text/javascript">
function accion(){
        $('.caja').animate({
                width:'200px',
                height:'150px',
                fontSize:'40px',
                opacity:0.5,
                left:'200px',
                top:'30px'
        },1500).animate({
                width:'1000px',
                left:'300px'
        },1000);
}
</script>
```

Esta vez, a continuación de la primera animación, creamos la segunda, que dura 1000 milisegundos (1 segundo).

Callbacks

Cada vez que finaliza la ejecución de una animación, jQuery nos ofrece la posibilidad de informarnos para que nosotros podamos ejecutar una función. Por ejemplo:

ANIMACIONES EN JQUERY

Con un poco de creatividad, podremos realizar banners animados y elementos que le den un atractivo visual a nuestro sitio y que, además, sean compatibles a través de múltiples plataformas y navegadores. Aprovechando el encadenamiento de animaciones y los efectos **fadeIn**, **fadeOut**, **hide**, **show** y **toggle** podremos lograr una buena experiencia de usuario.

```
<script type="text/javascript">
function accion(){
        $('.caja').animate({
                width:'200px',
                height:'150px',
                fontSize:'40px',
                opacity:0.5,
                left:'200px',
                top:'30px'
        },1500,function(e){
                console.log("Efecto 1 terminado");

        }).animate({
                width:'1000px',
                left:'300px'
        },1000,function(e){
                alert("Efecto 2 terminado");
        });
}</script>
```

Luego de cada animación, se ejecuta la función pasada como tercer parámetro.

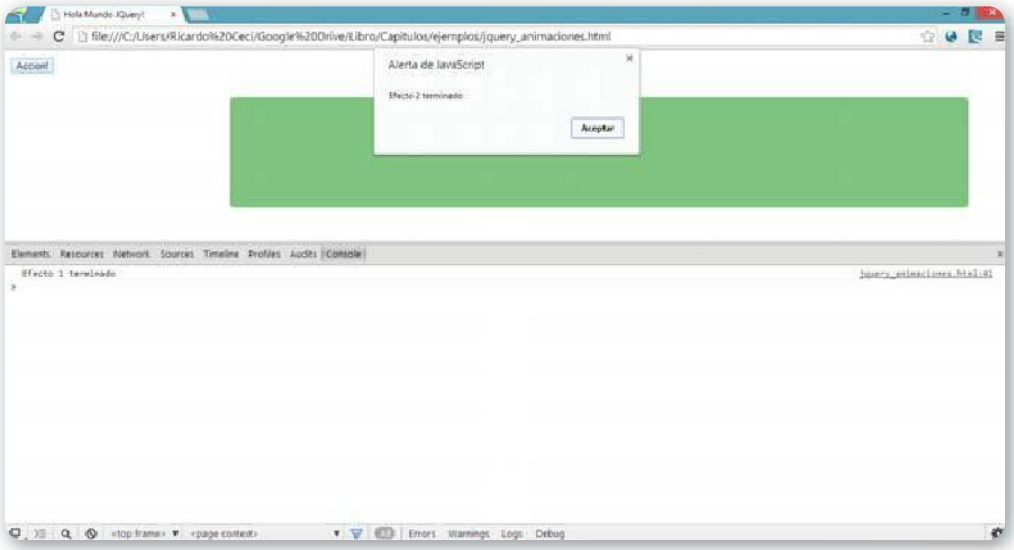


Figura 8. Ejecución de funciones luego de las animaciones.

Agregar y quitar elementos en el DOM

jQuery también nos permite agregar o quitar elementos con facilidad. Para esto, vamos a utilizar los métodos **.append()** y **.prepend().** El método **append()** insertará, debajo del último elemento hijo, un elemento. Por ejemplo:

```
…
<ul id="ingredientes">
        <li>Tomate</li>
        <li>Lechuga</li>
</ul>
…
<script>
var miLista = $('#ingredientes');

//Agrega un ítem al final de la lista
miLista.append('<li>Cebolla</li>');

//Agrega un ítem al principio de la lista
miLista.prepend('<li>Sal</li>');
</script>
```

Para quitar elementos del DOM utilizaremos el método **remove().** Por ejemplo, para quitar el primer elemento de la lista de ingredientes, escribiremos:

```
miLista.first().remove();
```

Eventos

Otra gran característica de jQuery que facilita la tarea del programador web es la posibilidad de escuchar eventos de manera muy sencilla, sin preocuparnos por el navegador donde estamos ejecutando nuestro código.

Para esto, nos provee de ciertos métodos predefinidos, algunos de los cuales enumeramos a continuación. Están agrupados según los lugares donde se disparan y deben ser escuchados.

MÉTODOS PARA ESCUCHAR EVENTOS EN JQUERY

DOCUMENTO	FORMULARIO	MOUSE	TECLADO
load()	blur()	click()	keydown()
ready()	change()	dblclick()	keypress()
unload()	focus()	focusout()	keyup()
	focusin()	hover()	
	select()	mousedown()	
	submit()	mouseenter()	
		mouseleave()	
		mousemove()	
		mouseup()	

Tabla 2. Eventos de jQuery.

Para escuchar cualquiera de estos eventos, debemos seleccionar un elemento con jQuery y agregarle cualquiera de los métodos enumerados anteriormente. En el ejercicio de animaciones, supongamos que queremos que, cuando el mouse se posicione sobre el botón "acción", se dispare la animación. Entonces, debemos modificar nuestro JavaScript de la siguiente manera:

```
…
<input type="button" onclick="accion()" value="Accion!" id="btnAccion"/>
…
<script type="text/javascript">
$(document).ready(function(){
        $('#btnAccion').hover(accion);
});
function accion(){
        $('.caja').animate({
                width:'200px',
                height:'150px',
                fontSize:'40px',
                opacity:0.5,
```

```
                left:'200px',
                top:'30px'
        },1500,function(e){
                console.log("Efecto 1 terminado");

        }).animate({
                width:'1000px',
                left:'300px'
        },1000,function(e){
                alert("Efecto 2 terminado");
        });
}</script>
```

Con mucha facilidad podremos añadirles a los elementos la opción de escuchar ciertos eventos y ejecutar funciones.

Introducción a Modernizr

Modernizr es una librería JavaScript que se encarga de detectar si el navegador donde se está ejecutando nuestro sitio web soporta de manera nativa las nuevas tecnologías de HTML5.

Se ejecuta al momento de cargarse la página y genera un objeto llamado **Modernizr**, que contiene los resultados del chequeo integral de HTML5. Además, posee un sistema de carga de scripts alternativos (polyfills) en caso de que el navegador no tenga soporte específico para alguna característica.

¿Cómo trabajar con Modernizr?

Para trabajar con Modernizr, lo primero que debemos hacer es descargarlo de su sitio web: **www.modernizr.com**.

Al ingresar al sitio, en primer lugar aparecen dos opciones de descarga: la versión de desarrollo (descomprimida) y la versión de producción, que podemos armar de acuerdo a nuestras necesidades, incluyendo solo lo que vayamos a utilizar.

Por ejemplo, supongamos que nuestro sitio utiliza **AppCache** y **LocalStorage** de HTML5. Simplemente, podríamos armar un paquete seleccionando estas opciones, permitiendo, así, que testee solamente la compatibilidad de estas dos APIs.

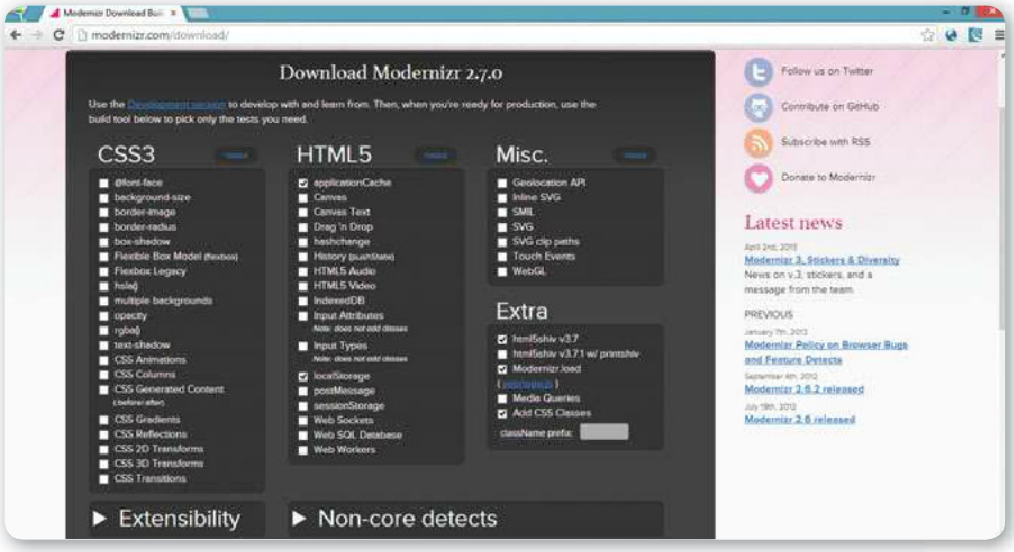


Figura 9. Armado de un paquete especial de Modernizr.

Armar el paquete de producción con las herramientas que vayamos a utilizar es una práctica recomendada para reducir el tiempo de carga de Modernizr. Una vez descargado el archivo generado, lo guardamos dentro de la carpeta de nuestro sitio.

Luego debemos hacer referencia al script en nuestro código. Modernizr recomienda incluirlo justo después de las referencias de CSS. Esto se debe a que esta librería habilita **html5shiv** para hacer compatibles las etiquetas de HTML5 en Internet Explorer; es por esto que debe estar antes de abrir la etiqueta **<body>**. Además, si utilizamos alguno de los estilos y clases que agrega Modernizr, evitaremos que nuestros elementos se vean sin estilos hasta que se cargue la librería.

```
<!doctype html>
<html>
<head>
<title>Trabajando con Formularios</title>
<link href="css/estilos.css" rel="stylesheet" />
```

```
<script src="js/modernizr.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

Modernizr.load()

Una de las formas de utilizar Modernizr es mediante el método **load()**, que sirve para cargar archivos CSS o JavaScript, dependiendo de si el browser soporta o no determinada característica.

Para tenerlo activado, debemos tildar la opción **Modernizr.load()** al momento de descargar el paquete de producción. Para usarlo, debemos llamar a dicho método, que recibe como parámetro un array de objetos con las diferentes pruebas a realizar, con la siguiente estructura:

```
<script>
Modernizr.load([{
        test:Modernizr.localStorage,
        nope:['localStorage-polyfill.js']
        complete:function(){
        alert("Finalizada la carga");
        }
},
        {
        test:Modernizr.canvas && Modernizr.cssgradients,
        nope:['canvas-polyfill.css','estilosalternativos.css']
        }
        ]);
</script>
```

Modernizr.load() también puede servirnos, por ejemplo, para cargar jQuery o cualquier librería de una CDN (*Content Delivery Network* o Red de Entrega de Contenidos). En caso de que no pueda tomarla de allí, nos cargará una versión alternativa dentro de la carpeta **js**. Esta técnica es usada por **html5boilerplate.com**, un generador de plantillas para

arrancar nuestro sitio en HTML5, optimizado y con soporte para muchas características. Veamos el ejemplo:

```
Modernizr.load([
 {
  load: '//ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.js',
  complete: function () {
   if ( !window.jQuery ) {
        Modernizr.load('js/libs/jquery-1.7.1.min.js');
   }
  }
 }
]);
```

Primero, intenta obtener jQuery de la CDN; cuando completa el pedido, testea que el objeto jQuery exista; si no existe, lo carga de la carpeta **js/libs**. Otra forma es preguntando si existe cierta propiedad del objeto **Modernizr**, como en este ejemplo:

```
if(Modernizr.canvas && Modernizr.localStorage){
        // Está todo disponible
}
else {
        //Alguna característica no está disponible.
}
```

RESUMEN

En nuestra tarea como programadores web nos enfrentaremos, frecuentemente, con problemas de compatibilidad entre navegadores y de soporte de ciertas características. Por ello, especialistas del rubro crearon herramientas que son usadas en el día a día, y que en algunos casos se vuelven indispensables. En este apartado hemos visto dos de ellas: jQuery y Modernizr. Utilizamos la primera para hacer más bonito y sencillo nuestro código JavaScript y, a la vez, compatible con todos los browsers. La segunda, para testear de manera eficiente si el navegador soporta o no determinada característica.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es jQuery?
2. ¿Qué diferencia sustancial hay entre la versión 2.x y la versión 1.x de jQuery?
3. ¿Cómo buscamos elementos en el DOM?
4. **$('#elemento').text('Hola Mundo');** ¿es un getter o un setter?
5. **$('#elemento').html(),** ¿es un getter o es un setter?
6. ¿Cómo funciona la sintaxis encadenada de jQuery?
7. ¿Qué es un callback?
8. ¿Qué es Modernizr?
9. ¿En qué parte del documento se recomienda que se haga la llamada al script Modernizr?
10. Mencione dos formas de trabajar con Modernizr para testear compatibilidades.

EJERCICIOS PRÁCTICOS

1. Arme una lista desordenada y, utilizando jQuery, agregue o quite elementos de esta lista.
2. Arme un formulario y valide que todos los campos estén completos con jQuery.
3. Arme una página de ofertas sin precios y haga que estos se muestren al presionar un botón.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

USERS

INCLUYE VERSIÓN DIGITAL GRATIS

SITIOS MULTIPLATAFORMA CON HTML5 + CSS3

Este libro está orientado a diseñadores, programadores y maquetadores que deseen conocer en profundidad el mundo de HTML5, CSS3 y JavaScript para crear sitios y aplicaciones web multiplataforma. A lo largo de los capítulos se detallan los beneficios y el potencial de trabajar con la nueva versión de cada lenguaje, lo que permite crear proyectos web con mejor estructura semántica y mejores resultados. Sin duda, este libro ocupará un lugar dentro de los materiales de consulta de diseñadores y desarrolladores web, de modo tal que puedan recurrir a explicaciones detalladas ante cualquier duda que surja en el desarrollo de un proyecto.

> **La potencia de HTML5, CSS3 y JavaScript permite realizar sitios interactivos, de alto impacto visual y excelente performance, plenamente accesibles.**

EN ESTE LIBRO APRENDERÁ:

- **HTML5 y web semántica:** historia y evolución de HTML. Características, nuevas capacidades y beneficios. Diferencias con versiones anteriores.
- **JavaScript:** características y técnicas avanzadas de programación.
- **Multimedia:** elementos de audio y video de HTML5. Uso de reproductores nativos de navegadores. Incorporación de subtítulos y control desde JavaScript.
- **Canvas:** descripción de la nueva herramienta para gráficos vectoriales.
- **Selectores y nuevas propiedades de CSS3:** sintaxis, compatibilidad con navegadores y creación de interfaces visualmente enriquecidas.
- **Formularios:** nuevos campos para formularios de HTML5. Validación y modo de completarlos por reconocimiento de voz.
- **Responsive Web Design:** fundamentos del diseño y la maquetación adaptable. Aplicación de estilos según resolución (Media Queries).

SOBRE LOS AUTORES

Eugenia Casabona es diseñadora gráfica y docente. Ha trabajado en empresas nacionales y extranjeras en el desarrollo de aplicaciones y proyectos web.

Ricardo Ceci es programador web, docente y consultor en la implementación de nuevas tecnologías. Además, dirige una empresa de desarrollo web.

NIVEL DE USUARIO

Básico / Intermedio

CATEGORÍA

Diseño / Desarrollo web

ISBN 978-987-1949-45-8

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.