USERS

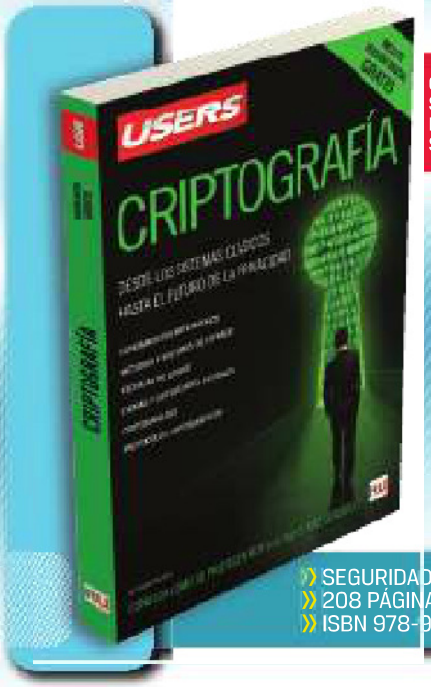
INCLUYE VERSIÓN DIGITAL GRATIS

PHP desde cero

Sintaxis básica del lenguaje + Programación orientada a objetos + Manejo y control de errores + Uso de funciones + Programación de gráficas estadísticas

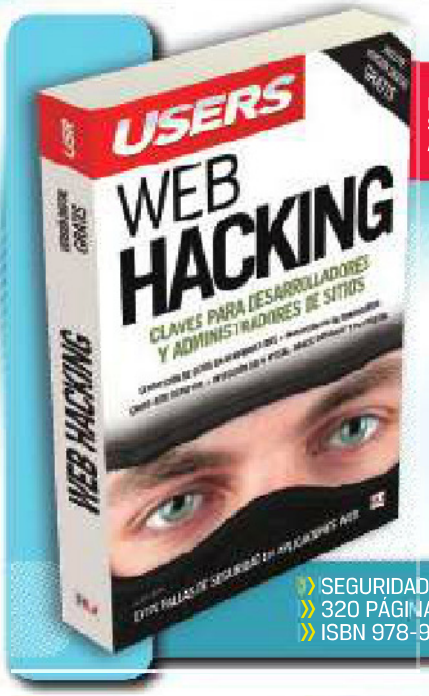
RU

CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN



CONOZCA CÓMO SE PROTEGEN HOY LOS DATOS MÁS SENSIBLES

» SEGURIDAD
» 208 PÁGINAS
» ISBN 978-987-1949-35-9

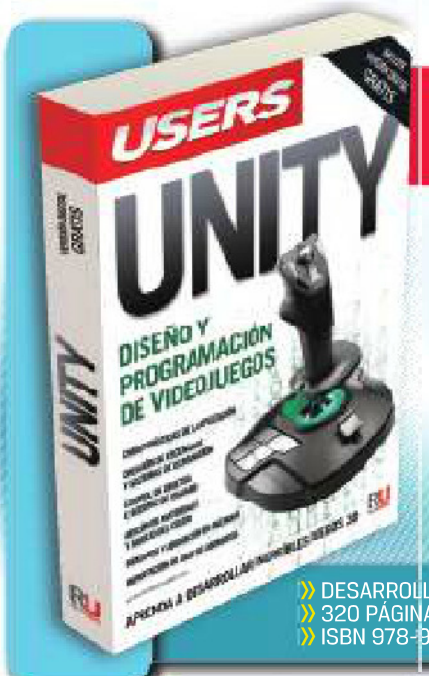


EVITE FALLAS DE SEGURIDAD EN APLICACIONES WEB

» SEGURIDAD / INTERNET
» 320 PÁGINAS
» ISBN 978-987-1949-31-1

POTENCIE SU EQUIPO CON EL ÚLTIMO SISTEMA OPERATIVO

» MICROSOFT / WINDOWS
» 320 PÁGINAS
» ISBN 978-987-1949-09-0



APRENDA A DESARROLLAR INCREÍBLES JUEGOS 3D

» DESARROLLO
» 320 PÁGINAS
» ISBN 978-987-1857-81-4

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

MÁS INFORMACIÓN / CONTÁCTENOS

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

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

PHP
DESDE CERO

RedUSERS

TÍTULO: PHP desde cero

AUTOR: Francisco Minera

COLECCIÓN: Desde Cero

FORMATO: 19 x 15 cm

PÁGINAS: 192

ISBN 978-987-1949-58-8

Minera, Francisco

PHP desde cero / Francisco Minera ; coordinado por Gustavo Carballeiro.

1a ed. - Ciudad Autónoma de Buenos Aires : Fox Andina; Buenos Aires: Dalaga, 2014.

192 p. ; 19x15 cm. - (Desde cero; 34)

ISBN 978-987-1949-58-8

1. Informática. I. Carballeiro, Gustavo, coord. II. Título

CDD 004

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

Prólogo al contenido

¿Por qué elegir PHP como lenguaje para desarrollar aplicaciones web? Existe un sinfín de respuestas para esta pregunta, y uno de los propósitos de este libro es contestarla a través del análisis de casos concretos, para que el lector pueda comprender las características que han hecho de este lenguaje uno de los más populares en la actualidad.

PHP se utiliza en todo tipo de aplicaciones, desde las más simples hasta las más complejas. Este hecho origina que se convierta en una de las opciones preferidas en el momento de comenzar desarrollos de cualquier clase. Otra cuestión interesante es su licenciamiento. En una época en que las empresas buscan, por un lado, estar al corriente de los requisitos legales en cuanto a la utilización de software y, por otro, disminuir lo máximo posible los gastos de licencias y mantenimiento, este lenguaje se presenta como una alternativa prácticamente sin competidores.

Más allá de estas certezas, el principal objetivo de esta obra es ilustrar, de manera clara y práctica, cuáles son las características y posibilidades de PHP, y describir su interacción con las herramientas y tecnologías que lo rodean. Conocer cabalmente el escenario en el que se desempeñan las aplicaciones web hoy en día nos ayudará de manera determinante a sacar provecho de las ventajas que este lenguaje de programación nos brinda.

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

El libro de un vistazo

Conocer a fondo todas las características y posibilidades que un lenguaje de programación como PHP nos brinda es una tarea que requiere paciencia y aplicación por parte del desarrollador. A lo largo de los siguientes capítulos, intentaremos ir asimilando, poco a poco, las técnicas y los métodos necesarios para sacar provecho de la potencia de este lenguaje.

*01 ÁMBITO DE DESARROLLO

Analizaremos las cuestiones y los requerimientos fundamentales para comenzar a desarrollar aplicaciones a través del lenguaje de programación PHP.

*02 SINTAXIS BÁSICA

PHP posee reglas estrictas que deberemos respetar para poder generar aplicaciones; en este capítulo, veremos las más importantes: escritura de instrucciones, manejo de variables, expresiones, inserción de comentarios, entre otras.

*03 ORIENTACIÓN A OBJETOS

La programación orientada a objetos es una técnica muy difundida, adoptada por un gran número de aplicaciones. En este capítulo, conoceremos sus características fundamentales.

*04 REFERENCIA DE FUNCIONES

PHP pone a nuestra disposición una serie de funciones incorporadas que nos permitirán solucionar la mayoría de las necesidades más habituales que surgen durante el proceso de desarrollo de una aplicación. En este capítulo,estudiaremos algunas de las más importantes.

*05 MANEJO Y CONTROL DE ERRORES

Durante el período de desarrollo de una aplicación, deberemos encontrar y remendar todos los posibles errores y falencias. En este capítulo, conoceremos el conjunto de opciones que brinda PHP con el objetivo de visualizar de manera clara las fallas localizadas, para luego poder solucionarlas.

*Ap A ON WEB
FUNCIONES ADICIONALES

En este primer apéndice veremos funciones adicionales para el manejo de archivos y el tratamiento de matrices, que nos permiten cargar contenido en una página web y manipular datos variables, ya sea para el cálculo o la administración de información.

*Ap B ON WEB
GENERACIÓN DE GRÁFICOS

En un sitio web, por lo general, deberemos manipular diferentes clases de imágenes y gráficos. En este apéndice analizaremos cómo PHP nos permite controlar todos los aspectos referidos al tratamiento y a la generación de imágenes.

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 del libro

04
Referencia de funciones

05
Manejo y control de errores

Ap A ON WEB
Funciones adicionales

Ap B ON WEB
Generación de gráficos

Introducción

PHP se caracteriza por brindar al desarrollador una gran libertad, algo que lo diferencia de otros lenguajes más restringidos e inflexibles. Esta cualidad lo posiciona como un medio entre el desarrollador y la idea que persigue, es decir, el objetivo que desea plasmar a través del lenguaje.

En este libro, nos proponemos transmitir su filosofía, y la manera de construir y desarrollar aplicaciones. Sin duda, PHP mantiene una coherencia al respecto y, a medida que vayamos obteniendo experiencia, notaremos que hay un denominador común en la manera de resolver cuestiones en principio muy diferentes.

Algo que destaca al lenguaje es la curva de aprendizaje demandada para quienes quieren comenzar a especializarse en él: tiene una sintaxis simple y a la vez eficaz, que nos permite aplicar soluciones funcionales con pocas líneas de código y en un tiempo reducido.

Todas estas cuestiones explican el éxito y la consolidación que ha tenido en los últimos años. Debemos destacar que no se trata de una moda pasajera, sino de una realidad que lleva años afianzándose y ampliando sus capacidades. En cuanto al lenguaje en sí, los responsables actualizan sus características y funcionalidades en cada nueva versión y toman en cuenta los pedidos de los propios desarrolladores. Esta comunicación hace que el vínculo entre la comunidad de usuarios se fortalezca cada día y genere un compromiso que permita la evolución.

Transmitir un lenguaje en su totalidad es imposible, por eso lo que buscamos es dar a conocer las características de PHP, los puntos fuertes que lo distinguen y lo posicionan como una de las alternativas más viables, en la actualidad, en lo que atañe al desarrollo de aplicaciones web.

*01

Ámbito de desarrollo

En este capítulo, haremos un recorrido por las principales características que hacen de PHP un lenguaje popular y robusto para el desarrollo de aplicaciones web profesionales. Pero antes de comenzar con este lenguaje, repasaremos algunos conceptos fundamentales, que debemos tener presentes sobre las aplicaciones web.

Aplicaciones web

Mientras transcurren los años, surgen distintos tipos de aplicaciones que se adaptan a los diferentes dispositivos que encontramos en el mercado.Teniendo esto en cuenta, podemos clasificar las aplicaciones en: **portátiles, de escritorio** y **web** (dentro de esta categoría, hay subgrupos de aplicaciones).

En la actualidad, la integración de plataformas nos lleva a unificar una gran variedad de portales. Por ejemplo, una aplicación de escritorio puede tener una interfaz web o, a partir de un mismo lenguaje, es posible desarrollar una aplicación y, luego, definir si será accesible a través de un navegador o si será instalada en el equipo personal del usuario. Durante el desarrollo de este libro, analizaremos cuáles son los alcances de PHP como lenguaje. Antes repasaremos algunas nociones sobre las aplicaciones web.

Arquitectura cliente-servidor

Este concepto manejado en varias aplicaciones y particularmente en las de interfaz web, puede definirse como un juego de peticiones y respuestas. Un cliente requiere una acción (por ejemplo, a través de un enlace), y el servidor deberá, por un procesamiento, resolver la demanda y devolver una respuesta. La aplicación cliente por excelencia es el navegador web: desde esta clase de programas accedemos a la interfaz del sistema para interactuar con él.

REDUSERS PREMIUM

Para obtener material adicional gratuito, ingrese a la sección Publicaciones/Libros dentro de **http://premium.redusers.com**. Allí encontrará todos nuestros títulos y podrá acceder a contenido extra de cada uno, como sitios web relacionados, programas recomendados, ejemplos utilizados por el autor, apéndices, archivos editables o de código fuente. Todo esto ayudará a comprender mejor los conceptos desarrollados en la obra.

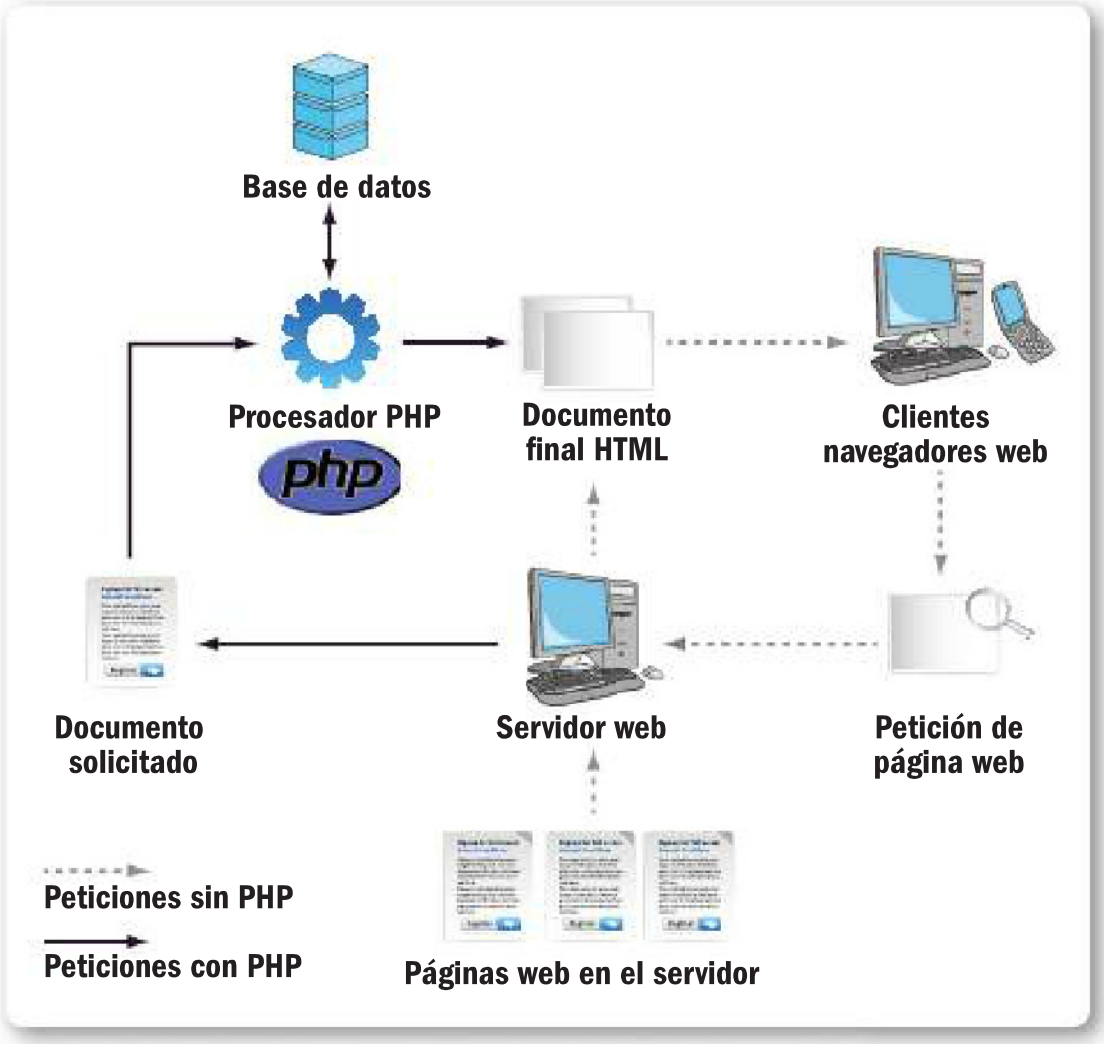


Figura 1. Esquema de funcionamiento cliente-servidor.

En el servidor tendrá lugar una serie de alternativas que desembocará, finalmente, en la construcción de una respuesta comprensible para el cliente. Entre estas alternativas, es posible incluir el tratamiento de la respuesta a través de un lenguaje de programación, por ejemplo, la extracción de información desde una base de datos, entre muchas otras.

Definimos a las peticiones desde un navegador como peticiones web, que son resueltas por los llamados *servidores web*.

Las aplicaciones (**Apache** e **IIS** son los exponentes más conocidos) se encargan de generar las respuestas y para eso se valen, en los casos en los que es necesario, de otros actores como los mencionados antes: servidores

de bases de datos (**Oracle**, **SQL Server**, **MySQL**, **PostgreSQL**, por ejemplo) y lenguajes de programación (**PHP**, **ASP.net**, **JSP**, o **PERL**, entre otros).

Sitios dinámicos

Dentro de las aplicaciones web, podemos diferenciar entre sitios dinámicos y sitios estáticos.

Un lenguaje de programación como alguno de los ya citados nos da la posibilidad de modificar, en tiempo real, la respuesta enviada al cliente sin tener que variar el código de la página.

Pongamos como ejemplo un sitio que incluye un catálogo de productos: si utilizáramos páginas estáticas, deberíamos crear un archivo diferente por cada producto. En cambio, con los lenguajes de programación y al obtener la información particular de cada elemento desde una fuente determinada (por ejemplo, una base de datos), solo necesitaríamos contar con un archivo cuyo contenido dinámico (nombre del producto, foto, descripción, etc.) sería modificado tomando como referencia la petición de usuario.Veamos la siguiente línea:

```
http://www.nombre-sitio.com/pagina.php?idProducto=10
```

En este caso, lo que encontramos después de **.../pagina.pgp** es la parte dinámica del enlace, donde **idProducto** es una palabra que le asignamos a cierta información y **=10** es su identificador dentro de un conjunto de datos.

Sitios estáticos contra sitios dinámicos

Un **sitio estático** es aquel que no utiliza lenguajes dinámicos y devuelve la misma respuesta siempre, más allá del tipo de petición. En cambio, un **sitio dinámico** es aquel que modifica su comportamiento sobre la base de los ingresos del usuario. Una de las ventajas principales del uso de algunas herramientas, como lenguajes de programación del lado servidor, bases de

datos o cualquier fuente de información externa (documentos XML, servicios web, archivos, etc.), es mantener desarrollos centralizados que nos demanden el menor trabajo y tiempo posibles, tanto durante su creación como en su manutención/actualización a lo largo del tiempo.

Tecnologías del lado servidor

Repasando lo analizado anteriormente, podemos decir que, al ingresar una URL en la barra de direcciones de algún navegador web (un navegador es una aplicación cliente), estamos enviando una petición o requerimiento al servidor. Veamos los distintos tipos de servidores que podemos encontrar (entre otros):

- Web: por ejemplo, **Apache**, **IIS**, etc.
- Bases de datos: por ejemplo, **MySQL**, **SQL Server**, **Oracle**, etc.
- Correo electrónico: por ejemplo, **sendmail**, **qmail**, **squirrel**, etc.

Servidores que soportan PHP

Actualmente, PHP se puede ejecutar bajo los servidores web **Apache**, **IIS** (**Internet Information Server**), **PWS** (**Personal Web Server**), **AOLserver**, **Roxen**, **OmniHTTPd**, **O'Reilly WebSite Pro**, **Sambar**, **Xitami**, **Caudium**, **Netscape Enterprise Server** y **THTTPD**, por nombrar algunos.

ORIGEN DEL NOMBRE

Como vimos anteriormente, PHP es un acrónimo recursivo de **PHP: Hypertext Preprocessor** (recursivo porque utiliza PHP en el nombre y como parte de su acrónimo). La primera letra **P** se eligió porque, en los comienzos del lenguaje, este era conocido por la denominación **Personal Home Page Tools**.

Todos estos servidores, o por lo menos los web, están instalados en un equipo remoto que recibe las peticiones y devuelve las respuestas. Por este motivo, no es necesario que una máquina cliente (un equipo que se utiliza para acceder a través de Internet a distintos sitios) tenga instalados servidores web o soporte para bases de datos. Por ejemplo: es el destinatario de la petición el que resolverá los requerimientos y devolverá un documento comprensible para el navegador web. Todo el proceso se lleva a cabo en el servidor.

En cuanto a los lenguajes de programación para el desarrollo de aplicaciones web, también podemos diferenciar los del lado servidor y los del lado cliente. En el primer grupo, incluimos las siguientes alternativas:

- ASP.net (**Active Server Pages**)
- JSP (**Java Server Pages**)
- Perl (**Pratical Extracting and Report Language**)
- PHP (**PHP Hypertext Preprocessor**)

En el segundo grupo, encontramos, principalmente, el lenguaje de programación **JavaScript**. Nada nos impide, y de hecho es usual, incluir o utilizar lenguajes de distinto tipo dentro de una misma página.

El lenguaje PHP

PHP (**PHP Hypertext Preprocessor**) es uno de los lenguajes de programación más utilizados. Gracias a la variedad de clases y funciones disponibles en él, se considera de propósito general. Su uso está destinado, especialmente, al desarrollo de sitios web. Se encarga de programar scripts que se ejecutan del lado del servidor, permitiendo generar páginas de manera dinámica y brindando una gran cantidad de funcionalidades, como el acceso a una enorme variedad de bases de datos (**MySQL**, **PostgreSQL**, **Firebird** y **SQLite** son solo algunas), la generación dinámica de documentos (**PDF**, **XLS**, y **XML**, por ejemplo) o el uso de diferentes pro-

tocolos de Internet (**LDAP**, **IMAP**, **POP3**, **FTP**, **TELNET**, o **Jabber**, entre otros). A continuación, veremos algunos de los múltiples factores que caracterizan a este lenguaje de programación.

Libre y de código abierto

El motor de PHP (es decir, el código que se encuentra en el servidor web y ejecuta scripts PHP) es de código abierto, lo que significa que cualquiera puede acceder y trabajar con el código fuente, ya sea solo para estudiarlo o, incluso, para modificarlo.

A la vez, PHP es un software libre, se puede descargar y utilizar libremente, y esta es una de las razones por la que es tan popular. Su licenciamiento destaca tres puntos básicos:

- Libertad de uso.
- Libertad para acceder al código fuente y modificarlo.
- Libertad para distribuir aplicaciones desarrolladas con PHP.

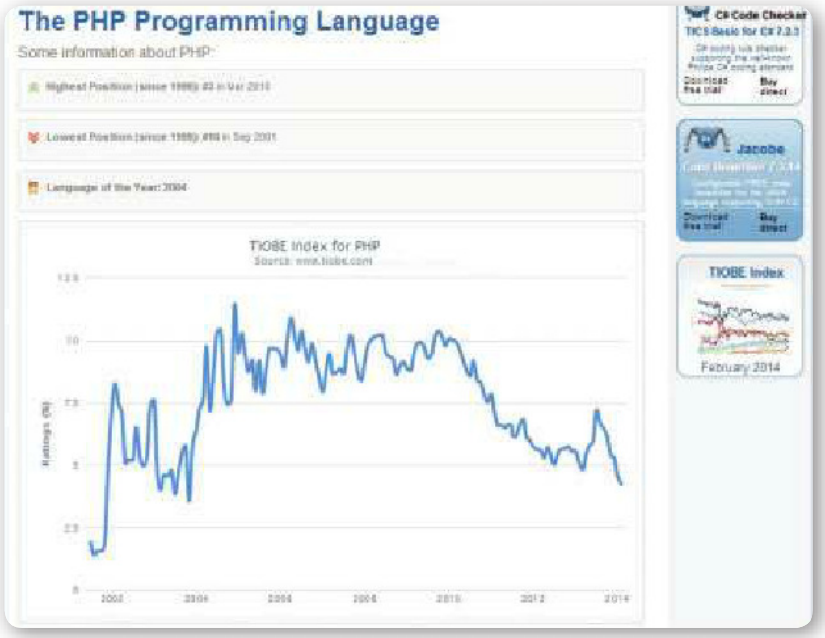


Figura 2. El grado de utilización de PHP en aplicaciones web lo posiciona como uno de los lenguajes más utilizados.

Podemos acceder a la licencia en el sitio web **www.php.net/license**, o bien, en la distribución de la aplicación. La redistribución, la modificación y el uso de este lenguaje están permitidos, siempre y cuando el código fuente esté acompañado de la licencia y la referencia al derecho de autor de PHP. No se permite utilizar su nombre para promocionar productos, a menos que se tenga autorización por escrito del **PHP Group**.

Cuando se desarrolla una aplicación y se la vende a terceros, lo que se cobra no es el lenguaje, sino la solución a un problema, el soporte, etc. La libertad para distribuir software desarrollado con PHP es total.

Al igual que muchos otros lenguajes y entornos de programación, PHP está amparado por el movimiento **open source** (código abierto), que permite a los programadores de aplicaciones aprovechar sus beneficios de manera completamente gratuita.

Facilidad de aprendizaje

PHP se caracteriza por ser de fácil aprendizaje, incluso, para quienes nunca han trabajado con ningún otro lenguaje de programación. Por supuesto que tener conocimientos previos en lo referido al desarrollo de aplicaciones ayuda a entender más rápidamente qué se hace y de qué manera, pero no es indispensable. Aprender sus fundamentos requiere mucho menos tiempo en comparación con otras tecnologías. Además, profundizar sobre algunas cuestiones específicas no es tan difícil si contamos con una base teórica sólida y una cierta experiencia. En PHP es más importante saber con precisión qué se quiere hacer que cómo se hace: contamos con las posibilidades de un lenguaje eficaz y simple a la vez, que se ubica como un medio y no como un fin. Este lenguaje combina dos cualidades: es fácil de aprender y, al mismo tiempo, muy poderoso, y permite avanzar sobre casi cualquier tópico.

> PHP ES FÁCIL DE APRENDER Y MUY PODEROSO, PERMITE AVANZAR SOBRE CASI CUALQUIER TÓPICO

Su sintaxis deriva (y aun hoy es similar) de la del lenguaje C, que, al ser tan popular, nos facilitará el acercamiento. Claro que, para desarrollar aplicaciones complejas, necesitaremos avanzar dentro del lenguaje y conocer sus particularidades.

Disponibilidad

Si bien PHP no es la única alternativa a la hora de desarrollar aplicaciones web, sin duda, es la más popular. En el momento de contratar un servicio de alojamiento, seguramente contaremos con el soporte necesario para empezar a programar nuestras aplicaciones. Al ser gratuito, fácil de instalar y configurar, y además muy requerido por los usuarios, en la mayoría de los casos, este lenguaje estará instalado en el servidor, listo para ser utilizado. PHP es multiplataforma, esto significa que está preparado para trabajar sobre distintos sistemas operativos (incluso, diferentes arquitecturas de hardware):

- Mac OS
- Microsoft Windows
- Unix
- Unix / HP-UX
- Unix / Linux
- Unix / Mac OS X
- Unix / OpenBSD
- Unix / Solaris

En este sentido, es importante destacar que la migración de una aplicación desde un servidor que cuenta con un determinado sistema operativo a otro que posee uno distinto no es un problema. Más allá de las cuestiones específicas, no habrá diferencias en cuanto al comportamiento de las aplicaciones. Esto es fundamental en el momento de desarrollar aplicaciones: normalmente, en el ámbito laboral, no sabremos a ciencia cierta las características de los servidores en los cuales funcionarán de manera definitiva nuestras aplicaciones; por lo tanto, esta característica del lenguaje es central.

Soportes diversificados

Otra característica de PHP es la posibilidad de utilizarlo en diferentes tipos de soportes. Además, a través de **ODBC** (**Open Data Base Connectivity**, conectividad abierta de bases de datos), una capa intermedia entre un motor de bases de datos en particular y el lenguaje, es posible acceder a muchas más.

MULTIPLICIDAD DE SOPORTES EN PHP

▼ SERVIDORES WEB	▼ BASES DE DATOS	▼ PROTOCOLOS DE INTERNET
Apache	Dbase	LDAP
IIS (**Internet Information Server**)	Informix	IMAP
PWS (**Personal Web Server**)	Interbase/Firebird	POP3
AOLServer	Microsoft SQL Server	FTP
Roxen	msql	TELNET
OmniHTTPd	MySQL	Jabber
O'Reilly Website Pro	Oracle	
Sambar	PostgreSQL	
Xitami	SQLite	
Caudium	Sybase	
Netscape Enterprise Server		
THTTPD		
Abiss Web Server		

Tabla 1. Algunos de los principales servidores, bases de datos y protocolos admitidos por PHP.

Comunidad de usuarios

En PHP, se estila y se recomienda reutilizar el código ya desarrollado para no realizar la misma tarea más de una vez. En el mismo sentido, los usuarios buscan, a través de variadas formas, dar a conocer sus experiencias para ayudar a los demás desarrolladores. Detrás de este lenguaje no existe una empresa comercial.

Las continuas mejoras y avances se dan gracias a una comunidad de desarrolladores que contribuyen, sin obtener réditos comerciales, con código fuente, soporte a otros usuarios a través de listas de correo, revisión del programa para detectar errores y notificación de fallas de seguridad (al haber tantas personas que observan el comportamiento de la aplicación, las irregularidades y los posibles agujeros de seguridad se detectan y solucionan muy rápido). Sobre esta base, se sostiene la licencia del lenguaje, que asegura su libertad y que no permite de ninguna manera que alguien saque beneficios comerciales de PHP, ya que nadie es el dueño del lenguaje.

Masividad

Los sistemas **LAMP** y los **WAMP** son muy populares, y PHP mantiene una gran superioridad en cuanto a la cantidad de desarrollos, sobre lenguajes como ASP.NET, ASP, Coldfusion, JSP, Perl y Python. Según **Netcraft**, una popular compañía dedicada, entre otras cosas, a brindar estadísticas acerca del uso de tecnologías en Internet desde el año 1995, el uso de PHP en servidores viene creciendo en forma ininterrumpida, y se hace cada vez más popular en todo el mundo.

Desarrollo en una máquina local

Para comenzar a programar sitios propios, no será necesario contar con dos equipos: podemos utilizar solamente uno, que hará de cliente y de servidor al mismo tiempo. Como mínimo deberemos tener un servidor web (optaremos por uno de los más utilizados y estables de la actua-

lidad: **Apache**), un servidor de bases de datos (elegiremos **MySQL**, por su rapidez, estabilidad, fácil administración y popularidad) y un lenguaje de programación, que será, por supuesto, PHP.

Instalación de las herramientas

Fundamentalmente, hay dos maneras de instalar estas herramientas en nuestro equipo. Una es en forma manual, es decir, con la descarga, la instalación y la configuración de cada una por separado.

La otra es a través de **paquetes de instalación**. Estas aplicaciones están disponibles para distintos sistemas operativos, en especial **Windows**, **Linux** y **MacOs**) y se encargan de automatizar el proceso liberando al usuario de posibles errores e incompatibilidades; además, le ahorran tiempo. Realizan instalaciones estándares, y luego podemos configurar cada herramienta según nuestras necesidades. Además, ofrecen como valor agregado herramientas de administración (por ejemplo, **PHPMyAdmin** y **SQLiteManager**, para bases de datos MySQL y SQLite, respectivamente) y programas para controlar el funcionamiento de los distintos servidores (inicio, apagado, reinicio, etc.). Al descargar esta clase de programas, obtenemos las últimas versiones de cada herramienta, más un instalador para configurar las distintas opciones. Algunas de las alternativas disponibles son:

PAQUETES DE INSTALACIÓN

HERRAMIENTA	SISTEMA	DIRECCIÓN
AppServ	Windows	www.appservnetwork.com
EasyPHP	Windows	www.easyphp.org
MAMP	OS X de Apple	www.mamp.info
VertrigoServ	Windows	vertrigo.sourceforge.net

PAQUETES DE INSTALACIÓN (CONTINUACIÓN)		
WAMP Server	Windows	www.wampserver.com
XAMPP	Linux, Windows, MacOS y Solaris	www.apachefriends.org

Tabla 2. Paquetes disponibles según los diferentes sistemas operativos.

Tomaremos como ejemplo **WAMP**, una de las herramientas más utilizadas por su calidad y sencillez. Al descargar la versión de WAMP, es recomendable elegir la última disponible. Este paquete provee distintas distribuciones, y la diferencia principal entre ellas radica en las herramientas que cada una incorpora. Lo mínimo que requerimos para comenzar a desarrollar aplicaciones será una distribución que cuente con Apache, MySQL y PHP.

Instalar WAMP

La instalación de WAMP es la típica de las aplicaciones para Windows, por lo que no deberíamos tener inconvenientes. Luego de la pantalla de bienvenida, aprobamos las condiciones de licenciamiento (es una aplicación gratuita) y seleccionamos el directorio de instalación (por defecto es **c:\wamp**; en los siguientes apartados, asumiremos que se seleccionó este directorio). Después, elegimos un nombre para los accesos directos y la opción de iniciar o no WAMP de manera automática junto con Windows.

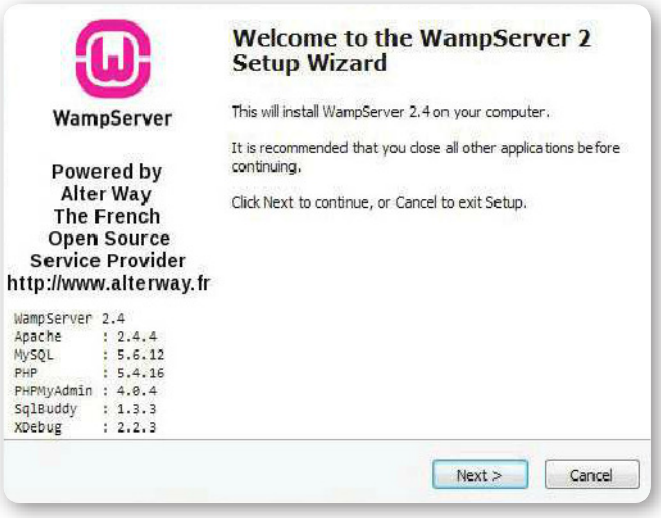


Figura 3. Al ejecutar el instalador, contaremos con todas las herramientas necesarias para desarrollar aplicaciones web.

Una vez concluidos estos pasos, se nos preguntará cuál es la carpeta que hará de **DocumentRoot** (el directorio en el cual almacenaremos los sitios). Conservamos la opción predeterminada:

```
c:\wamp\www
```

En la siguiente sección, se nos solicitará la dirección del servidor de correo. Como este dato no es imprescindible para nuestros primeros desarrollos, dejamos el valor por defecto: **localhost** (lo mismo para la dirección de correo, **you@yourdomain**).

En algunos minutos, veremos que la interfaz de WAMP nos permite acceder de manera rápida e intuitiva a diferentes opciones relativas al manejo y la administración de servidores y sitios. En relación con esto, el instalador nos pide que definamos la ruta hacia el navegador que WAMP utilizará para acceder a nuestros sitios. Por ejemplo, podríamos seleccionar Internet Explorer (**c:\windows\explorer.exe**) o Mozilla Firefox (**c:\archivos de programa\Mozilla Firefox**).

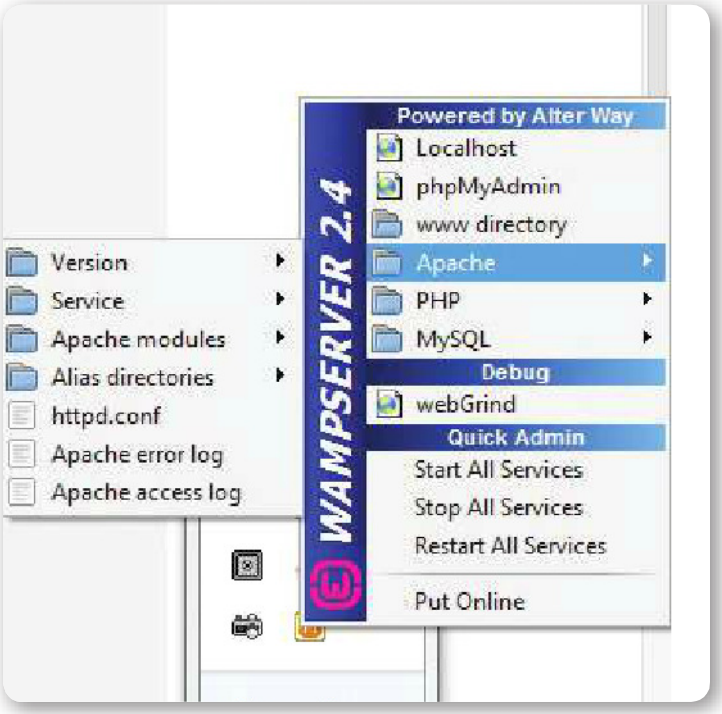


Figura 4. El acceso a las funcionalidades de WAMP es sencillo e intuitivo; podremos controlar todo lo relacionado con las herramientas de desarrollo web.

Finalizada la instalación, se ofrece la posibilidad de iniciar la aplicación. En el área de notificación del sistema (por defecto, en la parte inferior derecha) veremos un nuevo icono y, al hacer clic sobre él, accederemos a las opciones de WAMP. Explicaremos las principales:

- **Localhost**: nos da la posibilidad de acceder (a través del navegador seleccionado durante el proceso de instalación) a una página de inicio que contiene un listado con todos nuestros sitios activos. En un primer momento este directorio, lógicamente, estará vacío.
- **phpMyAdmin**: por medio del navegador, llegamos a esta aplicación que nos permite administrar el servidor MySQL, y así manipular las distintas bases de datos que utilizarán nuestros sistemas.

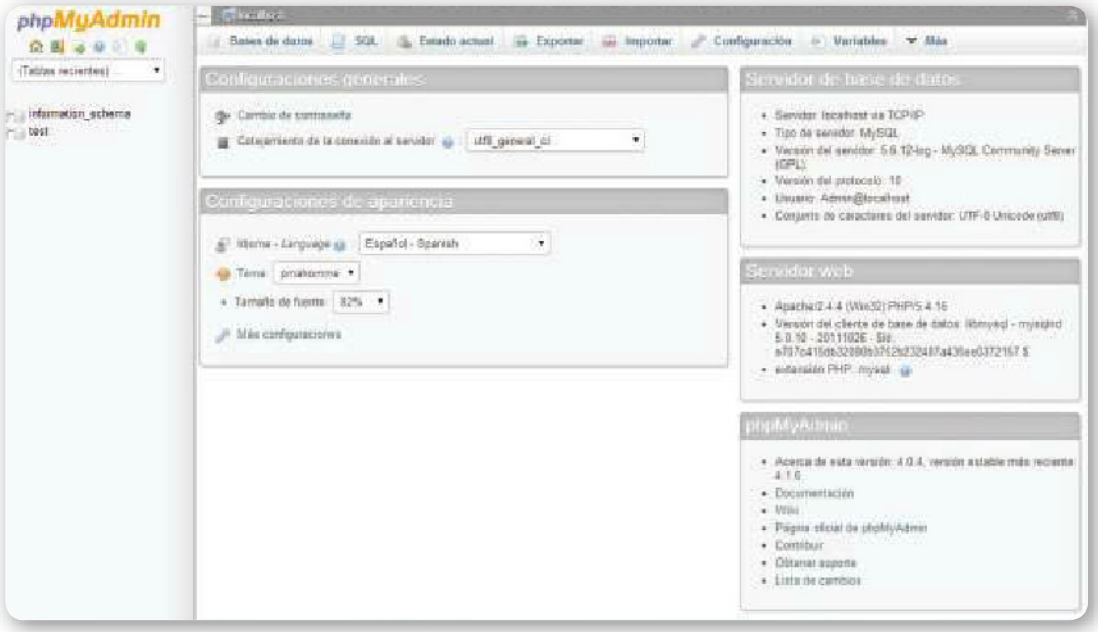


Figura 5. **phpMyAdmin** es una popular herramienta web que permite administrar servidores de bases de datos MySQL de manera gráfica.

- **wwwdirectory**: es un acceso directo a la carpeta **DocumentRoot** de nuestro sistema, que contiene todos nuestros sitios.
- **Log files** y **Config Files**: muestran una lista de los archivos correspondientes al registro de errores y a los archivos de configuración.

Daremos más información acerca de esto en los próximos apartados.

- **Apache modules** (módulos del servidor web Apache) y **PHP settings** (opciones y extensiones PHP): controlan características específicas y avanzadas de estas herramientas y habilitar o deshabilitar funcionalidades.
- **Alias directories**: genera accesos directos a determinados sitios locales, por ejemplo, **http://localhost/sitio**.

Dentro de la sección **Services**, en el mismo menú, contamos con opciones sobre el manejo de los distintos servicios. Al entrar en funcionamiento, tanto Apache como MySQL crean sus propios procesos, como **wampapache** (**httpd.exe**) y **wampmysqld** (**mysqld-nt.exe**), a los que se denomina **servicios**. Desde aquí podremos controlar cada servicio en particular (**Start/ Resume**, **Stop** y **Restart**) o todos en general (**StartAllServices**, **Stop AllServices**, **RestartAllServices**). Esto será de utilidad cuando modifiquemos alguna de las opciones de configuración de PHP (archivo **php.ini**; veremos más acerca de esto en los próximos apartados), ya que para que tengan efecto deberemos reiniciar el servidor web. En el sitio web oficial de WAMP, hay una sección de preguntas frecuentes para resolver cuestiones referidas a la configuración e instalación de la herramienta.

Crear una página

Ya estamos listos para aprender a crear páginas con el lenguaje de programación PHP. Para empezar, en cualquier editor de textos (**Notepad**, **Emacs**, **EditPlus**, **Notepad++**, etc.), ingresamos el siguiente contenido:

```
<?php

echo "Esta es mi primera página utilizando PHP !";

?>
```

En este momento, no debemos preocuparnos si no comprendemos el código anterior, simplemente lo que este hace es imprimir un mensaje y enviarlo a la salida del navegador. Veremos más profundamente la sintaxis utilizada por PHP en este capítulo y en los que vienen a continuación Como próximo paso, guardamos este archivo bajo el nombre **ejemplo.php** dentro del **DocumentRoot** (el directorio en el cual almacenaremos los sitios, recordemos que fue definido durante la instalación de WAMP y que, por defecto, es **c:\wamp\www**). Una vez hecho esto, verificamos que el servicio correspondiente a Apache esté activo (en caso de no estar seguros, podemos utilizar la opción **RestartAllServices**). Si accedemos desde cualquier navegador a la dirección **http://localhost/ejemplo.php**, observaremos la salida correspondiente; en este caso, un simple mensaje.

Administrar varios sitios web

En nuestro servidor local podemos tener más de un sitio, por eso se estila, para mantener un cierto orden y estructurar de manera clara los desarrollos, almacenar cada uno de ellos en un directorio propio, siempre bajo **DocumentRoot**. Veamos un ejemplo. Primero generamos las carpetas dentro de **c:\wamp\www**. Cualquier nombre es válido, en este caso, usamos **sitio1** y **sitio2**. Dentro de **sitio1** creamos un archivo al cual denominamos **index.php**, que tendrá el siguiente contenido:

```
<?php

$hora = date("H:i:s");
echo "Esta es la pagina de inicio del Sitio 1. La hora actual es $hora";

?>
```

Repetimos el mismo procedimiento en la carpeta **sitio2**. El archivo también denominado **index.php** tendrá el siguiente contenido:

```
<?php

$hora = date("H:i:s");
echo "Esta es la pagina de inicio del Sitio 2. La hora actual es $hora";

?>
```

La función **date**, que veremos en detalle en el **capítulo 4**, recibe un formato de fecha-hora (en nuestro ejemplo **hora:minutos:segundos** actuales). Una vez creados y guardados los archivos, podemos ingresar a nuestra página de inicio a través de WAMP (opción **localhost**) o simplemente escribiendo la siguiente dirección en la barra del navegador: **http://localhost/**. Allí encontraremos un listado con nuestros sitios disponibles, y será posible seleccionar cualquiera de ellos si queremos acceder a su contenido. Veremos que las direcciones son del tipo:

```
http://localhost/sitio1
http://localhost/sitio2
```

Definir la página principal del sitio

Otro aspecto para tener en cuenta es que, a diferencia de lo que ocurría en nuestro primer ejemplo (**ejemplo.php**), en el que debíamos indicar el nombre de la página al ingresar tanto a **sitio1** como a **sitio2**, ahora el navegador recupera y accede directamente a **index.php**. Esto ocurre porque hay una directiva (opción de configuración) dentro del archivo **httpd.conf** (puesto a disposición por el servidor web Apache) llamada **DirectoryIndex**, que nos permite definir qué archivos se buscarán por defecto cuando no se especifica uno en las URL escritas en la barra de direcciones del navegador.

La directiva **DirectoryIndexindex.php index.html index.htm** es configurable por parte del usuario. En caso de que no exista alguno de los archivos

indicados en **DirectoryIndex** dentro del directorio al que intentamos acceder, simplemente se mostrará un listado con todos los archivos de la carpeta. Por cuestiones de seguridad mínimas, si estos archivos son privados o si no es conveniente que cualquiera que ingrese a nuestro sitio los vea, será necesario ubicar un archivo índice. Otra opción importante es **DocumentRoot**, que permite definir el directorio raíz desde el cual acceder a los documentos desde un navegador:

```
DocumentRoot "C:/wamp/www"
```

Aparte de configurar el servidor desde el archivo **httpd.conf**, podemos hacerlo desde un fichero con el nombre **.htaccess**, que se puede encontrar dentro de cualquier directorio del **DocumentRoot**. En cada uno, es posible ubicar las directivas de configuración del **httpd.conf**; la diferencia reside en que los valores de las directivas que se encuentran en un fichero **.htaccess** prevalecen frente a los valores de configuración especificados en el fichero **httpd.conf**.

Incluir código PHP en documentos HTML

Hasta ahora, vimos con distintos ejemplos muy simples cómo encerrar código PHP e incluirlo en un archivo de texto plano. Repasemos los conceptos principales.

LAMP Y WAMP

Estas dos siglas tienen como fin identificar sistemas según las herramientas con las que cuenten: **LAMP** significa **Linux-Apache-MySQL-PHP/Perl/Python**, y **WAMP**, por su parte, **Windows-Apache-MySQL-PHP/Perl/Python**. Estas denominaciones aparecen de manera frecuente en distintos artículos o definiciones.

1. Crear o editar un archivo con la extensión **.PHP**. Esto varía con las versiones; actualmente podemos encontrarnos con documentos de extensión **.PHTML**, **.PHP4**, **.PHP3**, **.PHP5** (en este caso se indica que la página está programada utilizando características específicas de una versión y que eventualmente habrá conflictos si el servidor sobre el cual trabajamos no cuenta con ella). Más allá de esto, los documentos que generemos deberán contar con la extensión **.PHP**. Al respecto podemos señalar que la extensión del archivo es importante para que el servidor pueda reconocerlo y tratarlo de una manera determinada. Si observamos en detalle el archivo de configuración **httpd.conf** del servidor web Apache, podremos encontrar líneas como las siguientes:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php5
```

Este caso particular indica que, al encontrar una petición que invoque a un archivo de extensión **.PHP** o **.PHP5**, se pedirá al intérprete PHP que resuelva el código allí escrito. Este comportamiento puede aplicarse a otros tipos de archivo, no solo a los relacionados con PHP:

```
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
AddType text/html .shtml
```

Incluso en algunos sistemas está habilitada la opción de reconocer código PHP dentro de documentos de extensión **.HTM** o **.HTML**:

```
AddType application/x-httpd-php .htm .html
```

2. Encerrar el código entre etiquetas de apertura y cierre:

```
<?php

//Aquí se incluye el código PHP

?>

//Aquí no
```

Existen varias maneras de incluir código PHP. La más usual y generalmente admitida por la mayoría de las configuraciones es la vista anteriormente:

```
<?php

//código

?>
```

Pero también contamos con otras:

```
<?

//código

?>

<%

//código

%>
```

Esta última estará disponible solo si la directiva correspondiente en el **php.ini** está habilitada:

```
asp_tags = On
```

Para incluir código PHP en un documento HTML, simplemente tendremos que ubicar el código en el lugar que corresponda dentro de la estructura del documento:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> titulo del <?php echo 'documento'; ?></title>
</head>
<body>

<?php

$hora = date("H:i:s");
echo "<b> Esto es codigo generado por el preprocesador PHP.
La hora actual es $hora </b>";

 ?>

</body>
</html>
```

Luego, habrá que copiarlo a un directorio que esté bajo el **DocumentRoot** del servidor, e invocarlo desde un navegador web. Quedaría como el código siguiente:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> titulo del documento </title>
</head>
<body>
<b> Esto es codigo generado por el preprocesador PHP.
La hora actual es 20:16:42 </b>
</body>
</html>
```

En los próximos capítulos, veremos cómo avanzar, agregando más funcionalidades a nuestras páginas, a través de las posibilidades que nos brinda PHP como lenguaje de cabecera.

Extensiones del lenguaje

Las **extensiones** o **bibliotecas** son conjuntos de funciones disponibles para programar aplicaciones. Estas extensiones pueden estar incorporadas al lenguaje o ser añadidas de manera explícita.

Una vez que se instalan y se habilitan estas bibliotecas, el comportamiento de las funciones componentes dentro del código de nuestros programas es idéntico al de cualquier otra función o procedimiento. Es decir, la programación se vuelve independiente y transparente al origen de los procedimientos.

Desarrollaremos todo lo referido a la invocación y creación de funciones en el **capítulo 2**. Para habilitar o deshabilitar extensiones, podemos modificar el archivo **php.ini** o, desde WAMP, utilizar la opción **Configuración de PHP/ Extensiones de PHP**.

Ejemplo sobre funciones

Por ejemplo, si necesitamos contar con las funciones puestas a disposición por PHP para acceder a bases de datos MySQL, deberemos habilitar la extensión correspondiente:

```
extension=php_mysql.dll
```

Si está deshabilitada, observaremos algo similar a:

```
;extension=php_mysql.dll
```

Debemos aclarar que, en este caso, para acceder a un servidor de bases de datos a través de estas funciones, este tendrá que estar activo. Una cosa son las funciones para acceder, y otra es el servidor de bases de datos en sí.

Bibliotecas en PHP

PHP incorpora, sin necesidad de ningún tipo de instalación ni habilitación adicionales, una gran cantidad de bibliotecas, por lo cual contaremos con múltiples funciones para comenzar a desarrollar sitios profesionales.

Entre estas extensiones se encuentran:

CONTAREMOS CON MÚLTIPLES FUNCIONES PARA DESARROLLAR SITIOS PROFESIONALES

- Funciones para manejo de matrices.
- Funciones matemáticas.
- BCMath (desde PHP 4.0.4, más funciones matemáticas).
- Para manejo de clases/objetos.
- Para manejo de variables de tipo de carácter.
- Para tratamiento de fecha y hora.
- Para acceso directo a entrada/salida.
- Funciones de directorio.

- Funciones de gestión de errores y registros.
- Funciones de sistema de archivos.
- Para utilizar el protocolo FTP.
- Para utilizar el protocolo HTTP.
- Funciones de correo.
- Funciones de red.
- Funciones de control de salida.
- Para ejecución de programas.
- Funciones para el manejo de sesiones.
- Funciones de secuencia.
- Funciones de cadenas.
- Funciones URL.
- Funciones para manejo de variables.

Para que los cambios (en este caso, la habilitación o deshabilitación de las extensiones) tengan efecto, hay que reiniciar el servidor web.

Podemos ver qué bibliotecas tenemos activas en nuestro sistema si utilizamos la función **phpinfo** de la siguiente manera:

```
<?php phpinfo(); ?>
```

RESUMEN

Analizamos los puntos básicos que nos permitirán avanzar y asimilar la manera de generar aplicaciones web con la utilización de PHP en conjunto con otras tecnologías. Además, realizamos un breve recorrido introductorio por las características de este lenguaje, que nos servirá para, en los próximos capítulos, conocer las virtudes de la versión 5 de PHP. Este es un punto de partida para ambientarnos en las posibilidades de este lenguaje.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una aplicación web?
2. ¿Qué es una aplicación cliente-servidor?
3. ¿Cuál es la relación entre un servidor web, una base de datos y un lenguaje de programación?
4. ¿Cómo se llama el archivo de configuración del entorno PHP?
5. ¿Por qué PHP es considerado un lenguaje modular?

EJERCICIOS PRÁCTICOS

1. Inicie, detenga y reinicie el servidor web a través de WAMP.
2. Inicie, detenga y reinicie el servidor de bases de datos a través de WAMP.
3. Acceda al directorio **DocumentRoot** a través de WAMP.
4. Compruebe si el servidor web está activo.
5. Busque y cambie el nombre de la página principal de inicio en PHP, para ver qué sucede. Luego restaure el nombre original.

PROFESOR EN LÍNEA

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

*02

Sintaxis básica

En este capítulo, haremos un recorrido por las características principales de PHP como lenguaje de programación y nos adentraremos en el conocimiento de las herramientas que brinda para implementar aplicaciones, entre ellas: las instrucciones, las variables, las constantes, las expresiones, los comentarios y las funciones.

↗↗↗

Introducción

A continuación, veremos cuáles son algunas de las alternativas más habituales con las que contamos para escribir código PHP y para generar aplicaciones web robustas y a la vez versátiles. Al lo largo de los próximos apartados, profundizaremos en cada opción disponible.

Instrucciones

Las instrucciones en PHP finalizan con un **;** (punto y coma), también se reconoce como equivalente a la etiqueta de fin de bloque **?>** (signo de interrogación de cierre y signo mayor).
Un script PHP tiene las siguientes delimitaciones:

```
<?php
?>
```

Variables

Una **variable** es un lugar reservado para almacenar valores.
Cada una está identificada por un nombre antecedido por el signo **$** (peso). El nombre debe empezar con una letra o un guion bajo y puede continuar con cualquier cantidad de letras, guiones bajos o números.
Veamos algunos nombres válidos:

```
$_variable;
$var1
$var2
$var_3
$vAr
$VAR
$_variablE
```

Otros no válidos:

```
$1;
$666;
$-variable
$2variable
$'variable
$>variable
$<variable
```

Para asignar valores a las variables contamos con el operador = (igual):

```
$variable = 10;
$variable = 10.99;
$variable = "Cadena de caracteres";
$variable = 'Cadena de caracteres';
```

```
<?php
/***************************************************************************
 *                                index.php
 *                            -------------------
 *   begin                : Saturday, Feb 13, 2001
 *   copyright            : (C) 2001 The phpBB Group
 *   email                : support@phpbb.com
 *
 *   $Id: index.php,v 1.99.2.7 2006/01/28 11:13:39 acydburn Exp $
 *
 *
 ***************************************************************************/

/***************************************************************************
 *
 *   This program is free software; you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 ***************************************************************************/

define('IN_PHPBB', true);
```

Figura 1. Utilizar un editor que reconozca código PHP nos permitirá escribir aplicaciones de manera segura y encontrar errores sintácticos rápidamente.

Para imprimir valores de variables o, incluso, que no han sido asignados a variables en pantalla, contamos con algunas alternativas:

- La construcción **echo** recibe como argumentos variables o valores separados por comas y los muestra como salida:

```
$variable1 = "Hola ";
$variable2 = "a ";
$variable3 = "todos";

echo $variable1,$variable2;
echo $variable3;
```

- **Print** recibe como argumento una cadena de caracteres:

```
$variable = "Hola a todos";
print($variable);
```

El lenguaje PHP nos permite **modificar el nombre de las variables**, algo conocido como **variables variables**. Para lograrlo, debemos utilizar dos signos **$** en lugar de uno.

```
<?php

$tmp = 'nombreVariable';

$$tmp = 'Este es el contenido de la variable $tmp';

echo $nombreVariable;
// Este es el contenido de la variable $tmp

// o lo que es lo mismo: echo "${$tmp}";

?>
```

PHP incluye funciones que hacen posible saber a qué tipo pertenece una variable. Comencemos por **gettype**, que recibe como argumento una variable:

```
<?php

$var1 = 'Soy un cadena';
$var2 = 101;

echogettype($var1); //string
echogettype($var2); //integer

?>
```

Además, contamos con funciones para saber si una variable pertenece a un tipo específico o no. Todas devuelven **verdadero** si corresponden al tipo o **falso** en el caso contrario:

- **is_array**: para saber si se trata de un array.
- **is_float**: para saber si se trata de un flotante.
- **is_int**: para saber si se trata de un entero.
- **is_object**: para saber si se trata de un objeto.
- **is_string**: para saber si se trata de una cadena de caracteres.

FUNCIONES

En el **capítulo 4** de este libro, estudiaremos las funciones disponibles relacionadas con cada tipo de dato provisto por PHP. Estas nos permitirán aumentar la capacidad de nuestras aplicaciones y nos darán más herramientas para lograr los objetivos propuestos durante la planificación previa.

Constantes

Las constantes en PHP se generan a través de la función **define**, que recibe como argumentos un nombre y un valor. Si queremos acceder a este último, no hace falta incluir el signo **$** antes de su nombre:

```
<?php

define('PI', 3.14);
define('diasDeJulio', 31);

echo "El valor de PI es ".PI;
echo "Julio tiene  ".diasDeJulio." dias";

?>
```

Una constante, a diferencia de lo que sucede con una variable, no puede modificar su valor a lo largo del script. Como veremos a lo largo de los capítulos, PHP mantiene un gran número de constantes predefinidas.

Expresiones

PHP soporta el pre y el posincremento y decremento. Veamos algunos ejemplos para clarificar este tema:

```
<?php

$variable = 10;

echo $variable++;
//muestra el valor y luego lo incrementa en 1
```

```
echo ++$variable;
//incrementa en 1 el valor y luego lo muestra

echo $variable--;
//muestra el valor y luego lo decrementa en 1

echo --$variable;
//decrementa en 1 el valor y luego lo muestra

?>
```

Estas opciones nos permiten ir de uno en uno, pero, si quisiéramos, podríamos ir más allá y utilizar asignaciones como las siguientes:

```
<?php

$variable = 10;

$variable += 5; //suma 5 a $variable
$variable -= 5; //resta 5 a $variable
$variable *= 5; //multiplica por 5 a $variable
$variable /= 5; //divide por 5 a $variable

echo $variable; // 10

?>
```

En el ejemplo anterior, hemos sumado, restado, multiplicado y divido por 5. Otra manera de asignar es a través del operador condicional ternario. La sintaxis es la que presentamos a continuación:

```
$variable = condicion ? valor si es verdadera : valor si es falsa;
```

Veamos un ejemplo:

```
<?php

$tmp = 10;

$variable = $tmp >5 ? 'Es mayor a 5' : 'Es menor a 5';

echo $variable; // Es mayor a 5

?>
```

Además, tenemos la alternativa de las asignaciones múltiples, que van de derecha a izquierda:

```
<?php

$var = 10;

$var1 = $var2 = $var + 1;

echo $var1;

?>
```

Entre las expresiones de comparación más comunes contamos, entre muchas otras, con las siguientes opciones:

- **>** (mayor que)
- **>=** (mayor o igual que)
- **==** (igual que)

- **!=** (distinto que)
- **<** (menor que)
- **<=** (menor o igual que)

Para acceder al listado completo de operadores, podemos visitar **www.php.net/manual/es/language.operators.php**.

Una expresión se evalúa como verdadera si es verdadera o no es falsa. Según PHP, una expresión es falsa cuando es nula, vacía, contiene el valor cero, etc. Es importante recordar que las expresiones se aplican desde la más interna hasta la más externa, por ejemplo:

```
<?php

$var = 10;

echo funcion1(funcion2(funcion3(++$var)));

?>
```

El resultado impreso se logra incrementando **$var** en uno; sobre ese valor se aplica la función **funcion3**; sobre ese valor se aplica la función **funcion2** y, por último, sobre ese valor se aplica la función **funcion1**.

Más adelante, veremos otras opciones de funciones disponibles.

Además de las variables creadas por los usuarios, PHP provee ciertos **arrays** especiales a partir de los cuales podemos acceder a variables predefinidas. Los arrays disponibles son los siguientes:

- **$GLOBALS** nos permite acceder a cualquier variable del script actual desde cualquier ámbito. Veremos más acerca de estas variables cuando tratemos la definición de funciones en PHP.
- **$_SERVER** almacena variables definidas por el servidor web, por ejemplo, nombre de la página actual, nombre del servidor, dirección IP, etc.
- **$_GET** guarda las variables pasadas a través del método **GET**.

Como ejemplo, podemos citar los argumentos pasados por URL:

```
http://localhost:8080/ejemplos/index.php?a=1&b=2
```

Si tomamos como base la llamada anterior, desde **index.php** podríamos acceder a los valores de las variables al utilizar la siguiente sintaxis:

```
<?php

echo $_GET['a'];
echo $_GET['b'];

?>
```

Cuando enviamos un formulario y utilizamos el método **GET**, también podemos acceder a los valores de sus campos a través de este array:

```
<formmethod="GET" action="index.php"></form>
```

- **$_POST** guarda las variables pasadas a través del método **POST**. El caso más común es el envío de formularios:

```
<formmethod="POST" action="index.php"></form>
```

- **$_COOKIE** almacena información acerca de las cookies guardadas en la máquina del cliente. Las cookies son estructuras que las páginas almacenan en archivos de texto ubicados en la máquina del usuario y sirven para que los sitios puedan reconocer a los visitantes a través de la identificación de los valores contenidos.

- **$_FILES** guarda información sobre los archivos subidos por formulario. Mientras que a los campos tradicionales (**text**, **radio**, **checkbox**, **textarea**, etc.), accedemos a través de **$_POST**, **$_GET** o **$_REQUEST**; a los de tipo **file** accedemos al utilizar este array.

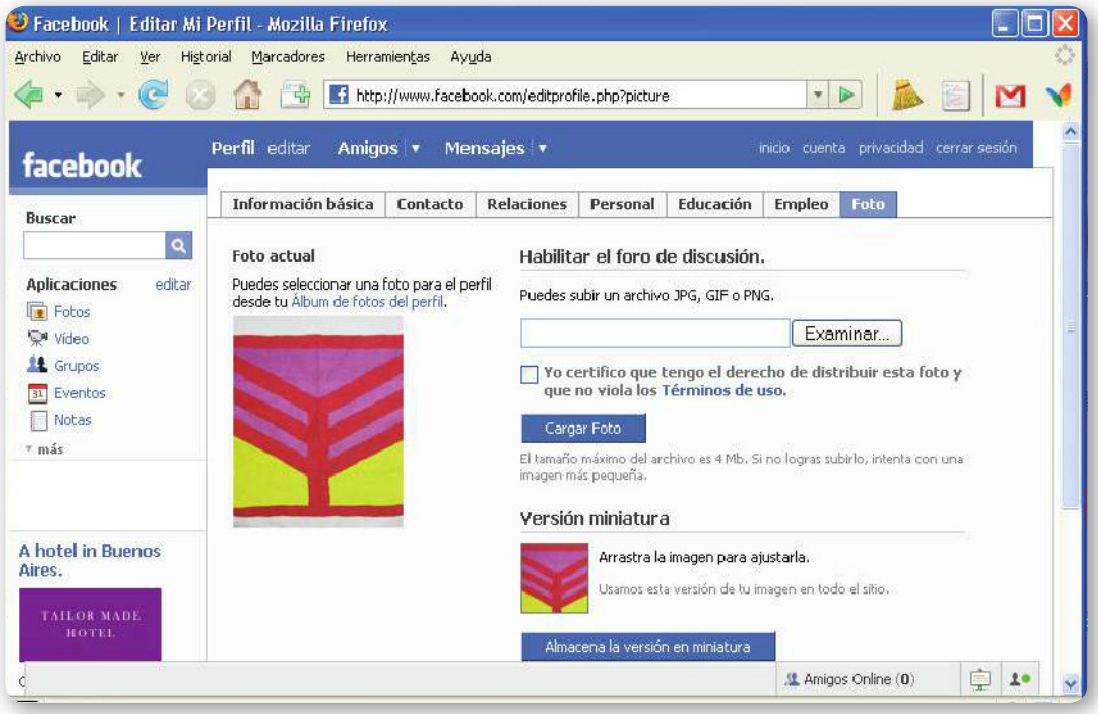


Figura 2. La carga de archivos forma parte de aplicaciones de cualquier tipo, dirigidas a toda clase de usuarios.

- **$_ENV** nos permite acceder a cierta información relacionada con la máquina desde la cual se ejecuta PHP, como por ejemplo, nombre del equipo, directorios especiales, sistema operativo, etc.
- **$_REQUEST** nos da la posibilidad de llegar a las variables contenidas en **$_GET**, **$_POST** y **$_COOKIES** (antes de **PHP versión 4.3.0**, la información de **$_FILES** también se incluía en **$_REQUEST**).
- **$_SESSION** almacena variables de sesión, que son las que mantienen su valor aun cuando el usuario salta de una página a otra. Un caso clásico es el del registro de usuarios: cuando un visitante ingresa sus datos para entrar a un área restringida de un sitio, lo hace solo una vez.

Si los datos son correctos, se inicializa una variable de sesión y se consultará su valor cuando el usuario intente ingresar en una de estas secciones.

Comentarios

Los comentarios se suelen utilizar para incluir textos aclaratorios sobre la composición del programa, que ayuden a quien está tratando de interpretarlo. También podemos comentar fragmentos de código para verificar el comportamiento de una aplicación. En PHP existen dos tipos de comentarios: los de una línea y los de múltiples líneas. En los de una línea contamos con dos posibilidades: utilizar la **barra doble** (//) o el **signo numeral** (#).
Ambos tienen el mismo efecto.

```
<?php

//echo "Ejemplo de comentario número (1).";

echo "Esto no esta comentado.";

#echo "Ejemplo de comentario número (2).";
# otro comentario.

?>
```

UTILIDAD DE LOS ARRAYS PREDISEÑADOS

Los arrays especiales están disponibles desde las últimas versiones de PHP y nos permiten mantener un control sobre el origen de los datos. Esto está relacionado con la seguridad de una aplicación, ya que intenta evitar posibles intrusiones o intentos de acceso no autorizados.

Como podemos observar, un comentario de este tipo tiene únicamente un delimitador en el comienzo: el término estará dado por el final de la línea (como vimos en el ejemplo anterior) o del bloque actual.

```
<?php //Aqui se termina el bloque actual ?>
```

Los comentarios de múltiples líneas tienen la siguiente sintaxis:

```
<?php

/*

Esto es un comentario
de múltiples
líneas

*/

?>
```

Esta alternativa puede emplearse para delimitar el comienzo y el final de un comentario, por ejemplo:

ORIGEN DE LOS TIPOS DE COMENTARIOS

El comentario de barra doble (**//**) de PHP tiene su origen en el lenguaje **C++**, mientras que, por su parte, el de múltiples líneas (**/**/**) proviene del lenguaje **C**. En el caso del signo numeral (**#**), podemos encontrarlo en la interfaz de comandos de **Unix**.

```
<?php

$a = 10;
$b = 10;
$c = 10;

$s = $a /* + $c */ + $b ;

echo $s;

?>
```

Todo lo que esté dentro de /* y */ será tomado como comentario:

```
<?php

/*

Abrimos el comentario

?>

Seguimos dentro del comentario

/* Incluso esto sigue formando parte del comentario

<?php

Finalmente cerramos el comentario

*/

?>
Este texto será impreso en pantalla
```

Lo único que no podemos incluir en esta clase de comentarios es el delimitador de cierre (***/**):

```
<?php

/* Abrimos el comentario

   */

Intentamos cerrar el comentario */

?>
```

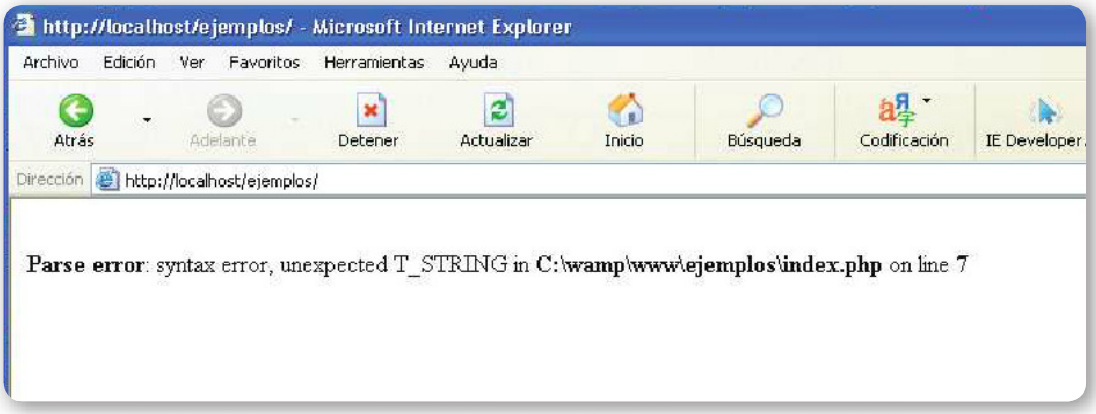


Figura 3. La mala implementación de comentarios puede provocar errores.

El **preprocesador PHP** no incluye el contenido de los comentarios en la salida generada, por lo cual solo podrán acceder al contenido de estos quienes puedan visualizar el código fuente de los documentos PHP.

Tipos de datos

PHP pone a disposición de los programadores distintos tipos de datos para definir el contenido de las variables. Podemos citar los siguientes.

Booleanos

El tipo booleano (**boolean**) puede tomar uno de dos valores: verdadero (**true**) o falso (**false**). Como vimos, PHP trata los valores de las variables según el contexto en el que se evalúen, y para el caso de los tipos booleanos las reglas para definir si un valor es **false** son las siguientes:

```
<?php

$variable = false;
//definición explícita

$variable = 0;
//valor numérico cero

$variable = "";
//cadena vacía

$variable = "0";
//cadena conteniendo el valor numérico cero

$variable = null;
//valor nulo

$variable = array;
//matriz sin elementos

?>
```

Una matriz es una colección de variables y, en caso de contar con cero elementos, se considera **false**. Estudiaremos los arrays en el **apéndice A**.

Numéricos

Entre los tipos de datos numéricos contamos con dos opciones: los enteros (**integer**) y los flotantes (**float**). En lo que respecta a los primeros, disponemos de una serie de notaciones en distintas bases: decimal (**base 10**), hexadecimal (**base 16**) y octal (**base 8**). Para saber cuándo utilizar una u otra, el intérprete verifica el comienzo del valor numérico:

```
<?php

$variable = 0123;
// Si empieza con cero es tomado como un octal

$variable = 0x1A;
// Si empieza con 0x (cero x) es tomado como un hexadecimal

$variable = 1234;
// Entero decimal

?>
```

Los enteros negativos se definen al anteponer el signo **–** (menos) al valor asignado a la variable:

```
<?php

$variable = -101;
// Entero decimal negativo

?>
```

Un tipo de dato **float** (también conocido como **punto flotante**, **doble** o **número real**) es un número con posiciones decimales:

```
<?php

$variable = 10.2;

?>
```

De acuerdo con la plataforma sobre la cual estemos trabajando, los límites en cuanto a los posibles valores de los tipos de datos numéricos pueden variar. Para los flotantes, normalmente, el máximo es de **~1.8e308**, y para los enteros contamos con las constantes **PHP_INT_SIZE** y **PHP_INT_MAX**:

```
<?php

echo 'Tamaño maximo de un entero: '.PHP_INT_SIZE;
echo '<br />Valor maximo de un entero: '.PHP_INT_MAX;

?>
```

En caso de sobrepasar estos valores, el entero será convertido automáticamente a flotante:

```
<?php

echovar_dump(PHP_INT_MAX);
//tipo integer

echo var_dump(PHP_INT_MAX+1);
//tipo float

?>
```

Cadenas de caracteres

Las cadenas de caracteres en PHP son asignadas a variables a través de dos maneras: encerradas entre comillas (simples o dobles) o por medio de la sintaxis **heredoc**. La diferencia entre utilizar **comillas simples** o **dobles** está dada por la llamada **expansión de variables**. Veamos un ejemplo:

```
<?php

$fecha = '21 de junio';
echo "El invierno comienza el $fecha.";

?>
```

El código anterior daría como respuesta la siguiente salida: **El invierno comienza el 21 de junio**. Pero si hubiéramos utilizado comillas simples:

```
<?php

$fecha = '21 de junio';
echo 'El invierno comienza el $fecha.';

?>
```

CICLOS

La utilización de **ciclos** dentro de los scripts nos permite, básicamente, escribir una menor cantidad de código y, en consecuencia, lograr rastrear errores de una manera rápida y clara. Además, nos brinda la posibilidad de modificar las aplicaciones sin tener que recurrir a tareas repetitivas.

La salida de respuesta a la utilización de estas comillas simples sería:
El invierno comienza el $fecha.

Con la sintaxis **heredoc**, podemos cargar en una variable todo el contenido ubicado entre los delimitadores. En el ejemplo, el delimitador es **SEPARADOR**:

```
<?php

$mensaje = <<<SEPARADOR
El invierno
comienza
el 21 de junio.
SEPARADOR;

echo $mensaje;

?>
```

Podemos incluir variables dentro del fragmento:

```
<?php

$fecha = '21 de junio';

$mensaje = <<<SEPARADOR
El invierno
comienza
el $fecha.
SEPARADOR;

echo $mensaje;

?>
```

Cuando PHP expande variables (al utilizar comillas dobles o la sintaxis anterior), lo que hace es buscar un carácter **$** (comienzo de variable) hasta el final del nombre, por ejemplo, un espacio, un punto y coma, dos puntos, otro signo **$** o cualquier carácter que dé a entender que el nombre de la variable ha terminado. Luego, reemplaza el nombre de la variable por el valor correspondiente. Si la variable no existe, la reemplaza por una cadena vacía. Hay una forma para delimitar manualmente el nombre de la variable: a través de la utilización de llaves. Veamos un ejemplo:

```
<?php

$var = 'perro';
$vars = 'gatos';

echo "Me gustan los ${var}s.";

?>
```

En este caso, PHP reemplaza **${var}** por el valor de la variable **var**. Si no se hubieran utilizado las llaves, se habría reemplazado por **vars:echo "Me gustan los $vars.";**. De todas maneras, siempre es posible concatenar cadenas y variables, ya sea con comillas dobles o simples.

La expansión de variables también vale para arrays y para objetos.

```
echo "Me gustan los ".$var."s.";
echo 'Me gustan los '.$var.'s.';
```

Matrices

Una **matriz** puede ser definida como una colección de valores, es decir, pares claves/valor. Hay dos maneras de generar arrays en PHP, y una de ellas es por medio de la construcción **array**:

```
$array = array('a' => '1', 'b' => '2', 'c' => '3');
```

El array del ejemplo anterior contiene tres posiciones. Podemos acceder a los valores de cada una a través de sus claves, por ejemplo:

```
echo $array['b']; //imprime 2
```

Las claves pueden ser de tipo entero o cadenas de caracteres. Al crear un array, podemos no especificar las claves, y en ese caso PHP les asigna números enteros consecutivos comenzando desde 0:

```
$array = array('febrero', 'marzo', 'abril');
echo $array[1]; //imprime marzo
```

Podemos modificar el valor de las posiciones:

```
$array = array('febrero', 'marzo', 'abril');
$array[1] = 'mayo';
echo $array[1]; //imprime mayo
```

También podemos asignar nuevas posiciones:

```
$array = array('febrero', 'marzo', 'abril');
$array[3] = 'mayo';
$array[4] = 'junio';
```

En cuanto a los valores utilizados, estos pueden ser de cualquier tipo, incluso, pueden ser otros arrays.

```
<?php

$mes = 'Febrero';

$meses = array('Abril', 'Mayo');

$array[1] = 'Enero';
$array[]  = $mes;
$array[]  = 'Marzo';
$array[]  = $meses[0];
$array[]  = $meses[1];
$array[]  = 'Junio';
$array[]  = 'Julio';
$array[]  = 'Agosto';
$array[]  = 'Septiembre';
$array[]  = 'Octubre';
$array[]  = 'Noviembre';
$array[]  = 'Diciembre';

echo $array[6]; //imprime Junio

?>
```

Explicaremos más acerca de las funciones para el tratamiento de arrays en el **apéndice A**, en el que haremos un recorrido por las diferentes extensiones disponibles en el lenguaje PHP.

Estructuras de control

PHP cuenta con construcciones que nos permiten realizar tareas variadas, por ejemplo, tomar una acción u otra dependiendo del valor de una expresión o repetir instrucciones un número **N** de veces.

A continuación, veremos algunas de las alternativas que nos permitirán componer scripts haciendo uso de las características básicas del lenguaje.

Condicional o secuencial

Comencemos con la construcción **if** (si), que recibe como argumento una expresión y evalúa si es cierta (**true**, distinta de 0, no vacía) o falsa (**false**, 0, vacía). La sintaxis utilizada es la siguiente:

```
<?php

if (expresion) {
  //instrucciones
}

?>
```

Las llaves son opcionales en caso de contar solo con una instrucción. Podemos evaluar más de una expresión si utilizamos la construcción **elseif**:

```
<?php

if (expresion1) {
  //instrucciones
} elseif (expresion2) {
  //instrucciones
} elseif (expresion3) {
  //instrucciones
} elseif (expresion4) {
  //instrucciones
}

?>
```

Para tomar una acción por defecto, en caso de que ninguna expresión evaluada sea cierta, contamos con **else**:

```
<?php

if (expresion1) {
  //instrucciones
} elseif (expresion2) {
  //instrucciones
} elseif (expresion3) {
  //instrucciones
} elseif (expresion4) {
  //instrucciones
} else {
  //instrucciones
}

?>
```

En estos casos, PHP evalúa las expresiones desde arriba hacia abajo, hasta encontrar una que sea verdadera y, si la encuentra, deja de evaluar las siguientes. En cada expresión podemos utilizar los operadores disponibles del lenguaje. En el siguiente ejemplo, verificamos a qué década pertenece un año:

```
<?php

$anio = 1962;

if ($anio >= 1900 && $anio < 1910) {
echo "Década del 1900";
} elseif ($anio >= 1910 && $anio < 1920) {
echo "Década del '10";
} elseif ($anio >= 1920 && $anio < 1930) {
echo "Década del '20";
} elseif ($anio >= 1930 && $anio < 1940) {
```

```
echo "Década del '30";
} elseif ($anio >= 1940 && $anio < 1950) {
echo "Década del '40";
} elseif ($anio >= 1950 && $anio < 1960) {
echo "Década del '50";
} elseif ($anio >= 1960 && $anio < 1970) {
echo "Década del '60";
} elseif ($anio >= 1970 && $anio < 1980) {
echo "Década del '70";
} elseif ($anio >= 1980 && $anio < 1990) {
echo "Década del '80";
} elseif ($anio >= 1990 && $anio < 2000) {
echo "Década del '90";
} else {
echo "El año no está dentro de los 19XX.";
}

?>
```

En una construcción de este tipo hay un **if**, ningún, un o más de un **elseif** y un o ningún **else**. Dentro de las instrucciones pueden aparecer otras construcciones **if elseif else**. Existe una sintaxis alternativa poco utilizada, pero válida:

```
<?php

if (expresion1):
  //instrucciones
  //instrucciones
elseif (expresion2):
  //instrucciones
  //instrucciones
```

```
elseif (expresion3):
  //instrucciones
  //instrucciones
else:
  //instrucciones
  //instrucciones
endif;

?>
```

Notamos la ausencia de llaves aun cuando hay más de una instrucción. La llave de apertura ({) es reemplazada por los dos puntos (:), y la de cierre (}), por la siguiente evaluación (**elseif**, **else**, o **endif**). El bucle **switch** es equivalente a un conjunto de sentencias de tipo **if ... elseif ... else**. Veamos un ejemplo:

```
<?php

$mes = 2;

switch ($mes) {
case 1:
echo "Enero";
break;
case 2:
echo "Febrero";
break;
case 6:
echo "Junio";
break;
default:
echo "No es ni Enero ni Febrero ni Junio";
}

?>
```

El valor para evaluar por cada case puede ser de cualquier tipo. Si un **case** coincide con el valor para evaluar y no contiene una definición, se pasa al siguiente:

```
<?php

$mes = 1;

switch ($mes) {
case 1:
case 2:
echo "Enero o Febrero";
break;
case 6:
echo "Junio";
break;
default:
echo "No es ni Enero ni Febrero ni Junio";
}

?>
```

Ciclos o estructuras repetitivas

Si en alguna situación tenemos la necesidad de ejecutar un mismo fragmento de código un número **N** de veces, contamos con diferentes alternativas. Una de ellas está dada por la construcción **while** (mientras), que recibe como argumento una expresión. Las instrucciones contenidas se ejecutan **mientras** la expresión sea verdadera. La sintaxis es la que presentamos a continuación:

```
<?php

while (expresion) {
```

```
  //instrucciones
}

?>
```

Como en el caso de **if**, si contamos con una sola instrucción, podemos omitir las llaves de apertura y cierre. En esta clase de estructuras será necesario modificar, en algún momento, el valor de la expresión, puesto que, si no lo hiciéramos, el ciclo no terminaría nunca.

```
<?php

$variable = 10;

while ($variable < 100) {
echo '<li> Esto no termina nunca.';
}

?>
```

En el siguiente ejemplo, modificamos valores para que la expresión sea falsa en algún momento:

```
<?php

$variable = 10;

while ($variable < 100) {
echo '<li> Vuelta #'.++$contadorVueltas;
  $variable++;
}

?>
```

En las estructuras repetitivas (como **while**, por ejemplo), se habla de **bucles** (ciclos) e **iteraciones** (vueltas). En el bucle **while** anterior, se producen noventa iteraciones. Puede suceder que la expresión sea falsa en todo momento (o por lo menos antes del inicio del ciclo) y que no se produzcan iteraciones:

```
<?php

$variable = 10;

while ($variable > 10) {
echo '<li> Esto nunca se ejecuta.';
  $variable++;
}

?>
```

Al igual que **if**, **while** también posee una sintaxis alternativa a la habitualmente utilizada, que es la siguiente:

```
<?php

$variable = 100;

while ($variable > 10):
echo '<li> Ejecutando sentencias ...';
  $variable--;
endwhile;

?>
```

Como vimos, **while** evalúa la expresión antes de ejecutar las instrucciones. También contamos con un bucle llamado **do while**, que nos permite ejecutar las instrucciones y luego verificar la expresión:

```
<?php

$variable = 10;

do {
echo '<li> Esto si se ejecuta al menos una vez.';
  $variable++;
} while ($variable > 100);

?>
```

Como vemos en el ejemplo anterior, estas construcciones hacen posible ejecutar las instrucciones contenidas por lo menos una vez.

Los bucles **for** están compuestos por tres expresiones que se encuentran separadas por un punto y coma:

```
<?php

for($c=0;$c<10;$c++) {
echo $c;
}

?>
```

La primera expresión se evalúa siempre, y la segunda se evalúa ante el comienzo de cada iteración: si es verdadera, se ejecutan las sentencias dentro del **for**, y al final se evalúa la tercera expresión. Eventualmente, las expresiones pueden omitirse siempre y cuando el bucle finalice en algún momento. La sintaxis alternativa es la siguiente:

```
<?php

for ($c=0;$c<10;$c++):
echo $c;
endfor;

?>
```

Los bucles de tipo **foreach** nos permiten recorrer un array:

```
<?php

$array[] = 'a';
$array[] = 'b';
$array[] = 'c';

foreach ($array as $clave => $valor) {
echo $array[$clave]; // o $valor
}

?>
```

Podemos omitir la clave y tomar directamente el valor:

```
<?php

$array[] = 'a';
$array[] = 'b';
$array[] = 'c';

foreach ($array as $valor) {
echo $valor;
}

?>
```

En los bucles de tipo for, while o switch, podemos utilizar break para interrumpir las iteraciones. Si tenemos bucles anidados (uno dentro de otro), podemos pasar a break un parámetro adicional para especificar cuántas estructuras hay que terminar:

```
<?php

for ($c=0;$c<10;$c++) {
echo "<br />c igual a $c";

for ($i=0;$i<10;$i++) {
        echo "<br />iigual a $i";

        if ($c == 5 && $i == 4) {
echo "<br />Fin: c = 5 e i = 4";
        break 2; //terminan los dos bucles
        }
 }
}

?>
```

En caso de no querer saltear solo la iteración actual y evitar que se ejecute el código dentro del bucle, contamos con **continue**:

```
<?php

for ($c=0;$c<10;$c++) {
if ($c > 2 && $c < 5)
continue;

echo "<br />c igual a $c";
}

?>
```

Inclusión de archivos

PHP cuenta con la posibilidad de incluir el contenido de archivos externos en el script actual. Las opciones con las que contamos son las siguientes:

- **require**: incluye el archivo pasado como argumento. Ante un error produce un error fatal.
- **include**: también incluye el archivo pasado como argumento, pero, ante un error, produce solo una advertencia.
- **require_once**: es similar a **require**, pero incluye el archivo solamente una vez, aunque intentemos hacerlo más de una.
- **include_once**: es semejante a **include**, pero incluye el archivo solamente una vez, aunque intentemos hacerlo más de una.

```
<?php

// código

include 'docs/pagina3.php';
require_once 'docs/pagina2.php';

// código

require 'docs/pagina1.php';

?>
```

Todas las variables, constantes, funciones, etc., que hayamos definido en el archivo incluido, podrán ser utilizadas en el archivo actual.

Funciones

PHP mantiene dos clases de funciones: las incorporadas y las definidas por el usuario; ambas trabajan de manera idéntica. Tienen la siguiente estructura:

```
<?php

function nombreFuncion($arg1, $arg2, $argN) {
 // código de la función
return $resultado;
}

?>
```

El número de argumentos y su tipo es definido por el autor de la función. Con **return** devolvemos el valor generado por la función, que puede ser de cualquier tipo. Una función puede o no devolver valores. Para invocar una función, escribimos su nombre y asignamos valores a los argumentos:

```
<?php

$var = saludo('Charles', 'Chaplin');
echo saludo('Groucho', 'Marx');

function saludo($nombre, $apellido) {
 $resultado = "Bienvenido $nombre $apellido !";
return $resultado;
}

?>
```

Las funciones pueden definirse e invocarse desde cualquier lugar del script. Si definimos más de una función con el mismo nombre, obtendremos un mensaje de error. Desde el interior de una función podemos invocar otras.

Hay dos maneras en las que se pueden pasar los argumentos a funciones: por valor (la opción por defecto) y por referencia, si anteponemos un **ampersand** (**&**) al argumento en la definición de la función. La diferencia es que la variable original no se modifica por valor, pero sí por referencia.

```
<?php

$numero = 10;

procesarPorValor($numero);

echo $numero; //10

procesarPorReferencia($numero);

echo $numero; //11

function procesarPorValor($numero) {
return $numero+=1;
}

function procesarPorReferencia(&$numero) {
return $numero+=1;
}

?>
```

Para convertir los argumentos en opcionales, podemos asignarles valores por defecto:

```
<?php

echo saludo('Groucho');
echo saludo('Charles', 'Chaplin');

function saludo($nombre, $apellido = 'Marx') {
return "Bienvenido $nombre $apellido !";
}

?>
```

Tenemos **variables variables** y también tenemos **funciones variables**:

```
<?php

$funcion = 'saludo';
echo $funcion('Groucho');

function saludo($nombre, $apellido = 'Marx') {
return "Bienvenido $nombre $apellido !";
}

?>
```

No podemos acceder a las variables definidas dentro de una función desde fuera de ellas, y viceversa. Para tomar variables externas a una función desde el interior de su definición, podemos anteponer **global** a las variables:

```
<?php

$a = $b = 10;

echo nombreFuncion();

function nombreFuncion() {
global $a, $b; //a y b se busca fuera de la función
return $a * $b;
}

?>
```

RESUMEN

Analizamos cuáles son las características generales más importantes de PHP como lenguaje: sus opciones, su sintaxis, las alternativas que ofrece y la manera de incluirlas en los scripts.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son los tipos de datos disponibles?
2. ¿Para qué se utiliza la construcción **echo**?
3. ¿Qué es un array?
4. ¿Qué es un ciclo?
5. ¿Qué diferencia hay entre **elseif** y **else**?

EJERCICIOS PRÁCTICOS

1. Desarrolle las siguientes aplicaciones: que reciba dos números y devuelva el mayor.
2. Que reciba una cadena de caracteres y devuelva el número de vocales de ella.
3. Que muestre un texto por pantalla diez veces.
4. Que, sobre la base de un número ingresado en pantalla (del 1 al 12), devuelva un string con el mes equivalente.
5. Modifique la aplicación del punto 4 para controlar que el número ingresado no sea menor a 1 ni mayor a 12.

PROFESOR EN LÍNEA

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

*03

Orientación a objetos

En este capítulo, analizaremos el paradigma orientado a objetos, implementado a través de PHP, una técnica que, utilizada en múltiples desarrollos, es preciso conocer para beneficiarnos de las capacidades del lenguaje. Haremos una introducción al paradigma, conoceremos los pilares del modelo y, luego, veremos su implementación en PHP.

Introducción

Lo primero que debemos decir es que la orientación a objetos no es una técnica propia de PHP, sino una implementación de un paradigma, solo una entre tantas.

A diferencia de otros lenguajes, en PHP, la incorporación de la orientación a objetos no fue sólida desde el principio, sino que fue moldeándose a medida que surgían las nuevas versiones. Recién en la versión 5, se alcanzó el grado de madurez necesario para que los programadores pudieran aplicar sus conocimientos de la **POO** (programación orientada a objetos) al lenguaje.

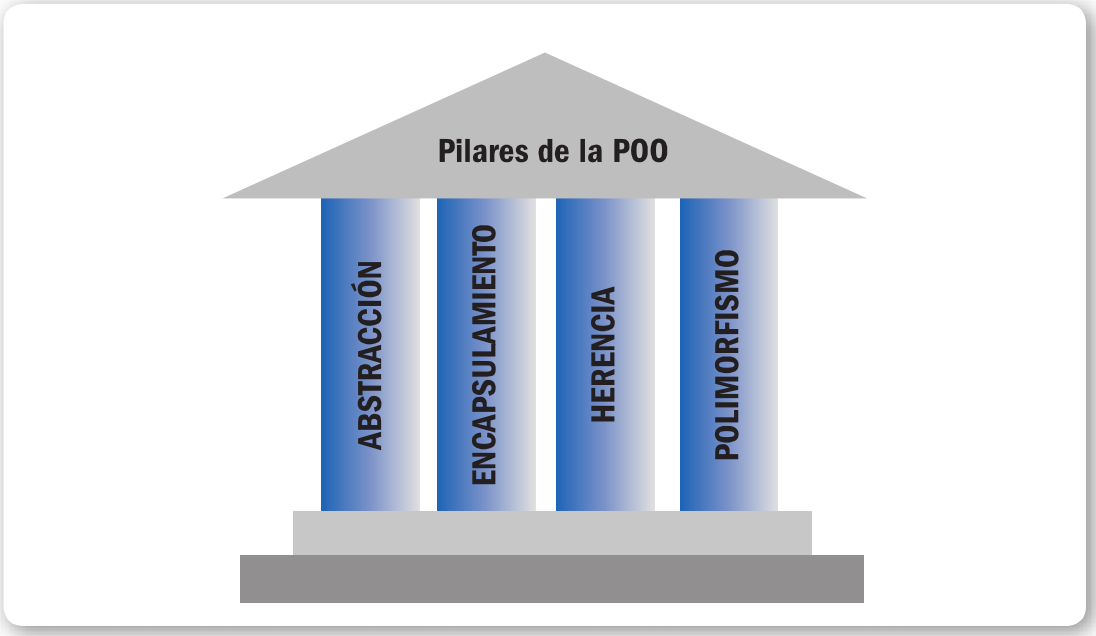


Figura 1. Pilares fundamentales de la programación orientada a objetos.

Pilares del modelo

Si observamos otros lenguajes con historia en este campo (como por ejemplo, **Java**, **C#** y **Python**), notaremos que los desarrolladores a cargo de la ampliación de PHP comenzaron a idear y plasmar la orientación a objetos de manera gradual. Cada paso llevó su tiempo, pero los resultados están a la

vista, ya que se ha logrado una evolución que permite, a quien quiera desarrollar con esta técnica, hacerlo sin problemas contando con la mayoría de sus características y cualidades.

Un punto importante es que la POO es un modo de llevar adelante un desarrollo. Como todo, tiene sus ventajas y desventajas, pero es una alternativa a la que tenemos la libertad de acceder. Esto siempre queda a nuestro criterio según nuestras necesidades. Conocer sus principios básicos nos ayudará a comprender e, incluso, a interactuar con aplicaciones ya desarrolladas que utilizan este paradigma. Nos referimos a cualquier tipo de aplicación: desde las más simples hasta las más complejas. Podemos obtener más información en el sitio oficial de PHP, en la sección **www.php.net/oop5**, donde se enumeran las características de la POO en las versiones más actuales del lenguaje.

Implementación en PHP

A continuación, veremos cómo implementar la orientación a objetos desde PHP. Tomaremos como base las versiones más recientes del lenguaje.

Clases, propiedades y métodos

Podríamos definir una **clase** como una plantilla o molde desde el cual generar objetos, es decir, a partir de un esquema con características bien

MÓDULOS

Debemos saber que una clase puede ser instanciada dentro de otra, algo que nos permite vincular y modular aplicaciones de manera sencilla. Pensemos en este hecho como una forma de dividir la carga de trabajo: una clase puede ocuparse de una tarea específica, y las demás, de otras.

establecidas, es posible crear un número **N** de instancias. Veámoslo de la siguiente manera: contamos con una clase **gato** que podría llegar a tener distintas instancias (**Garfield**, **Tom**, **Silvestre**, etcétera). La ventaja de esto es que la clase es una abstracción de las instancias particulares, y podemos definir solo una estructura de lugar de **N**. Cada instancia se diferenciará de las demás a partir de la modificación de las características disponibles en la clase original.

La estructura de una clase contiene dos elementos básicos: **variables** y **funciones** (declaradas mediante la palabra reservada **function**). La definición de una clase dentro del código PHP se reconoce al iniciar una línea con la palabra reservada **class**. Veamos un ejemplo:

```
<?php

class nombreClase {
  // definición de la clase nombreClase ...
}

?>
```

Atributos de las clases

Cada clase puede contar con sus propios **atributos**, por ejemplo, un gato tiene un nombre, una raza, una edad, etcétera. Estos atributos o propiedades se definen, dentro de la clase, de la siguiente manera:

```
<?php

class nombreClase {
  public $nombreVariable1;
  public $nombreVariable2;
  public $nombreVariableN;
}

?>
```

La palabra que antecede al nombre de la propiedad (en el ejemplo, **public**) indica el tipo de acceso a ella. Más adelante trataremos este tema.

Además, una clase puede tener **métodos**, que definimos como una función asociada a una clase. Continuando con nuestro ejemplo, un gato contendría métodos tales como **calcularFechaNacimiento** u **obtenerNúmeroDeHijos**. Para definir un método, usamos la siguiente sintaxis:

```
<?php

class nombreClase {
  public $nombreVariable;

public function nombreFuncion() {
    return $this->nombreVariable;
}
}

?>
```

Vemos que un método, incluso, utiliza la palabra reservada **function**, es decir, la misma que usamos para definir algunas funciones comunes, fuera del ámbito de una clase.

También encontramos una nueva palabra: **$this**, variable reservada especial que contiene al objeto actual y se utiliza en el ámbito de una función definida dentro de una clase. Sirve para referenciar otras funciones o variables dentro de la misma clase. Otra opción es utilizar **self**.

Continuando con el ejemplo anterior, veamos cómo crear instancias de una clase a través de PHP:

```
<?php

class nombreClase {
  public $nombreVariable;
```

```
public function nombreFuncion() {
return $this->nombreVariable;
}
}

$nombreInstancia = new nombreClase();

?>
```

```
<?php

class nombreClase {
  public $nombreVariable;

public function nombreFuncion() {
return $this->nombreVariable;
}
}

$nombreInstancia1 = new nombreClase();
$nombreInstancia1->nombreVariable = 'valor 1';

$nombreInstancia2 = new nombreClase();
$nombreInstancia2->nombreVariable = 'valor 2';

?>
```

Prestemos atención a la sintaxis utilizada para asignar un valor determinado a una propiedad:

```
$nombreInstancia1->nombreVariable = 'valor 1';
```

Además, como vemos, es posible acceder a las propiedades disponibles, tanto a partir de una instancia de una clase como de la definición de la clase.

Ahora veamos cómo invocar métodos de una clase con la utilización de las funcionalidades que PHP pone a nuestra disposición:

```
<?php

class nombreClase {
  public $nombreVariable;

public function nombreFuncion() {
return $this->nombreVariable;
}
}

$nombreInstancia = new nombreClase();
$nombreInstancia->nombreVariable = 'valor 1';

echo $nombreInstancia->nombreFuncion();

?>
```

Debemos notar que el método **nombreFuncion** devuelve el valor actual de la propiedad **nombreVariable**:

```
return $this->nombreVariable;
```

El cual mostramos por pantalla:

```
echo $nombreInstancia->nombreFuncion();
```

Permisos de acceso

Pasemos a ver, ahora, cómo funcionan los permisos de acceso y cuál es la utilidad de cada uno, aplicados tanto a propiedades como a métodos:

```
<?php

class nombreClase {
  protected $nombreVariable;

public function nombreFuncion() {
    return $this->nombreVariable;
}
}

$nombreInstancia = new nombreClase();
$nombreInstancia->nombreVariable = 'valor 1';

?>
```

Lo anterior genera un error, ya que estamos intentando acceder (asignar un valor) a una propiedad protegida. Observemos el código siguiente:

```
<?php

class nombreClase {
  private $nombreVariable;

  public function nombreFuncion() {
    $this->nombreVariable = 'valor 1';
return $this->nombreVariable;
}
}

$nombreInstancia = new nombreClase();
echo $nombreInstancia->nombreFuncion();

?>
```

Con **private**, la propiedad o método solo es accesible desde la clase a la que pertenece. Podemos modificar el código anterior utilizando otros modificadores, como **public** o **protected**, para observar su comportamiento.

Los métodos, al igual que cualquier función común y corriente, también reciben argumentos. Veamos un ejemplo acerca de esto:

```
<?php

class nombreClase {
  public $nombreVariable;

  public function nombreFuncion($mensaje, $num) {
    return $mensaje.'\'.$this->nombreVariable * $num;
  }
}

$nombreInstancia = new nombreClase();
$nombreInstancia->nombreVariable = 100;

echo $nombreInstancia->nombreFuncion('El resultado es', 3);

?>
```

LECTURA Y ESCRITURA

Es evidente que no tenemos la obligación de desarrollar aplicaciones con la utilización de la orientación a objetos, pero, más allá de esto, se trata de un tema con el que debemos familiarizarnos para comprender y poder obtener provecho de una gran cantidad de scripts escritos de esta manera. La programación orientada a objetos cada vez se halla más difundida y no podemos desconocerla.

Constantes

También es posible definir constantes dentro de una clase; veremos que la sintaxis es diferente a la presentada en el **capítulo 2**, ya que se utiliza la palabra reservada **const**. A diferencia de las propiedades, no se les antepone el carácter **$** y siempre son de acceso público:

```
<?php

class nombreClase1 {
const nombreConstante = 'xxx';

  //...
}

echo "<li>".nombreClase1::nombreConstante;

?>
```

Constructores y destructores de clases

Anteriormente, vimos cómo crear una instancia a través de la palabra reservada **new**. Cada vez que se crea una instancia de una clase, se ejecuta la función **constructor**, cuyo nombre es **__construct**, que podemos redefinir para modificar su comportamiento por defecto:

```
<?php

class nombreClase {
  public $nombrePropiedad;

  function __construct() {
    $this->nombrePropiedad = 10;
  }
}

$c = new nombreClase();
```

```
echo $c->nombrePropiedad;

?>
```

Si ejecutamos el código anterior, notaremos que al crearse la clase, instanciarla y llamar a su propiedad, devuelve el valor que iniciamos en el constructor de la clase. Por motivos de compatibilidad, debemos tener en cuenta que, hasta la versión 5 de PHP, el nombre de la función constructora debía coincidir con el de la clase. Esto sigue teniendo validez en caso de que no definamos explícitamente la función **__construct**.

Por otro lado, disponemos de los **destructores**, funciones que se encargan de realizar las tareas que necesitamos ejecutar cuando un objeto ya no es referenciado por ninguna variable, antes de liberar la memoria de manera definitiva. El nombre de esta función es **__destruct**, y podemos redefinirlo de la siguiente manera:

```
<?php

class nombreClase {
  public $nombrePropiedad;

  function __construct() {
    $this->nombrePropiedad = 10;
  }

  function __destruct(){
    echo "Terminando ... objeto destruido de memoria";
  }
}

$c = new nombreClase();

?>
```

En el ejemplo anterior, al terminar el script, PHP libera la memoria y destruye las variables locales, entre ellas **$c**, por lo cual se ejecuta el destructor.

Como vimos, el método destructor se invoca al final de un script, pero puede ejecutarse explícitamente si eliminamos la instancia antes:

```
<?php

class nombreClase {
  public $nombrePropiedad;

  function __construct() {
    $this->nombrePropiedad = 10;
  }

  function __destruct(){
    echo "Terminando ... objeto destruido de memoria ";
}
}

$c = new nombreClase();

unset($c);

echo "<br /> Ahora si, termina el programa ...";

?>
```

ÚLTIMOS AVANCES

Las últimas versiones del lenguaje PHP introdujeron mejoras notables en diversos aspectos; entre ellos, uno de los más importantes y necesarios fue el de la implementación de la orientación a objetos (POO). Este era, sin duda, un punto muy esperado por la enorme comunidad de usuarios de PHP.

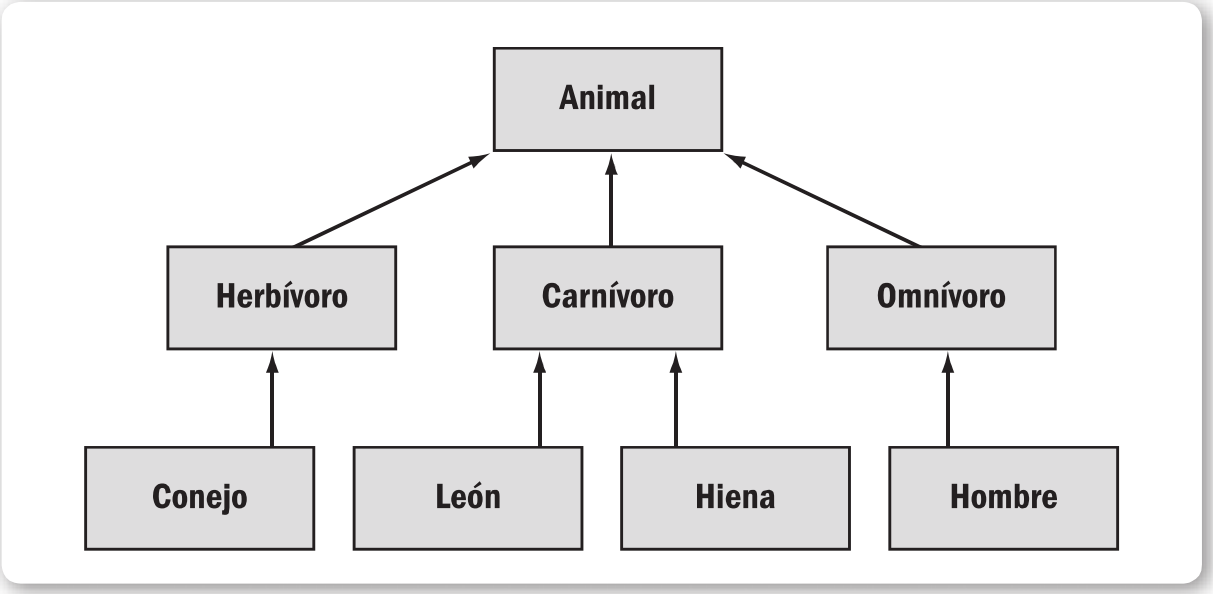


Figura 2. La creación de una clase principal nos permite crear otras subclases.

Errores en clases, métodos y propiedades

Si intentamos invocar un método inexistente, por defecto, recibiremos un mensaje de error. Para capturarlo, existe el método especial **__call**, que se ejecuta de manera automática en esos casos y recibe como argumentos el nombre del método y los parámetros pasados:

```
<?php

class nombreClase {
  private $nombrePropiedad;

  function __call($metodo, $atributos) {
    echo "Error: el método '$metodo' no está definido.";
  }
}

$c = new nombreClase();
$c->nombreMetodo();

?>
```

El método **__set** se invoca en caso de intentar asignar valores a propiedades no declaradas:

```
<?php

class nombreClase {
  private $nombrePropiedad;

private function __set($propiedad, $valor) {
echo "Error: no es posible modificar '$propiedad'.";
  }
}

$c = new nombreClase();
$c->variableNoExistente = 100;

?>
```

Por su parte, **__get** se invoca en caso de intentar recuperar valores de propiedades no declaradas:

PALABRAS RESERVADAS

Es importante tener en cuenta que todos los identificadores que comienzan con __ (doble guión bajo) se encuentran reservados por el lenguaje PHP para uso interno. Por esta razón, debemos tratar, en lo posible, de no utilizarlos en nuestras propias definiciones para evitar que surjan conflictos con el código que escribimos. En su lugar, aconsejamos usar cualquier otro carácter para comenzar los identificadores.

```
<?php

class nombreClase {
  private $nombrePropiedad;

  private function __get($propiedad) {
    echo "Error: no es posible leer '$propiedad'.";
  }
}

$c = new nombreClase();
echo $c->variableNoExistente;

?>
```

Los métodos **__set** y **__get**, en caso de estar definidos, serán invocados cuando se intente acceder o modificar el valor de alguna propiedad no declarada en la clase.

El método **__call** (como **__set** y **__get**) puede ser utilizado para sobrecargar métodos, permite capturar y controlar todas las funciones que queremos sobrecargar, definiendo cada una de ellas mediante condiciones dentro de **__call**.

En cuanto a las clases, contamos con la función especial **__autoload**, que, en caso de ser definida de manera explícita, será invocada automáticamente al intentar utilizar clases que no han sido declaradas:

```
<?php

function __autoload($nombreClase) {
  echo "La clase '$nombreClase' no está definida.";
  exit;
```

```
}

$c = new claseInexistente();

?>
```

Modificadores de acceso

Tanto las propiedades como los métodos admiten los llamados **modificadores de acceso**, entre los que se cuentan **public**, **private** y **protected**. Veamos el significado de cada uno de ellos:

SIGNIFICADO DE LOS MODIFICADORES DE ACCESO

▼ MODIFICADOR	▼ DESCRIPCIÓN
public	La propiedad o método es accesible desde cualquier ámbito, incluso desde otras clases.
private	La propiedad o método solo es accesible desde la clase a la que pertenece.
protected	La propiedad o método solo es accesible desde la clase a la que pertenece o desde cualquier clase que derive de ella.

Tabla 1. Modificadores de acceso disponibles para las propiedades y métodos de una clase.

```
<?php

class Ejemplo {

        public $variableA;
```

```
        protected $variableB;
        private $variableC;

        function __construct() {
                $this->variableB = 10;
        }

        public function valorB() {
                return $this->variableB;
        }
}

$temp = new Ejemplo();

echo $temp->valorB();

?>
```

Estos modificadores son comunes en la programación orientada a objetos, sobre todo, para implementar la encapsulación. La **encapsulación** consiste en organizar datos y métodos de una estructura, evitando su acceso por cualquier otro medio. Garantiza la integridad de los datos que contiene un objeto.

Hasta la versión 5 de PHP, todas las variables y las funciones eran públicas, y se declaraban con la palabra reservada **var**. Por cuestiones de compatibilidad, esta opción sigue disponible para las propiedades y se considera un sinónimo de **public**.

```
var $nombrePropiedad;
```

Los métodos, por su parte, no contaban con ningún modificador de acceso disponible; únicamente se definían mediante un nombre y un conjunto de argumentos.

Propiedades y métodos estáticos

Además de los modificadores de acceso, una opción para transformar el comportamiento de las propiedades y de los métodos es **static**, que nos permite mantener los valores más allá de una instancia en particular:

```
<?php

class nombreClase {
  private static $nombrePropiedad = 10;

public static function nombreMetodo() {
return nombreClase::$nombrePropiedad;
  }
}

echo nombreClase::nombreMetodo();

?>
```

Notemos que, en el ejemplo anterior, no hay una instancia de **nombreClase**. Lo mismo sucede en el caso de los métodos de objeto y de clase:

```
<?php

class Ejemplo {

        static function mostrarValor() {
                //...
                //...
                //...
        }
}

?>
```

El acceso a los atributos de clase se hace a través del operador **::** y, para hacerlo, no es necesario instanciar la clase:

```
<?php

class Ejemplo {

        static $propiedad = 10;

}

echo Ejemplo::$propiedad;

?>
```

Lo mismo, para los métodos de clase:

```
<?php

class Ejemplo {

        static $propiedad = 10;

        static function mostrarValor() {
                echo Ejemplo::$propiedad;
        }
}

Ejemplo::mostrarValor();

?>
```

Debemos tener en cuenta las siguientes consideraciones:

- No se puede acceder a un atributo de objeto sin instanciar la clase.
- No es posible acceder desde un objeto a los atributos estáticos de la clase.
- No se puede acceder a un atributo no estático desde un método estático.

Se pueden aplicar modificadores de acceso a las propiedades y métodos de clase, como podemos observar en el siguiente ejemplo:

```
<?php

class Ejemplo {

        private static $propiedad = 10;

        public static function mostrarValor() {
                echo Ejemplo::$propiedad;
        }
}

Ejemplo::mostrarValor();

?>
```

HERENCIA MÚLTIPLE

El uso de interfaces suple la falta de herencia múltiple. El lenguaje PHP no soporta esta característica: una clase derivada solamente puede tener una clase base. Sin embargo, sí es posible que una misma clase implemente varias interfaces. De esta manera, es posible hacer uso de la herencia múltiple.

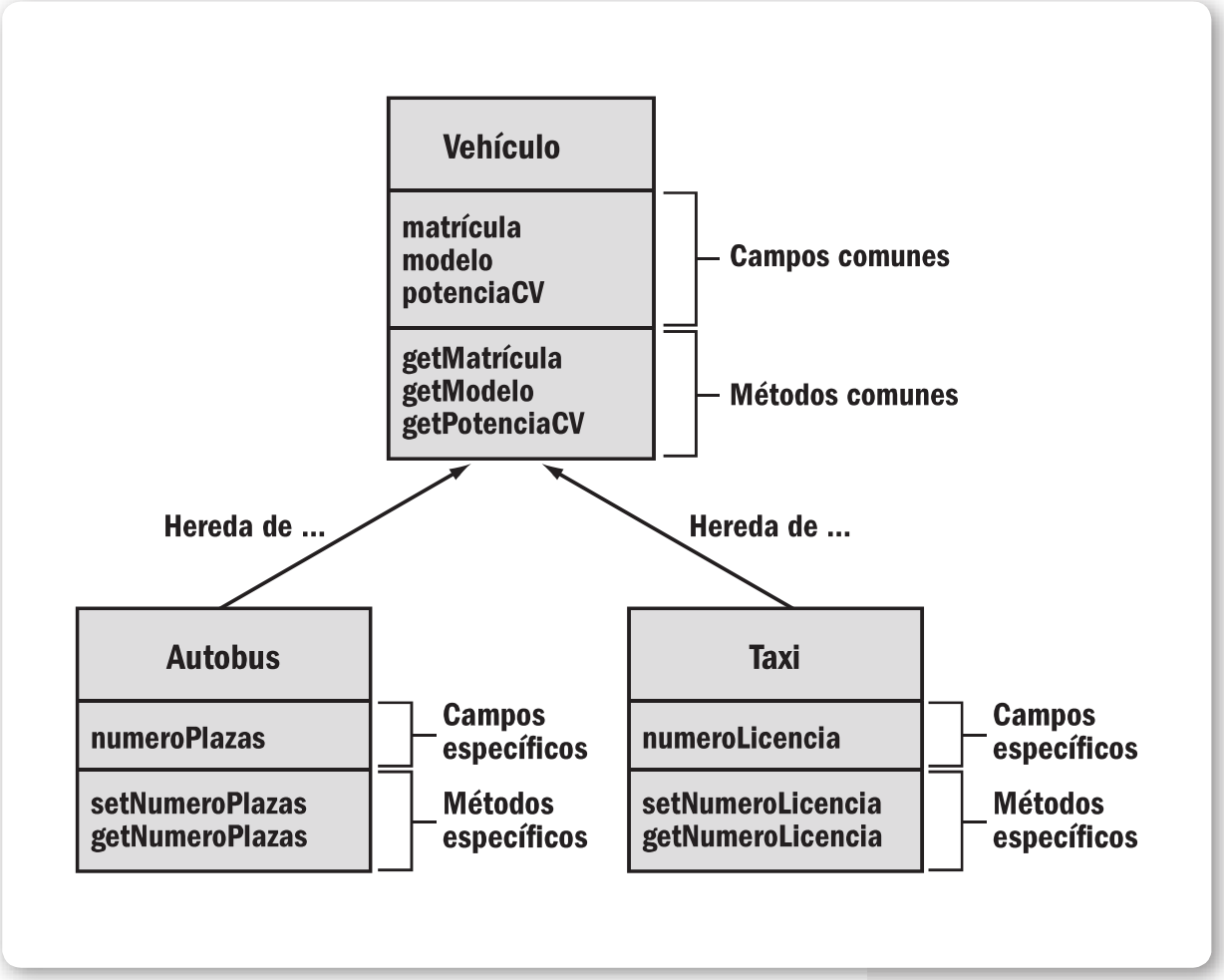


Figura 3. En esta imagen, podemos ver la herencia al crearse dos subclases desde **`vehículo`**.

Herencia en PHP

Este es un concepto muy importante en la POO y lo encontraremos de modo recurrente en los distintos desarrollos escritos bajo este paradigma. Básicamente, consiste en que, cuando una clase (una estructura especial que define características de un objeto) deriva de otra, de manera automática hereda sus propiedades (característica, atributo) y métodos (función asociada a una clase). En su propia definición puede modificar valores y redefinir métodos.

Para que una clase descienda de otra, se usa la palabra reservada **extends**:

```
<?php

class nombreClase1 {
  //...
}

class nombreClase2 extends nombreClase1 {
  //...
}

?>
```

La ventaja de esto es que la clase que extiende (**nombreClase2**) hereda y dispone de todos los métodos de la clase base (**nombreClase1**). Como dijimos, incluso puede redefinirlos y modificar su comportamiento. Por supuesto, también es posible que tenga sus propios métodos. Existe una manera de evitar la redefinición de un método de una clase, y es a través del modificador **final**:

```
<?php

class nombreClase1 {
  final function nombreMetodo() {
    //...
  }
}

?>
```

Clases abstractas

Otro modificador de acceso es **abstract**, que permite definir clases y métodos como abstractos. Las clases abstractas son las que no pueden ser instanciadas, cuyo fin es servir de base para otras clases (que sí podrán ser instanciadas).

```
<?php

abstract class Animal {
	public $nombre;

	function mostrarNombre() {
		return $this->nombre;
	}
}

class Gato extends Animal {
	public $ratonesCazados;

	function mostrarRatones() {
		return $this->ratonesCazados;
	}
}

class Perro extends Animal {
	public $gatosCazados;

	function mostrarGatos() {
		return $this->gatosCazados;
	}
}

$temp = new Gato();
$temp->nombre = 'felix';
$temp->ratonesCazados = 21;

echo $temp->mostrarNombre().': ';
echo $temp->mostrarRatones().' ratones<br>';

$temp = new Perro();
$temp->nombre = 'patan';
$temp->gatosCazados = 11;

echo $temp->mostrarNombre().': ';
echo $temp->mostrarGatos().' gatos<br>';

?>
```

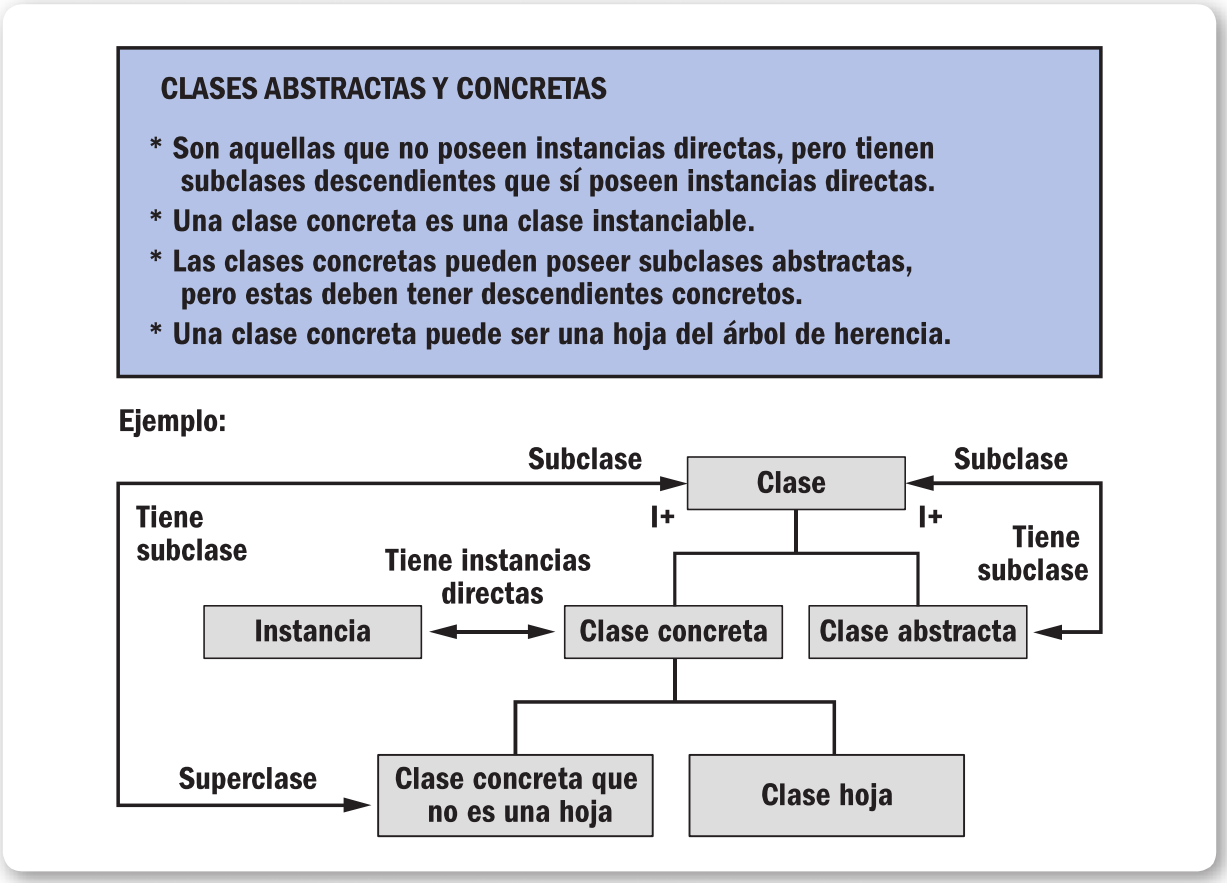


Figura 4. Observamos la diferencia entre una clase abstracta y una concreta.

Si deseamos definir métodos en las clases derivadas y no en la clase abstracta, debemos utilizar interfaces (veremos este concepto más adelante) o declarar métodos abstractos:

```
abstract class Ejemplo1 {
        public $propiedad;
        abstract function nombreMetodo();
}
```

Los métodos abstractos no se pueden llamar y se suelen utilizar para ser heredados por otras clases dentro de las cuales no tienen que ser, necesariamente, declarados como abstractos.

Las clases derivadas podrán o no redefinir su funcionalidad, dependiendo del modificador de acceso asignado a una función. Para evitar este hecho, existe el modificador **final**:

```
<?php

class Ejemplo {

        public $radio;

        final function superficie() {
                if (is_numeric($this->radio))
                        return pow(pi() * $this->radio, 2);
                else
                        echo 'Asigne   adio';   r
        }
}

class Ejemplo2 extends Ejemplo {

        //error:
        function superficie() {
                // ...
        }
}

$templ = new Ejemplo();

$templ->radio = 10;

echo $templ->superficie();

?>
```

Una clase derivada de una clase abstracta puede ser, a su vez, abstracta. En estos casos, una clase derivada puede omitir la definición de un método abstracto definido en la clase base, pero solo si esta (la clase derivada), se declara como abstracta.

Clonación de objetos

Cuando se crea una instancia de una clase y se asigna esa instancia a otra variable, se dice que se crea un alias del objeto:

```
<?php

$obj_a = new Ejemplo();
$obj_b = $obj_a;

?>
```

Los objetos en PHP 5 son referenciados a través de **manejadores**, y lo que se produce en estos casos es una duplicación del manejador. O sea, que tanto **$obj_a** como **$obj_b** hacen referencia al mismo objeto, y cualquier operación sobre **$obj_a** afectará las características de **$obj_b**, y viceversa. Además, en esta versión del lenguaje, todas las variables

CLONACIÓN

En las últimas versiones del lenguaje PHP, es posible redefinir la función reservada **__clone** para personalizar la clonación agregando los comportamientos deseados. El código dentro de la definición se ejecuta inmediatamente después de realizar la clonación.

que representan objetos son, en realidad, referencias a ellos, por lo que no es necesario utilizar el operador **&** (ampersand).

Para copiar objetos y trabajar sobre ellos de manera independiente, existe la función **__clone**, que de forma predeterminada crea un objeto copiando todos los atributos del original (bit a bit):

```
<?php

class Ejemplo {

        public $variable;

        public function mostrar() {
                echo "Valor: $this->variable<br>";
        }

}

$temp1 = new Ejemplo();

$temp1->variable = 10;

$temp2 = clone $temp1;

$temp2->variable = 20;

$temp1->mostrar();

$temp2->mostrar();

?>
```

Si definimos explícitamente la función **__clone**, podemos controlar los valores de las propiedades después de realizar la clonación (se ejecuta luego de la copia bit a bit):

```
<?php

class Ejemplo {

        public $variable;
        private $fechaCreacion;

        function __construct() {
                $this->fechaCreacion = date("Y-m-d H:i:s");
        }

        function __clone() {
                $this->fechaCreacion = date("Y-m-d H:i:s");
        }

        public function mostrar() {
                echo "Valor: $this->variable ($this->fechaCreacion)<br>";
        }

}

$temp1 = new Ejemplo();

$temp1->variable = 10;

sleep(2);

$temp2 = clone $temp1;

$temp2->variable = 20;

$temp1->mostrar();

$temp2->mostrar();

?>
```

Definir explícitamente la función **__clone** puede ser útil en caso de que una propiedad del objeto clonado tuviera asignado como valor un objeto (por caso **H**). En este caso, obtendríamos como resultado dos objetos (los objetos clonados inicialmente) diferentes, pero sus propiedades apuntarían al mismo objeto (**H**, que no se clona). Es decir, la clonación no es recursiva.

Comparación de objetos

Para comparar objetos, contamos con dos operadores que nos permitirán realizar distintas evaluaciones acerca del contenido de cada uno:

- Con el operador tradicional de comparación simple (**==**), el resultado será verdadero si ambos objetos tienen los mismos atributos y cada uno de ellos posee los mismos valores.
- Con el operador de identidad (**===**), el resultado será verdadero si las variables comparadas son referencias a la misma instancia de la misma clase.

Tipos de datos especiales

Desde las versiones más recientes del lenguaje PHP, podemos hacer que una función reciba como argumento un objeto de un tipo determinado:

```
<?php
class nombreClase1 {
  //...
}
function nombreFuncion(nombreClase1 $objeto) {
  //...
}
$obj1 = new nombreClase1();
nombreFuncion($obj1);
?>
```

Lo anterior también se aplica a métodos (recordemos que un método es una función asociada a una clase). En caso de que una función o método acepte un objeto de un determinado tipo, también recibirá instancias de las clases derivadas. Con respecto a esto, tenemos el operador **instanceof**, que permite saber si un objeto es una instancia de una determinada clase:

```
<?php

class nombreClase1 {
  //...
}

$obj1 = new nombreClase1();

if ($obj1 instanceof nombreClase1) {
  echo "Si.";
} else {
  echo "No.";
}

?>
```

Antes, vimos una referencia de la instancia, analicemos otro ejemplo donde utilizamos la clase **animal**:

COMPARACIÓN DE OBJETOS

El operador == es útil para poder comparar dos objetos clonados, por ejemplo, si sabemos que tienen clases que se pueden modificar, y que estas pudieron haber sido cambiadas en cualquier parte de la aplicación, ya sea por nosotros como programadores o por algún valor recolectado desde la pantalla de ingreso de datos, que es operada por el usuario de dicha aplicación.

```
<?php

include 'animal.class.php';

$temp = new Gato();
$temp->nombre = 'felix';

if($temp instanceof Gato)
        echo $temp->mostrarNombre().' es un gato';
elseif($temp instanceof Perro)
        echo $temp->mostrarNombre().' es un perro';
else
        echo $temp->mostrarNombre().' no es ni un gato ni un perro';

?>
```

Otro tipo de estructura que admite PHP son las **interfaces**, que se utilizan para el conjunto de métodos que deberán definir (de manera obligatoria) las clases que implementan la interfaz.

En una interfaz, no es posible definir métodos, solo declararlos con sus nombres y argumentos, y no por sus niveles de acceso: todos los métodos en una interfaz deben ser públicos. Esto sucede a diferencia de las clases abstractas, en donde existe la posibilidad de definir métodos en la misma clase.

Las interfaces son definidas utilizando la palabra reservada **interface**:

ACCESO A CONTENIDOS

En Internet, es posible encontrar y descargar una gran cantidad de código escrito bajo la metodología orientada a objetos. Si revisamos algunas de las opciones, podremos darnos una idea cabal de las ventajas que su utilización conlleva y de por qué se trata de un tema tan popular en la actualidad.

```
<?php

interface Ejemplo1 {

        public function metodo1($variable);
        public function metodo2();
}

class Ejemplo2 implements Ejemplo1 {

        function metodo1($variable) {
                echo '<br>Hola '.$variable;
        }

        function metodo2() {
                echo '<br>====================>';
        }
}

class Ejemplo3 implements Ejemplo1 {

        function metodo1($variable) {
                echo '<br>Saludos '.$variable;
        }

        function metodo2() {
                echo '<br>||||||||||||||||||||>';
        }
}

$temp1 = new Ejemplo2;
$temp1->metodo1('pepe');
$temp1->metodo2();

$temp2 = new Ejemplo3;
$temp2->metodo1('tito');
$temp2->metodo2();

?>
```

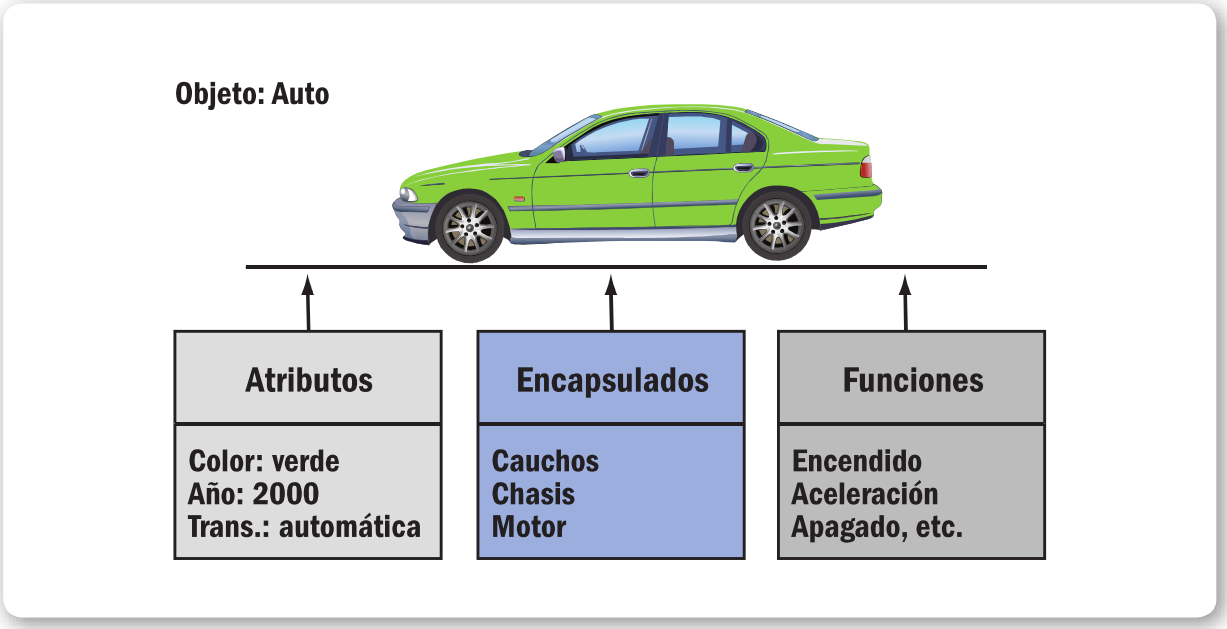


Figura 5. En esta imagen, podemos observar la creación de un objeto, con sus correspondientes características.

Las clases pueden implementar más de una interfaz, separando cada una mediante comas (,):

```
class cA implements iA, iB, iC { ... }
```

RESUMEN

Explicamos las características más destacadas de la programación orientada a objetos implementada a través de PHP: cómo crear clases e instancias de objetos a partir de ellas, los métodos principales, la comparación y la clonación de objetos, y todo lo que un objeto hereda de su clase principal por defecto.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una clase?
2. ¿Qué es una propiedad?
3. ¿Qué es un método?
4. ¿Cuál es la diferencia entre una función tradicional y un método?
5. ¿En qué situaciones sería importante definir el método **__call**?

EJERCICIOS PRÁCTICOS

1. Genere una clase **persona** con sus características, y luego clónela para poder modificar alguna de ellas.
2. Utilice un método de comparación para obtener las diferencias entre la clase **persona** y el clon que modificó.
3. Genere una clase para almacenar fechas de cumpleaños, y un método para calcular cuántos días faltan para llegar a cada cumpleaños.
4. Desarrolle una aplicación que utilice la función **__autoload**.

PROFESOR EN LÍNEA

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

*04

Referencia de funciones

En este capítulo, conoceremos algunas de las funciones disponibles en la versión base de PHP, que utilizaremos de manera frecuente en nuestro trabajo diario como desarrolladores. Entre ellas, las funciones para el manejo de sesiones, de cookies y de fecha y hora, las matemáticas y aquellas para el tratamiento de cadenas de carecteres.

↗↗↗

Extensiones disponibles

La variedad de extensiones disponibles a través de PHP es inmensa y es uno de los puntos más fuertes del lenguaje. Veamos la siguiente tabla:

FUNCIONES Y EXTENSIONES DISPONIBLES PARA PHP		
Acceso directo a E/S	Apache	BBCode
Caché	Cadenas de caracteres	Calendario
Capa de abstracción de bases de datos	Cifrado Mcrypt	Clases/Objetos
COM y .Net (para plataformas Windows)	Compresión Bzip2	Compresión Zlib
Control de procesos	Control de salida	Correo
Cracking	CURL (*Client URL Library*)	DB++
DBase	Dbx	Depurador avanzado de PHP
Directorio	DOM	DOM XML
Expresiones regulares	Fecha y hora	filePro
Firebird/InterBase	FriBiDi	FrontBase
FTP	GeoIP	Gestión de errores y registros
Gestión de funciones	Haru PDF Functions	HTTP
Hyperwave	IBM DB2	Iconv
IIS	Imágenes	Imagick (*Image Library*)
IMAP, POP3 y NNTP	Impresora	Informix

FUNCIONES Y EXTENSIONES DISPONIBLES PARA PHP (CONTINUACIÓN)		
Ingres II	Integración de Java y PHP	IRC
JSON	LDAP	Libxml
Lotus Notes	Manejo de archivos Zip	Manejo de sesiones
Matemáticas	Matemáticas de precisión arbitraria BCMath	Matrices
Memoria compartida	Mhash	Microsoft SQL Server
Mimetype	Miscelánea	mSQL
MySQL	MySQLi	ODBC
OpenSSL	Oracle	Para compilador Bytecode de PHP
PDF	PostgreSQL	PostScript
Rar	Red	Secuencias
Shockwave Flash	SimpleXML	Sistema de archivos
SNMP	SOAP	Sockets
SPL (Standard PHP Library)	SQLite	Subversion
Sybase	TCP	Tidy
Tipo de carácter	Tratamiento de etiquetas ID3	Unicode
URL	Verisign Payflow Pro	WDDX
XML	XMLReader	XML-RPC
XMLWriter	XSL	XSLT

Tabla 1. Algunas de las funciones principales para utilizar desde PHP.

Estas son solo algunas opciones; el lenguaje aumenta el número de extensiones en cada versión y perfecciona las ya existentes. En el manual de PHP, podemos encontrar más de ciento cincuenta extensiones documentadas de acuerdo con varios criterios. En muchas ocasiones, los programadores con experiencia se sorprenden al saber de cualidades de una extensión poco conocida, que les ayuda a resolver un problema específico de manera rápida y sin complicaciones. En los siguientes apartados, veremos algunas de las funciones más utilizadas a diario por los desarrolladores, aquellas que nos permitirán sacar provecho de las virtudes del lenguaje y que, por su facilidad de uso, nos dan la posibilidad de resolver diferentes cuestiones generales acerca de algún inconveniente.

Funciones para el manejo de sesiones

Las llamadas **variables de sesión** nos permiten mantener el valor de las variables a lo largo de las distintas páginas que visitamos en un mismo servidor. Hay muchos casos en los que podemos observar en funcionamiento el trabajo con sesiones. Pensemos en un sitio que requiere que el usuario ingrese ciertos datos (por ejemplo, **nombre de usuario** y **contraseña**) para validar su acceso al sistema. Si hay coincidencia (la información podría compararse con una base de datos), se inicializa (es decir, se asigna un valor en particular) una variable de sesión.

El lenguaje PHP almacena las variables de sesión en un array denominado **$_SESSION**. Su forma de uso es la siguiente:

```
<?php
$_SESSION[‘nombreVariable’] = $valor;
?>
```

Estas variables, más allá de su acceso, son idénticas a las comunes: podemos asignarles valores, eliminarlas o crear nuevas, incluso, podríamos asignarles arrays.

Desde las últimas versiones el lenguaje (PHP 4.1.0 en adelante), se usa el array **$_SESSION**, pero en las anteriores (4.0.6 e inferiores) se solía emplear el array especial **$HTTP_SESSION_VARS**. Por lo general, antes de interactuar con esta clase de variables, debemos iniciar de manera explícita una sesión, algo que logramos a partir de la función **session_start**, cuya sintaxis es la siguiente:

```
<?php

session_start();

$_SESSION[‘nombreVariable’] = ‘Este es el valor’;

echo $_SESSION[‘nombreVariable’];

?>
```

Antes de una llamada a **session_start**, no puede haber ninguna salida al navegador (ni espacios en blanco, ni código HTML, ni salidas PHP). Por ejemplo, lo siguiente produciría un error:

```
<?php

echo ‘Esto produce un error ...’;

session_start();

?>
```

En algunas configuraciones es innecesario invocar de manera explícita a **session_start**, si es que la directiva **session.auto_start** está definida en el archivo **php.ini**:

```
session.auto_start = 1
```

Para destruir los datos almacenados correspondientes a una sesión, utilizamos la función **session_destroy**, que no recibe argumentos y devuelve **verdadero** en caso de éxito y **falso** si por algún motivo no se llevó a cabo la destrucción.

```
session_destroy();
```

La función **session_unset** nos da la posibilidad de eliminar y liberar el espacio asignado a las variables de sesión inicializadas. No recibe argumentos.

```
<?php

session_start();

$_SESSION[‘nombreVariable1’]= ‘valor1’;
$_SESSION[‘nombreVariable2’]= ‘valor2’;
$_SESSION[‘nombreVariable3’]= ‘valor3’;

session_unset();
session_destroy();

?>
```

En lugar de esta función también podemos utilizar **unset**, que de manera general (funciona con cualquier clase de variables además de las de sesión) recibe como argumento un nombre de variable y la elimina:

```
unset($_SESSION);
unset($_SESSION[‘nombreVariable’]);
```

Otra opción es asignar un array vacío a **$_SESSION**, algo que en términos prácticos es válido para cualquier array y, en cuanto al trabajo con sesiones, equivale a utilizar la función **session_unset**:

```
$_SESSION = array();
```

A cada usuario que inicia una sesión, PHP le asigna un identificador único denominado **identificador de sesión** (**session id**). Para conocer nuestro identificador actual o asignarle un nuevo valor, existe la función **session_id**:

```
<?php

session_start();

echo session_id();

?>
```

El identificador de sesión generado por PHP tiene como característica que no se repite, es distinto para cada usuario que inicia una sesión. Un identificador tiene, generalmente, la siguiente forma:

```
89b8bd79599013bea68e274152c2a7f0
```

Si por algún motivo necesitamos obtener un nuevo identificador dentro de una misma sesión sin perder los valores (o sea, sin recurrir a **session_unset** y **session_destroy**), podemos utilizar la función **session_regenerate_id**:

```
<?php

session_start();
```

```
$id = session_id();

session_regenerate_id();

$nuevoId = session_id();

echo '<li>'.$id;
echo '<li>'.$nuevoId;

?>
```

Al igual que con **session_start**, no debe haber salidas al navegador antes de invocar **session_regenerate_id**.

La función **print_r** puede ser útil para visualizar todas las variables contenidas en un array específico, no solo **$_SESSION**. Veamos un ejemplo para explorar sus posibilidades:

```
<?php

session_start();

$_SESSION['nombreVariable1'] = 'valor1';
$_SESSION['nombreVariable2'] = 'valor2';
$_SESSION['nombreVariable3'] = 'valor3';

choArray($_SESSION);

function echoArray($array) {
  if (is_array($array)) {
echo '<pre>';
print_r($array);
echo '</pre>';
  }
}

?>
```

Así como cada sesión tiene asociado un identificador, también cada una posee un nombre. La función **session_name** nos permite conocerlo e incluso modificarlo (si es que le pasamos el nuevo nombre como argumento):

```
<?php

session_start();

$_SESSION[‘nombreVariable1’]= ‘valor1’;
$_SESSION[‘nombreVariable2’]= ‘valor2’;
$_SESSION[‘nombreVariable3’]= ‘valor3’;

echo session_name();

?>
```

Y si quisiéramos modificarlo:

```
<?php

session_name(‘nuevoNombreDeSesion’);

session_start();

$_SESSION[‘nombreVariable1’]= ‘valor1’;
$_SESSION[‘nombreVariable2’]= ‘valor2’;
$_SESSION[‘nombreVariable3’]= ‘valor3’;

echo session_name();

?>
```

En el archivo **php.ini** ya contamos con una directiva para recuperar el valor del nombre de la sesión (**session.name**), por lo cual, si quisiéramos modificarlo en nuestros scripts, tendríamos que invocar **session_name** en todas las páginas, incluso, antes de **session_start**.

```
session.name = PHPSESSID
```

La función **session_name** devuelve el nombre actual de la sesión, por lo tanto, algo como lo siguiente es válido:

```
<?php

$nombreAnterior = session_name('nombreNuevo');

?>
```

La función **session_cache_expire** nos permite conocer el tiempo de duración, en segundos, de una sesión:

```
<?php

session_start();
echo session_cache_expire();

?>
```

Dentro del archivo **php.ini**, contamos con una directiva llamada **session.cache_expire** para definir este valor en segundos:

```
session.cache_expire = 0
```

También nos encontramos con la directiva **session.cookie_lifetime** para definir el tiempo de vida (en segundos) de las sesiones. Si se le asigna **0**, la sesión caducará recién cuando el navegador se cierre y se vuelva a abrir:

```
session.cookie_lifetime = 0
```

Otra directiva relativa es **session.gc_maxlifetime**, cuya finalidad es indicar a PHP que, pasados **N** segundos, los datos de sesión serán limpiados (eliminados) del sistema:

```
session.gc_maxlifetime = 1440
```

A continuación, veremos un ejemplo para poner en práctica algunas de las funciones anteriores. Se trata de un pequeño script para validar el ingreso de un usuario a una determinada sección de un sitio (una sección restringida, no pública). En primer lugar, iniciamos la sesión a través de la función **session_start**. Debemos recordar que, antes de esta línea, no puede haber ninguna salida al navegador:

```
session_start();
```

El formulario para el ingreso de datos es como el siguiente (debemos copiar el código para dibujar los controles en la página):

```
<br />
<form method="post" action="?">
<table border="1" cellspacing="0" cellpadding="5" align="center">
<tr>
<td>Ingrese su nombre de usuario</td>
<td><input type="text" name="frmUsername"></td>
</tr>
<tr>
<td>Ingrese su contraseña</td>
<td><input type="password" name="frmPassword"></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Enviar datos"></td>
</tr>
</table>
</form>
```

Una vez enviada la información, comparamos los datos con el contenido de las variables **username** y **password**:

```
$username = "user";
$password = "pass";

if (count($_POST)) {
  if ($username = $_POST["frmUsername"] && $pass-
word = $_POST["frmPassword"]) {
    $_SESSION["usuarioRegistrado"] = true;
    header("location: ?");
} else {
    $error = "Datos incorrectos";
  }
}
```

Si no coinciden, mostramos un mensaje de error. En caso de ser correctos, asignamos un valor a la variable de sesión correspondiente:

```
$_SESSION[“usuarioRegistrado”] = true;
```

Luego, redirigimos a la misma página (a través de la función **header**) para que los datos del array **POST** (el método utilizado en el formulario) no se reenvíen en caso de que el usuario intente refrescarla. Si la variable de sesión así lo indica, mostramos un mensaje y un enlace para finalizar la sesión actual. En caso contrario, mostramos el formulario inicial:

```
if ($_SESSION[“usuarioRegistrado”]) {
  echo “<center>Ingreso correcto - <a
href=”?terminar=1”>Terminar sesion</a></center>”;
} else {
  echo “<center>$error</center>”;
  //mostramos aquí el formulario de ingreso
}
```

El código para terminar la sesión es el siguiente:

```
if ($_GET[“terminar”]) {
  session_unset();
session_destroy();
header(“location: ?”);
}
```

Ya desarrollamos nuestra primera aplicación con la utilización de sesiones. Esta técnica es de uso común en sistemas de cualquier tipo y alcance; será beneficioso saber cómo utilizar las funciones disponibles en la extensión PHP.

Funciones para el manejo de cookies

Las **cookies** son pequeños archivos de texto que una aplicación web puede generar y enviar a la computadora de un usuario. Esta las recibe solo si su configuración local así lo dispone, y esto, generalmente, se define a través de las opciones del navegador web.

El objetivo inicial de esta técnica es reconocer cuando un mismo usuario vuelve a un mismo sitio, y así poder personalizar su interacción con el sistema. En muchos sitios, al ingresar a una determinada sección, podemos observar mensajes como **Recordarme en este equipo**, por ejemplo.

PHP provee una función para generar y enviar cookies. Su nombre es **setcookie**, y recibe como argumentos un nombre, un valor, una fecha de expiración en formato **timestamp** (explicaremos más acerca de esto en la sección dedicada a las funciones de tratamiento de fecha y hora), la ruta en la que tendrá validez (podríamos hacer que la cookie solo se tenga en cuenta en el directorio **/usuarios**, por ejemplo), el dominio de nuestro sitio en formato **nombre-sitio.com**, y si se utilizara solamente con HTTP seguro: **https://www.nombre-sitio.com**. El único argumento requerido es el primero, el nombre de la cookie. Esta función debe invocarse antes de cualquier salida HTML y no podemos dejar espacios ni líneas en blanco. Veamos un ejemplo:

```
<?php

$nombre = 'Francisco';
setcookie('usuario', $nombre, time()+60*60*24*30);

?>
```

Con este código, almacenamos (siempre y cuando el cliente lo permita) una cookie que contendrá cierta información acerca del usuario, por ejemplo, su nombre. El archivo generado (ubicado comúnmente en la carpeta de archivos temporales de Internet) se ve de la siguiente manera:

```
usuario
Francisco
localhost/ejemplos/
1024
4217296640
29950535
50740704
29944501
*
```

En los siguientes accesos del usuario al sitio (hasta treinta días posteriores a la creación de la cookie), podríamos reconocerlo y utilizar la información almacenada, todo a través del array especial **$_COOKIE**:

```
<?php

echo 'Bienvenido nuevamente '.$_COOKIE['usuario'];

?>
```

Este es solo un caso entre tantos de los que podemos lograr a través de la utilización de cookies en nuestras aplicaciones web. Es muy útil para mejorar la experiencia de uso de los clientes de nuestras páginas.

Funciones para el manejo de fecha y hora

Las funciones relativas al tratamiento de fechas y horarios no son demasiadas, pero solo algunas de ellas nos bastarán para resolver prácticamente cualquier requerimiento que surja en nuestras aplicaciones.

Un aspecto que debemos tener en cuenta es el de diferenciar la **fecha/hora** en el cliente y en el servidor: un servidor puede estar ubicado en un

punto geográfico tal que su huso horario difiera del cliente que se conecta para realizar peticiones. En lo sucesivo, a menos que se especifique lo contrario, todas las fechas y horarios tomarán como base el lado servidor.

Antes de pasar a explicar las posibilidades de cada función, debemos tener presente un concepto relativo a una forma de almacenar fechas: **la marca de tiempo Unix**. Este formato es utilizado por múltiples funciones, ya sea como argumento o como resultado. Una marca de tiempo Unix (o **timestamp**) es el número de segundos transcurridos entre el 1 de enero de 1970 y la hora especificada. Para graficar esta situación, comencemos con la función **mktime**, que devuelve la marca de tiempo actual:

```
<?php

echo mktime();

?>
```

El resultado dependerá del momento en que ejecutemos el script, pero podría ser similar a esto:

```
1215823703
```

Esta función recibe una serie de argumentos (todos opcionales) y entre los más importantes podemos citar los siguientes:

- hora
- minuto
- segundo
- mes
- día
- año

Con respecto al año, tenemos la posibilidad de indicarlo utilizando dos o cuatro dígitos: los valores ubicados entre 0 y 69 se convierten a 2000-2069 y los que están entre 70 y 100, a 1970-2000. En caso de utilizar cuatro dígitos, debemos tener presente que, en la actualidad, hay sistemas que limitan el rango a valores entre 1901 y 2038.

```
<?php

echo mktime(10, 23, 59, 7, 11, 2013);

?>
```

Lo anterior equivale al 11 de julio de 2013, a las 10 horas 23 minutos y 59 segundos, que en el formato timestamp es igual a 1215782639. En caso de pasar algún argumento no válido, es decir, fuera de rango, la función **mktime** en ocasiones lo interpreta de una manera distinta de como lo haría una persona. Tomemos algunos ejemplos para observar esta clase de situaciones.

- El 31 de diciembre de 2013:

```
<?php echo mktime(0, 0, 0, 12, 31, 2013); ?>
```

- Si avanzamos un día (31 + 1), ya pasamos al 1 de enero de 2014:

```
<?php echo mktime(0, 0, 0, 12, 32, 2013); ?>
```

- El primer día de abril de 2014:

```
<?php echo mktime(0, 0, 0, 4, 1, 2014); ?>
```

- El último día de marzo de 2014 (1-1, el día anterior al 1/4):

```
<?php echo mktime(0, 0, 0, 4, 0, 2014); ?>
```

- El penúltimo día de marzo de 2014 (1-2):

```
<?php echo mktime(0, 0, 0, 4, -1, 2014); ?>
```

- Falta 1 segundo para el 1 de enero de 2015:

```
<?php echo mktime(24, 0, -1, 12, 31, 2014); ?>
```

- Falta 1 segundo para el 1 de enero de 2015:

```
<?php echo mktime(23, 59, 59, 12, 31, 2014); ?>
```

- Es el 1 de enero de 2015:

```
<?php echo mktime(24, 0, 0, 12, 31, 2014); ?>
```

- Es el 1 de enero de 2015:

```
<?php echo mktime(0, 0, 0, 1, 1, 2015); ?>
```

El potencial de esta función es muy grande, y su versatilidad nos permite realizar cálculos de fechas de manera rápida y sin complicaciones, algo que nos llevaría mucho tiempo si tuviéramos que hacerlo por nuestra cuenta.

Una función similar es **time**, que no recibe argumentos y devuelve la marca de tiempo Unix actual (o sea, es como invocar **mktime** sin argumentos):

```
echo time();
```

Por su parte, la función **microtime** devuelve la marca de tiempo Unix actual, medida en microsegundos:

```
echo microtime();
```

Es evidente que un **timestamp** no es un formato demasiado comprensible para un usuario común que quiere, simplemente, visualizar una fecha. Para estos casos, contamos con dos funciones que nos permiten dar formato a una fecha/hora preestablecida: **date** y **strftime**. La primera recibe como argumento principal el formato (más adelante explicaremos cómo construir uno) y opcionalmente, un **timestamp**. En caso de omitir este último valor, se toman por defecto la fecha y la hora actuales. La sintaxis de esta función es la siguiente:

```
date($formato, $fecha);
```

El formato es una cadena (encerrada entre comillas dobles), dentro de la cual algunos caracteres tienen un significado especial, por ejemplo:

MODO SEGURO

Una de las reglas de seguridad de PHP consiste en comprobar que los directorios a los que accedemos hayan sido creados por el usuario propietario del script en ejecución.

- **d**: día del mes en dos dígitos (completa con ceros a la izquierda).
- **D**: día de la semana en tres letras en inglés (Mon, Thu, Sat, etcétera).
- **j**: día del mes en dos dígitos (no completa con ceros a la izquierda).
- **l**: día de la semana en inglés (Monday, Thursday, Saturday, etcétera).
- **N**: representación numérica del día de la semana, desde 1 (para lunes) hasta 7 (para domingo).
- **S**: sufijo para formar el ordinal del día del mes en inglés (st, nd, rd o th).
- **w**: representación numérica del día de la semana desde 0 (para domingo) hasta 6 (para sábado).
- **z**: el día del año (desde 0 hasta 365).
- **W**: número de la semana del año (las semanas comienzan en los días lunes).
- **F**: representación textual del mes, en inglés (January, February, etcétera).
- **m**: mes del año en dos dígitos (completa con ceros a la izquierda).
- **M**: mes del año en tres letras, en inglés (Jan, Feb, Mar, etcétera).
- **n**: mes del año en dos dígitos (no completa con ceros a la izquierda).
- **t**: cantidad de días del mes (desde 28 hasta 31).
- **L**: devuelve 1 si el año es bisiesto y 0 si no lo es.
- **o**: devuelve el año en cuatro dígitos. Si la semana de la fecha termina en el año próximo o en el anterior, lo toma como valor.
- **Y**: devuelve el año en cuatro dígitos.
- **y**: devuelve el año en dos dígitos.
- **a**: devuelve am (antemeridiano) o pm (posmeridiano).
- **A**: devuelve AM (antemeridiano) o PM (posmeridiano).
- **B**: hora Swatch Internet (va desde 000 a 999).
- **g**: hora en formato 12-horas (no completa con ceros a la izquierda).
- **G**: hora en formato 24-horas (no completa con ceros a la izquierda).
- **h**: hora en formato 12-horas (completa con ceros a la izquierda).
- **H**: hora en formato 24-horas (completa con ceros a la izquierda).
- **i**: minuto (completa con ceros a la izquierda).
- **s**: segundos (completa con ceros a la izquierda).
- **u**: milisegundos.
- **e**: zona horaria.
- **O**: diferencia con la hora Greenwich (por ejemplo, +0400).

- **P**: diferencia con la hora Greenwich (por ejemplo, +04:00).
- **T**: abreviación de la zona horaria.
- **c**: formato predefinido: Y-m-dTH:i:sP (la T forma parte de la cadena).
- **r**: formato predefinido: D,d M Y H:i:sP.
- **U**: timestamp.

Veamos algunos ejemplos:

- hora:minutos:segundos de la fecha actual:

```
<?php echo date("H:i:s"); ?>
```

- hora:minutos:segundos de una fecha específica:

```
<?php echo date("H:i:s", mktime(23, 59, 59, 12, 31, 2014)); ?>
```

- día-mes-año hora:minutos:segundos de la fecha actual:

```
<?php echo date("d-M-Y H:i:s"); ?>
```

Hay dos cuestiones recurrentes que debemos tener en cuenta al utilizar funciones como **date**: qué pasa si incluimos dentro del formato un carácter que no tiene ningún significado específico, y cómo podemos incluir caracteres sin que sean tomados y reemplazados por la función. La respuesta a la primera pregunta es que, si un carácter no tiene un significado específico, simplemente se imprime tal cual, por ejemplo:

```
echo date("Cp H:i:s");
```

El resultado sería algo semejante a **Cp 22:21:38**. En el segundo caso, para hacer que la función ignore un carácter especial, podemos anteponerle una barra de esta manera:

```
echo date("\L\a \h\o\\r\a \a\c\\t\u\a\l \e\s: H:i");
```

Notemos que tanto antes de **r** como de **t** se incluyeron dos barras; esto es porque tanto **\r** (retorno de línea) como **\t** (tabulador) o **\n** (retorno de línea), por ejemplo, tienen por sí mismos un significado especial dentro del lenguaje.

```
La hora actual es: 22:26
```

Por su parte, la función **strftime** también nos permite aplicar un formato a una fecha dada y recibe los mismos argumentos que **date**. Una de las diferencias está en la composición de su formato. Las opciones son las siguientes:

- **%a**: nombre abreviado del día de la semana.
- **%A**: nombre completo del día de la semana.
- **%b**: nombre abreviado del día del mes.
- **%B**: nombre completo del día del mes.
- **%c**: fecha y hora.
- **%C**: centuria.
- **%d**: día del mes en dos dígitos (completa con ceros a la izquierda).
- **%D**: **%m/%d/%y**
- **%e**: día del mes en dos dígitos (completa con espacios a la izquierda).
- **%g**: similar a **%G**, pero sin tener en cuenta la centuria.
- **%G**: devuelve el año en cuatro dígitos. Si la semana de la fecha termina en el próximo año o en el anterior, lo toma como valor.
- **%h**: alias de **%b**.
- **%H**: hora en formato 24-horas (completa con ceros a la izquierda).
- **%I**: hora en formato 12-horas (completa con ceros a la izquierda).

- **%j**: día del año.
- **%m**: mes del año en dos dígitos (completa con ceros a la izquierda).
- **%M**: minuto.
- **%p**: devuelve **am** (antemeridiano) o **pm** (posmeridiano).
- **%r**: devuelve la hora **am** o **pm** según el caso.
- **%R**: hora en formato 24-horas (completa con ceros a la izquierda).
- **%S**: segundo.
- **%T**: hora en formato **%H:%M:%S**.
- **%u**: representación numérica del día de la semana yendo de 1 (para lunes) a 7 (para domingo).
- **%U**: semana del año (desde el primer domingo).
- **%V**: semana del año (toma como primera una que tenga al menos cuatro días).
- **%W**: semana del año (desde el primer lunes).
- **%w**: representación numérica del día de la semana yendo de 0 (para domingo) a 7 (para sábado).
- **%y**: año en dos dígitos.
- **%Y**: año en cuatro dígitos.
- **%Z**: zona horaria.
- **%z**: equivalente a **%Z**.

Esta función tiene varias utilidades que la caracterizan, y una de las más importantes es su capacidad para tomar valores locales como referencia, por ejemplo, nombres de meses o días de la semana. Para observar este aspecto, veamos la función **setlocale**, que recibe como argumentos una categoría (ver el manual de la función para más información al respecto) y una localización:

```
setlocale(LC_ALL,'esp_ESP');
```

Junto con **strftime**, puede utilizarse de la siguiente manera:

```
<?php

setlocale(LC_ALL, 'esp_ESP');
echo strftime("%A %d de %B %Y");

?>
```

La salida de lo anterior podría ser similar a:

```
lunes 10 de febrero 2014
```

La función **strtotime** recibe como argumento una hora (admite cualquier formato en inglés) y devuelve un **timestamp**. Veamos un ejemplo:

```
<?php

echo date("H:i", strtotime("23:30"));

?>
```

En cuanto a las zonas horarias, contamos con dos funciones específicas: **date_default_timezone_get** que nos permite obtener el valor actual y **date_default_timezone_set** para definirlo. Veamos un ejemplo de cada una:

```
<?php

echo date_default_timezone_get();

date_default_timezone_set('Europe/London');

echo date_default_timezone_get();

?>
```

Una función importante, llamada **checkdate**, recibe como argumentos tres enteros (correspondientes al mes, día y año, en ese orden, en formato gregoriano) y devuelve **verdadero** si la fecha es válida, y **falso**, en caso contrario.

```
<?php

if (checkdate(2, 29, 2014)) {
echo 'Fecha valida';
} else {
  echo 'Fecha no valida';
}
?>
```

Los puntos que tiene en cuenta son:

- El año deberá estar entre 1 y 32767.
- El mes deberá estar entre 1 y 12.
- Los valores del día tienen relación con los datos anteriores.

Esta función toma en consideración los años bisiestos.

Tratamiento de cadenas de caracteres

Al trabajar con el contenido textual que luego será exhibido en una página web, lenguajes como PHP están obligados a proveer funciones para el tratamiento de cadenas de caracteres o strings. Veremos cuáles son y cómo incluirlas en los scripts. Observemos las funciones que cumplen una tarea fundamental, como enviar una cadena a la salida estándar, en este caso, un navegador web. La sentencia **echo** recibe como parámetro una cadena de caracteres:

```
<?php

echo "Esto es un ejemplo";

?>
```

Podemos realizar saltos de línea de manera explícita si incluimos un **\n** o simplemente mientras generamos el valor. Tomemos los siguientes ejemplos para entender en forma concreta a qué nos referimos:

```
<?php

echo "Esto \nes \nun \nejemplo";

?>
```

En cuanto a la opción alternativa:

```
<?php

echo "Esto
es
un
ejemplo";

?>
```

Ahora bien, un navegador web por defecto no asume los saltos de línea textuales como tales, sino que solo comprende los tags **br** (**
). PHP nos provee una función para anteponer una etiqueta de este tipo a cada salto de línea encontrado. Se trata de **nl2br. De esta manera, los siguientes códigos:

```
<?php

echo nl2br("Esto \nes \nun \nejemplo");

?>

<?php
```

```
echo nl2br("Esto
es
un
ejemplo");

?>
```

generan la siguiente salida:

```
Esto <br />
es <br />
un <br />
ejemplo
```

Cabe aclarar que los caracteres especiales (como por ejemplo, los **\n**) tienen sentido como tales, siempre y cuando los incluyamos entre comillas dobles. Esto vale para cualquier situación, no solo para **echo**. También podemos incluir el contenido de variables dentro de una cadena, como vemos:

```
<?php

$nombre = 'Francisco';
$edad = 28;

echo "Hola, mi nombre es $nombre y tengo $edad años.";

?>
```

La sentencia **echo** puede recibir más de un parámetro. Estos parámetros deben estar separados por comas y, como salida, los concatenará y se verán así:

```
<?php

$nombre = 'Francisco';
$edad = 28;

echo "Hola, ",'mi nombre es ',$nombre,' y tengo $edad años.';

?>
```

En el archivo **php.ini**, contamos con una directiva llamada **short_open_tag**, que nos permite utilizar una sintaxis especial para ejecutar la funcionalidad de **echo**. En caso de estar activada:

```
short_open_tag = On
```

podremos incluir líneas como las siguientes:

```
<?php

$nombre = 'Francisco';
$edad = 28;

?>

Hola, mi nombre es <?=$nombre ?> y tengo <?=$edad ?> años.
```

Con respecto a **echo**, contamos con la sentencia **print**, que es muy similar:

```
<?php

print "Esto es un ejemplo";

?>
```

Tanto **printf** como **sprintf** nos permiten dar formato a una cadena de caracteres. La diferencia entre ellas es que la primera devuelve la longitud de la cadena, mientras que la segunda devuelve la cadena generada. La idea en ambos casos es definir, dentro de un formato, el tipo de dato que se va a utilizar para mostrar los argumentos.
Observemos el siguiente ejemplo:

```
<?php

$mes = 'Julio';
$dia = 19;
$anio = 2014;

printf('Hoy es %d de %s de %d', $dia, $mes, $anio);

?>
```

Dentro del formato establecido (primer argumento de **printf**), hay determinados símbolos especiales (**%d** y **%s**, en este caso), que luego serán reemplazados por los valores de los argumentos sucesivos: día, mes y año. Debemos respetar el orden de inclusión de estos. Los símbolos especiales, además de indicarle a **printf** que debe reemplazarlos por valores, indican un tipo de dato específico. Por ejemplo, con **%d** el valor se trata como un número decimal entero, mientras que con **%s** se lo toma como una cadena. Hay más opciones:

- **b**: número binario.
- **c**: valor ASCII correspondiente.
- **u**: número decimal sin signo.
- **f**: punto flotante (con posiciones decimales).
- **o**: número octal.
- **x**: número hexadecimal con letras minúsculas.
  X: número hexadecimal con letras mayúsculas.

En el ejemplo que presentamos a continuación, mostramos un punto flotante tomando dos posiciones decimales:

```
<?php

$monto = 316.157;
printf('El precio es de %.2f', $monto);

?>
```

Su salida es lo que se imprime a continuación:

```
El precio es de 316.16
```

Recordemos que **sprintf** devuelve la cadena y, por defecto, no la imprime:

```
<?php

$mes = 'Julio';
$dia = 19;
$anio = 2014;

print sprintf('Hoy es %d de %s de %d', $dia, $mes, $anio);

?>
```

La función **fprintf** es similar a **print**, solo que, en lugar de recibir como argumento una cadena, lo que hace es tomar un descriptor de archivo y leer su contenido. Para más información acerca de descriptores podemos recurrir, en este capítulo, a la sección dedicada a funciones para el sistema de archivos.

Vimos cómo **printf** nos devolvía la longitud de la cadena pasada como argumento. En caso de querer conocer este valor sin utilizar esta función, podemos acudir a **strlen**:

```
<?php

$cadena = ' the Long blondes ';
echo strlen($cadena);

?>
```

Se toman en cuenta los espacios en blanco, por lo cual lo anterior devuelve **20**. Por su parte, la función **str_word_count** nos brinda información acerca de las palabras utilizadas en la cadena. Recibe tres argumentos, los últimos dos son opcionales: la cadena por analizar, un modo y un listado de caracteres que serán tomados como palabras separadas. El modo puede ser uno de los siguientes:

- **0**: devuelve el número de palabras encontradas.
- **1**: devuelve un array con todas las palabras encontradas.
- **2**: devuelve un array con todas las palabras encontradas y la posición de cada una como índice.

```
<?php

$cadena = 'the Long blondes';
print_r(str_word_count($cadena, 2));

/*

Array
(
    [0] => the
    [4] => Long
    [9] => blondes
)

*/

?>
```

Una función relativa, aunque diferente, es **count_chars**, que nos da un detalle acerca de los caracteres incluidos en una cadena. Recibe como argumentos la cadena por analizar y un modo, que puede ser uno de los siguientes:

- **0**: un array cuyas claves serán el número de byte (de 0 a 255) que corresponde al carácter y a la frecuencia de este como valor. Es el modo por defecto.
- **1**: similar al modo **0**, excepto que lista los caracteres utilizados en la cadena.
- **2**: similar al modo **0**, pero lista los caracteres no utilizados en la cadena.
- **3**: devuelve una cadena que contiene todos los valores utilizados.
- **4**: devuelve una cadena que contiene todos los valores no utilizados.

```
<?php
$cadena = ' the Long blondes ';
print_r(count_chars($cadena, 1));
/*
Array
(
   [32] => 6
   [76] => 1
   [98] => 1
   [100] => 1
   [101] => 2
   [103] => 1
   [104] => 1
   [108] => 1
   [110] => 2
   [111] => 2
   [115] => 1
   [116] => 1
)
*/
echo count_chars($cadena, 3);
//devuelve " Lbdeghlnost"
?>
```

Vimos en la función anterior cómo, en algunos casos, los caracteres se representan como bytes, es decir, toman como base el código ASCII. En este sentido, contamos con dos funciones: **chr**, que recibe un código ASCII y devuelve el carácter correspondiente y **ord**, que recibe un carácter y devuelve su código ASCII correspondiente:

```
<?php

$cadena = '  the Long blondes  ';

for ($c=0;$c<strlen($cadena);$c++) {
  echo '<br />El caracter "'.$cadena[$c].'" corresponde a '.ord($cadena[$c]);
}

?>
```

La salida de lo anterior tendría semejanza con lo que sigue:

```
El caracter " " corresponde al codigo 32
El caracter " " corresponde al codigo 32
El caracter "t" corresponde al codigo 116
El caracter "h" corresponde al codigo 104
El caracter "e" corresponde al codigo 101
El caracter " " corresponde al codigo 32
El caracter "L" corresponde al codigo 76
El caracter "o" corresponde al codigo 111
El caracter "n" corresponde al codigo 110
El caracter "g" corresponde al codigo 103
El caracter " " corresponde al codigo 32
El caracter "b" corresponde al codigo 98
El caracter "l" corresponde al codigo 108
El caracter "o" corresponde al codigo 111
El caracter "n" corresponde al codigo 110
El caracter "d" corresponde al codigo 100
El caracter "e" corresponde al codigo 101
El caracter "s" corresponde al codigo 115
El caracter " " corresponde al codigo 32
El caracter " " corresponde al codigo 32
```

La función **substr_count** devuelve el número de ocurrencias de una cadena (segundo argumento) dentro de otra (primer argumento):

```
<?php

$cadena = 'Someone To Drive You Home';
$busqueda = 'om';

echo substr_count($cadena, $busqueda); // 2

?>
```

La función **substr** nos permite recuperar un fragmento de una cadena. Recibe como argumentos la cadena original, la posición de comienzo y la longitud (la cantidad de posiciones por devolver es opcional):

```
<?php

$cadena = 'Someone To Drive You Home';
echo substr($cadena, 8, 12); //To Drive You

?>
```

Substr_replace posibilita reemplazar parte de una cadena por otra. Recibe como argumentos la cadena original, la cadena sustituta, la posición de comienzo y la longitud (la cantidad de posiciones por tomar es opcional):

```
<?php

$cadena = 'Someone To Drive You Home';
$me = 'Me';

echo substr_replace($cadena, $me, 17, 3);
//Someone To Drive Me Home

?>
```

Si quisiéramos comparar dos cadenas podríamos hacer lo siguiente:

```
<?php

$cadena = 'Someone To Drive You Home';

if (substr($cadena, 0, 3) == 'Som') {
  echo 'Son iguales';
} else {
  echo 'No son iguales';
}

?>
```

La función **substr_compare** desempeña una tarea similar y recibe como argumentos la cadena principal, la cadena por comparar, la posición de inicio, el desplazamiento y, opcionalmente, la insensibilidad a mayúsculas. Devuelve **0** si son iguales:

```
<?php

$cadena = 'Someone To Drive You Home';
$buscar = 'som';

if (substr_compare($cadena, $buscar, 0, 3, true) == 0) {
  echo 'Son iguales';
} else {
  echo 'No son iguales';
}

?>
```

Estas funciones trabajan con posiciones. Si quisiéramos encontrar la posición de una determinada cadena dentro de otra, podríamos utilizar la función **strpos**:

```
<?php

$cadena = 'Someone To Drive You Home';
$buscar = 'To';

$posicion = strpos($cadena, $buscar);

if ($posicion === false) {
  echo 'No se encontro.';
} else {
  echo 'Se encontro en la posicion '.$posicion;
}

?>
```

La función **stripos** es semejante, pero no diferencia entre mayúsculas y minúsculas. Estas funciones devuelven la aparición de la primera ocurrencia, y, si quisiéramos encontrar la última aparición, podríamos utilizar las funciones **strrpos** y **strripos**, respectivamente. Para reemplazar todas las apariciones de una cadena dentro de otra, existe la función **str_replace** (su equivalente **str_ireplace** no diferencia entre mayúsculas y minúsculas):

```
<?php

$cadena = 'Su sueldo es de $800.-';
$buscar = '$800';
$reemplazar = 'u$s 8000';

echo str_replace($buscar, $reemplazar, $cadena);

?>
```

La función **strtr** es similar, solo que con ella podemos reemplazar caracteres específicos por otros:

```
<?php
echo strtr("Oslo", "oO", "aI"); //Isla
?>
```

Las funciones **strtolower** y **strtoupper** nos permiten, respectivamente, pasar una cadena a minúsculas o a mayúsculas. Por su parte, **ucfirst** y **ucwords** toman la cadena y convierten solamente el primer carácter de la primera palabra a mayúsculas o el primer carácter de todas las palabras.

```
<?php
$cadena = 'the Long blondes';
echo strtolower($cadena);
echo '<br />';
echo strtoupper($cadena);
echo '<br />';
echo ucfirst($cadena);
echo '<br />';
echo ucwords($cadena);
?>
```

La salida de lo anterior sería:

```
the long blondes
THE LONG BLONDES
The Long blondes
The Long Blondes
```

La función **trim** elimina determinados caracteres ubicados al principio y al final de la cadena (primer argumento). Entre estos caracteres se encuentran:

- Espacios en blanco.
- Tabuladores (**\t**).
- Tabuladores verticales (**\x0B**).
- Saltos de línea (**\n**).
- Un retorno de carro (**\r**).
- Byte **NULL** (**\0**)

```
<?php

$cadena = ' the Long blondes ';

echo trim($cadena);

?>
```

Como segundo argumento (opcional), se pueden enumerar todos los caracteres por eliminar. La funciones **ltrim** y **rtrim** son similares a **trim**, pero se aplican únicamente al lado izquierdo o derecho, respectivamente.

Para quitar todas las etiquetas de una cadena existe la función **strip_tags**, que recibe como argumento un string y, opcionalmente, los tags permitidos, es decir, aquellos que no se eliminarán:

```
<?php

echo strip_tags('the <b>Long</b> blondes');

?>
```

Esta función elimina todo tipo de etiquetas, no únicamente las de **XHTML**.

La función **str_split** convierte una cadena a una matriz. Como argumentos, recibe la cadena original y, opcionalmente, el tamaño en caracteres de cada posición del array devuelto:

```
<?php
print_r(str_split('Ejemplo de str_split', 4));
/*
Array
(
    [0] => Ejem
    [1] => plo
    [2] => de s
[3] => tr_s
    [4] => plit
)
*/
?>
```

Por su lado, **chunk_split** divide una cadena en partes y recibe como segundo argumento el ancho máximo para cada una:

```
<?php
echo chunk_split('Ejemplo de chunk_split', 6);
/*
Ejempl
o de c
hunk_s
plit
*/
?>
```

La función **wordwrap** es similar, solo que permite, además, incluir la cadena de separación (tercer argumento) y si se van a cortar o no las palabras que sobrepasen la longitud (cuarto argumento):

```
<?php

echo wordwrap('Ejemplo de wordwrap', 5, '\n', 1);

/*

Ejemp
lo de
wordw
rap

*/

?>
```

En caso de que, por algún motivo, necesitáramos reordenar los caracteres de una cadena, podríamos utilizar la función **str_shuffle**:

```
<?php

echo str_shuffle("mar"); //arm, ram, etc.

?>
```

Otra función muy particular es **str_repeat**, que repite un número N de veces una cadena dada:

```
<?php

echo str_repeat(" 2008 ", 10000);

?>
```

La función **str_pad** nos permite completar una cadena hasta llegar a un determinado número de caracteres. Recibe como argumentos la cadena original, la longitud hasta la que queremos llegar, una cadena de relleno para completar y un tipo de relleno (**STR_PAD_RIGHT** para completar por la derecha, **STR_PAD_LEFT** para completar por la izquierda o **STR_PAD_BOTH** para completar alternativamente por ambos lados):

```
<?php
echo str_pad("32", 5, "0", STR_PAD_LEFT); // 00032
?>
```

A través de **strrev** obtenemos una cadena inversa a la original:

```
<?php
echo strrev("anita lava la tina");
?>
```

En ocasiones tales como la generación de instrucciones que luego serán enviadas a una base de datos, las cadenas deben **escaparse** previamente. Esto significa que a algunos caracteres, por ejemplo, las comillas simples, las comillas dobles y las barras, entre otros, deben anteponérseles una barra (\). Podemos escapar cadenas de caracteres si utilizamos la función **addslashes**:

```
<?php
echo addslashes("he's my doctor");
?>
```

La operación inversa, quitar las barras de una cadena escapada previamente, se puede lograr por medio de la función **stripslashes**:

```
<?php
$cad = addslashes("he's my doctor");
echo stripslashes($cad);
?>
```

La función **htmlentities** convierte caracteres especiales a sus entidades **HTML** correspondientes. Recibe como argumentos la cadena por convertir, el tratamiento que se dará a las comillas simples y dobles (opcional, **ENT_COMPAT** para convertir las dobles y preservar las simples, **ENT_QUOTES** para convertir ambas y **ENT_NOQUOTES** para preservarlas, por defecto se utiliza **ENT_COMPAT**) y el juego de caracteres (opcional, por ejemplo **ISO-8859-1**, **ISO-8859-15**, **UTF-8**, **cp866**, **cp1251**, **cp1252**, **KOI8-R**, **BIG5**, **GB2312**, **BIG5-HKSCS**, **Shift_JIS** o **EUC-JP**, por defecto, se utiliza **ISO-8859-1**):

```
<?php
echo htmlentities("<a href=''>Clic Aqui</a>", ENT_QUOTES);
//&lt;a href=&#039;&#039;&gt;Clic Aqui&lt;/a&gt;
?>
```

Podemos obtener la traducción completa a partir de la función **get_html_translation_table** (al utilizar la constante **HTML_ENTITIES**), que devuelve un array con los caracteres origen y destino:

```
<?php
print_r(get_html_translation_table(HTML_ENTITIES));
?>
```

La función **html_entity_decode** produce el efecto inverso, es decir, convierte desde entidades hacia los caracteres correspondientes. Recibe los mismos argumentos que **htmlentities**.

La función **htmlspecialchars** es idéntica a **htmlentities** y recibe los mismos argumentos; la diferencia es que traduce solamente ciertos caracteres: el ampersand (**&** por **&**), las comillas dobles (**"** por **"**), las comillas simples (**'** por **'**) y los símbolos menor que (**<** por **<**) y mayor que (**>** por **>**).

Aquí también, la traducción completa puede obtenerse a partir de la función **get_html_translation_table** (al utilizar la constante **HTML_SPECIALCHARS**), que devuelve un array con los caracteres origen y destino:

```
<?php

print_r(get_html_translation_table(HTML_SPECIALCHARS));

?>
```

Así como existe la función **html_entity_decode**, también disponemos de **htmlspecialchars_decode**, que convierte desde entidades hacia los caracteres correspondientes.

Existen ciertos algoritmos que se utilizan para encriptar cadenas de caracteres, tal es el caso de **md5** y **sha1**. PHP provee funciones para realizar este trabajo sobre cadenas a través de las funciones **md5** y **sha1**:

```
<?php

$cadena = 'Magic Markers';

$md5 = md5($cadena);
$sha1 = sha1($cadena);

?>
```

En principio, estos algoritmos actúan en un solo sentido: a partir de la cadena original se puede obtener su valor encriptado, pero no a la inversa. La función **crypt** realiza una tarea similar.

Funciones matemáticas

Dentro de la extensión provista por PHP para aplicar operaciones de carácter matemático sobre valores de variables, podemos distinguir dos grandes grupos: las funciones de conversión entre distintas bases numéricas (binario, decimal, octal, hexadecimal, etcétera) y aquellas orientadas a la geometría (seno, coseno, tangente, etcétera). A continuación, haremos un breve repaso para tener una noción acerca de cada una de ellas y de su utilización en los scripts. Comencemos por ver la función **abs**, que devuelve el valor absoluto de un valor numérico dado. Veamos un ejemplo de uso:

```
<?php

echo abs(-123.24);

?>
```

El valor devuelto por esta función es de tipo flotante si el argumento también lo es. Caso contrario, devuelve valores enteros.

Algunas de las funciones geométricas que podemos encontrar en PHP son:

- **acos**: devuelve el arco coseno del argumento pasado, en radianes.
- **acosh**: devuelve el coseno hiperbólico inverso del argumento pasado, en radianes.
- **asin**: devuelve el arco seno del argumento pasado, en radianes.
- **asinh**: devuelve el seno hiperbólico inverso del argumento pasado, en radianes.
- **atan**: devuelve el arco tangente del argumento pasado, en radianes.

- **atan2**: calcula el arco tangente entre dos valores pasados como argumento, en radianes.
- **atanh**: devuelve la tangente hiperbólica inversa del argumento pasado, en radianes.
- **cos**: devuelve el coseno del argumento pasado, en radianes.
- **cosh**: devuelve el coseno hiperbólico del argumento pasado, en radianes.
- **hypot**: devuelve la longitud de la hipotenusa de un triángulo de ángulo recto.
- **sin**: devuelve el seno del argumento pasado, en radianes.
- **sinh**: devuelve el seno hiperbólico del argumento pasado, en radianes.
- **tan**: devuelve la tangente del argumento pasado, en radianes.
- **tanh**: devuelve la tangente hiperbólica del argumento pasado, en radianes.

De manera complementaria, para convertir de grados a radianes contamos con la función **deg2rad** y, para la operación inversa, es decir, de radianes a grados, con **rad2deg**:

```
<?php

echo rad2deg(deg2rad(90));

?>
```

Ambas funciones anteriores entregan valores en punto flotante.

La función **pi**, por su parte, nos devuelve, justamente, una aproximación al valor del número **pi**. La constante **M_PI** devuelve el mismo valor:

```
<?php

echo pi(); // 3.14159265359

?>
```

Por medio de la función **log**, podemos calcular el logaritmo natural de un número dado (correspondiente al argumento pasado):

```
<?php
echo log(1);
?>
```

Para obtener el logaritmo en base 10, contamos con la función **log10**, que recibe como argumento el número sobre el cual se trabajará:

```
<?php
echo log10(10);
?>
```

La función **pow** devuelve el resultado de elevar una base (primer argumento, no puede ser negativo) a un exponente (segundo argumento):

```
<?php
echo pow(5, 2.3); // 40.5164149173
?>
```

La función **exp** devuelve el resultado de elevar el número **e** (aproximadamente 2.71828182846) a un exponente (primer y único argumento):

```
<?php
echo exp(1);
?>
```

La función **sqrt** devuelve la raíz cuadrada de un argumento dado:

```
<?php

echo sqrt(81);

?>
```

Por su parte, la función **fmod** devuelve el módulo o resto que se obtiene al dividir dos números (primer y segundo argumento):

```
<?php

echo fmod(11, 3); // 2

?>
```

En cuanto al redondeo de valores con números decimales, contamos con tres funciones: **round**, que recibe un valor y opcionalmente, una precisión; **floor**, que recibe un valor y devuelve el siguiente valor entero más bajo, y **ceil**, que recibe un valor y devuelve el siguiente valor entero más alto. Tomemos algunos ejemplos para clarificar el uso de cada una:

```
<?php

$numero = 317.518;

echo round($numero, 2); // 317.52

echo floor($numero); // 317

echo ceil($numero); // 318

?>
```

Tenemos estas funciones de conversión entre distintas bases numéricas:

- **bindec**: de binario a decimal.
- **decbin**: de decimal a binario.
- **dechex**: de decimal a hexadecimal.
- **decoct**: de decimal a octal.
- **hexdec**: de hexadecimal a decimal.
- **octdec**: de octal a decimal.

La función **base_convert** recibe como argumento un número, una base origen (entre 2 y 36 inclusive) y una base destino (entre 2 y 36 inclusive):

```
<?php

echo base_convert('CAFE', 16, 10); // 51966

?>
```

La función **rand** devuelve un número aleatorio entre el primer y el segundo argumento, ambos opcionales. En caso de no incluirlos, se toman como límites 0 y el valor máximo definido (podemos conocerlo mediante la función **getrandmax**). En caso de no especificar el límite superior e inferior:

```
<?php
echo rand();
/*
idéntico a: echo rand(0, getrandmax());
*/
?>
```

La función **mt_rand** es similar a **rand**, pero funciona mejor y más rápido. La función **mt_getrandmax** es el equivalente a **getrandmax**. Tanto **max** como **min** reciben un conjunto de valores (puede ser un array o un listado de argumentos) y devuelven el mayor o el menor según corresponda:

```
<?php
echo max(1, 3, 99, 1029, 64);
?>
<?php
echo min(1, 3, 99, 1029, 64);
?>
```

El resultado de una operación puede exceder los límites establecidos para los números de punto flotante según la plataforma en la que estemos trabajando. Para esos casos, contamos con dos funciones que nos permiten saber si un número es demasiado grande como para ser representado (**is_infinite**) o no (**is_finite**). En cambio, para saber si el resultado no es un número, utilizamos **is_nan**. En todos los casos las funciones reciben un valor y devuelven **verdadero** o **falso**.

RESUMEN

Hicimos un recorrido por las principales funciones vinculadas al trabajo con PHP, aquellas que utilizaremos prácticamente a diario en nuestros proyectos. Algunas de las categorías vistas fueron: funciones para manejo de sesiones, cookies, cadenas de caracteres, números y matrices.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una sesión?
2. ¿Qué es una cookie?
3. ¿Cuál es la funcionalidad de **htmlspecialchars**?
4. ¿En qué casos utilizaría arrays y en qué casos no?
5. ¿Cree que las funciones numéricas son suficientes? ¿Qué cree que falta?

EJERCICIOS PRÁCTICOS

1. Genere un script que produzca palabras al azar.
2. Genere un script que calcule cuántos días faltan para una fecha determinada.
3. Genere un script que permita saber si un número es primo o no.
4. Genere un script que permita determinar si una frase es un anagrama de otra.
5. Genere un script que cuente las visitas de un usuario a cada página.

PROFESOR EN LÍNEA

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

*05

Manejo y control de errores

En este capítulo, conoceremos de qué manera PHP puede ayudarnos, a través de diferentes funciones y configuraciones, a observar de manera detallada los errores que pueden encontrarse dentro de un script. Para esto, analizaremos las configuraciones posibles del lenguaje y los distintos tipos de errores que pueden producirse.

Introducción

Es indudable que, para lograr aplicaciones robustas y estables, primero debemos someterlas a numerosas pruebas para verificar su integridad y corregir aquellas falencias que vayamos encontrando durante las distintas revisiones.

Uno de los aspectos más importantes a la hora de desarrollar sistemas es la posibilidad de detectar y corregir los errores o fallas que puedan surgir y, para esto, PHP pone a nuestra disposición una variada serie de herramientas que nos permitirán mantener un control pormenorizado de ellos. A lo largo de este capítulo, analizaremos las principales alternativas para obtener información adicional acerca de los errores y las advertencias surgidos durante la ejecución de un script.

Tipos de errores

Es posible encontrar dos clases de errores: por un lado, los que se generan en las funciones predefinidas del lenguaje y, por otro, los que ocurren en las distintas configuraciones especificadas en los archivos disponibles a través de la distribución de PHP. Al mismo tiempo, podemos clasificar los errores en dos grupos principales: los relacionados con el intérprete y los específicos del código que estamos escribiendo. PHP nos provee funcionalidades para ambos casos.

TIPOS DE APLICACIONES Y ERRORES

Es necesario destacar que el manejo de errores se encuentra orientado a cualquier clase de aplicación, en forma independiente de su alcance o de su propósito inicial. La robustez de un sistema no está relacionada, necesariamente, con la cantidad de usuarios que lo utilizan ni con cuál es su finalidad.

Dentro del primer grupo, podemos citar los errores que se producen al intentar cargar una determinada extensión o los que tienen lugar por la ausencia de un determinado archivo esencial para el funcionamiento del intérprete. Los errores relacionados con la escritura del código probablemente se deban a la utilización de funciones no definidas o a problemas de la sintaxis demandada por el lenguaje en sí.

> UN SISTEMA NO DEBERÍA SUFRIR GRANDES CAMBIOS MIENTRAS ESTÁ EN PRODUCCIÓN

Ciclo de un sistema

El ciclo de un sistema está compuesto por varias fases que dependen del punto de vista desde el cual se lo analice. En el caso específico de los errores, podríamos categorizar dos puntos bien diferenciados: por un lado, la etapa de desarrollo y, por otro, la de producción. Durante el desarrollo de una aplicación, se realizan todas las pruebas y modificaciones que creemos necesarias para perfeccionar el funcionamiento y lograr que los resultados obtenidos se acerquen lo más posible a lo planteado en un principio. Una vez concluida esta etapa, el siguiente paso será publicar el sitio y hacerlo accesible a los usuarios que lo utilizarán. Idealmente, un sistema no debería sufrir modificaciones drásticas mientras está en producción.

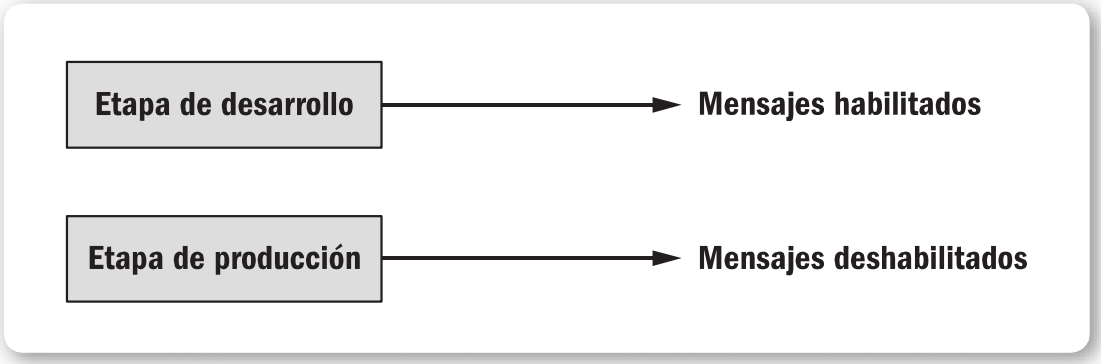


Figura 1. Según la etapa de desarrollo en la cual nos encontremos, deberemos optar por una política para la exhibición de mensajes de error y advertencias.

Opciones del lenguaje

Dentro de lo que es el lenguaje en sí, PHP cuenta con una serie de herramientas que nos permitirán aspirar a mantener aplicaciones libres de errores a un bajo costo. Esto se debe a su potencia y claridad, y a que brinda grandes posibilidades para el control y la solución rápida de posibles fallas en los sistemas. Veremos cómo el lenguaje mantiene su filosofía de simplificar los procesos de desarrollo al darle una libertad extrema al programador, pero, a la vez, limitando esos márgenes cuando cree conveniente hacerlo.

Configuraciones posibles

PHP incluye en su distribución oficial un archivo de configuración denominado **php.ini**, desde el cual podremos definir, en gran parte, el comportamiento del intérprete en variadas situaciones. Este archivo está compuesto por directivas de configuración, que tienen asignados valores:

```
nombre_directiva = valor
```

Las líneas que comienzan por un **;** (punto y coma) se toman como comentadas, es decir, el intérprete PHP no las tendrá en cuenta.

En los servicios de alojamiento (proveedores de hosting), este archivo,

LIBERTADES

Las características de PHP hacen que el programador se sienta libre en lo que respecta a cómo escribir el código de las aplicaciones porque cuenta con una gran cantidad de opciones. Esto es una ventaja por un lado, pero por otro nos obliga a mantener una línea en cuanto a cómo conservar desarrollos prolijos y comprensibles.

en general, no está disponible para ser modificado o actualizado por parte de los autores de páginas web, pero, como veremos en breve, por medio de un script es posible modificar algunos valores de sus directivas.

Con relación al manejo y control de errores, específicamente, en el archivo de configuración **php.ini** se incluyen directivas para definir distintas opciones, entre las que podemos encontrar las siguientes:

- **error_reporting**
- **display_errors**
- **display_startup_errors**
- **log_errors**
- **error_log**
- **log_errors_max_len**
- **ignore_repeated_errors**
- **ignore_repeated_source**
- **track_errors**
- **html_errors**
- **error_prepend_string**
- **error_append_string**

En PHP existen los llamados **niveles de error** que nos permiten seleccionar qué clases de errores se informarán en las salidas. Estos niveles están definidos a través de constantes, entre las cuales podemos citar las siguientes:

CONSTANTES DE NIVELES DE ERROR

▼ CONSTANTE	▼ DESCRIPCIÓN
E_ALL	Muestra todos los errores y advertencias, excepto los correspondientes a **E_STRICT**.
E_COMPILE_ERROR	Emite mensajes acerca de errores fatales en tiempo de compilación.

CONSTANTES DE NIVELES DE ERROR (CONTINUACIÓN)

E_COMPILE_WARNING	Similar a **E_COMPILE_ERROR**, solo que se muestran únicamente las advertencias ante eventuales errores.
E_CORE_ERROR	Trata errores fatales producidos durante el inicio de PHP (carga de librerías, directivas, etcétera).
E_CORE_WARNING	Es similar a la anterior, pero muestra solo las advertencias ante eventuales errores.
E_ERROR	Emite mensajes sobre errores fatales en tiempo de ejecución. Si omitimos esta opción, los errores fatales serán exhibidos de todos modos.
E_NOTICE	Se encarga de los errores no críticos en el código fuente de la aplicación (variables no inicializadas, índices de arrays inexistentes, etcétera).
E_PARSE	Emite advertencias ante errores de compilación internos.
E_RECOVERABLE_ERROR	A partir de PHP versión 6, se incluye este nivel, que podremos utilizar en lugar de **E_ERROR** para identificar errores de los que el sistema puede recuperarse, es decir, aquellos que producen una inestabilidad no definitiva.
E_STRICT	Está disponible a partir de PHP 5 y se enfoca en la compatibilidad del código escrito con relación a versiones anteriores del lenguaje.
E_USER_ERROR	Es similar a **E_ERROR**, solo que podemos personalizar los mensajes.
E_USER_NOTICE	Es similar a **E_NOTICE**, pero es posible personalizar los mensajes.
E_USER_WARNING	Es similar a **E_WARNING**, solo que podemos personalizar los mensajes.

CONSTANTES DE NIVELES DE ERROR (CONTINUACIÓN)	
E_WARNING	Imprime advertencias en tiempo de ejecución y errores no fatales.

Tabla 1. Constantes para la directiva **error_reporting**.

Estos valores pueden ser utilizados en la directiva **error_reporting**, y es posible definirlos mediante el uso de los operadores disponibles:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

Los distintos niveles pueden combinarse por medio de operadores, para lograr una personalización avanzada según nuestras necesidades:

- ~ (negación)
- | (alternativa)
- & (concatenación)

Veamos algunos ejemplos de aplicación:

```
error_reporting = E_ALL

error_reporting = E_ALL &~(E_NOTICE | E_WARNING)

error_reporting = E_COMPILE_ERROR | E_ERROR | E_CORE_ERROR
```

Por defecto, PHP utiliza una combinación de constantes que nos permite visualizar todos los errores y advertencias (**E_ALL**), excepto aquellos que no son críticos para el sistema (**E_NOTICE**):

```
error_reporting = E_ALL & ~E_NOTICE
```

La deshabilitación explícita de **E_NOTICE** se basa en el hecho de que, por lo general y dadas las características del lenguaje, en las aplicaciones las variables no suelen ser declaradas o inicializadas o normalmente se invocan posiciones inexistentes dentro de una matriz. Estas inconsistencias dentro del código, que rara vez influyen sobre el correcto funcionamiento del programa, hacen que se deshabilite la opción dentro de error_reporting. Como mencionamos algunos párrafos atrás, PHP brinda una libertad considerada por algunos como excesiva, que puede ser restringida a través de este nivel de error.

> DISPLAY_ERRORS INFORMA AL INTÉRPRETE SI TIENE QUE EMITIR MENSAJES DE ERROR

Pues bien, todo lo visto hasta ahora tendrá validez únicamente si la directiva **display_errors** está habilitada. Esta opción se encarga de informar al intérprete si tiene que emitir mensajes de error (para esto tiene en cuenta el nivel declarado en **error_reporting**) o no.

Puede tomar dos valores: **On** o **1** para habilitar mensajes u **Off** o **0** para deshabilitarlos. El valor por defecto de esta directiva es **On**:

```
display_errors = On
```

Hay ciertos errores que no están incluidos en los alcances de **display_errors** y, para complementar su funcionamiento, contamos con la directiva **display_startup_errors**, que nos da la posibilidad de mostrar errores producidos durante la carga de PHP.

```
display_startup_errors = Off
```

Algunos casos en los cuales se despliega este tipo de errores podrían ser la inexistencia de algún directorio especificado en el archivo **php.ini** o también

extensiones habilitadas no halladas. Como en el caso anterior, puede tomar dos valores: **On** o **1** para habilitar mensajes u **Off** o **0** para deshabilitarlos. El valor por defecto de esta opción es **Off** y, desde el archivo **php.ini**, se recomienda mantenerlo así, al menos durante la etapa de producción.

Los mensajes de error pueden ser observados a través de la salida estándar (por ejemplo, un navegador web) o desde un registro mantenido en un archivo de texto plano, generado y actualizado de manera automática por el servidor. Estos archivos nos serán de gran ayuda, y, en caso de poder contar con ellos (nuevamente, desde un servicio de alojamiento puede ser que no tengamos acceso), podremos consultarlos y obtener un detalle preciso acerca de las ocurrencias generadas. Entre las informaciones disponibles tenemos la fecha, la hora, el nivel y la descripción del error. Para registrarlos existe la directiva **log_errors**, que por defecto está deshabilitada:

```
log_errors = Off
```

Esta clase de almacenamiento silencioso nos permite mantener un control detallado de los sucesos acontecidos y, a la vez, no interferir la salida estándar del navegador con mensajes. Es útil durante la etapa de producción de un sitio. En la instalación clásica de Apache, podemos ubicar este archivo dentro del directorio **logs**. El documento que contiene los errores normalmente se llamará **error.log**, que, al ser un archivo de texto plano,

MIGRACIÓN

Al migrar un sistema de un servidor a otro, es probable que surjan inconvenientes relacionados con las versiones de los lenguajes utilizados o con las configuraciones propias de cada herramienta. Estos errores deberán ser capturados y resueltos de manera rápida para no afectar las aplicaciones.

podrá visualizarse con cualquier editor.Para esto, el servidor deberá estar inactivo. Si queremos especificar otro nombre,disponemos de la directiva **error_log**, cuyo valor es la ruta hacia el archivo:

```
error_log = c:/www/logs/php_error.log
```

Esto puede ser de utilidad ya que, por defecto, Apache utiliza el mismo archivo para ubicar y almacenar su propio registro de errores, es decir, aquellos relacionados con las tareas del servidor web.

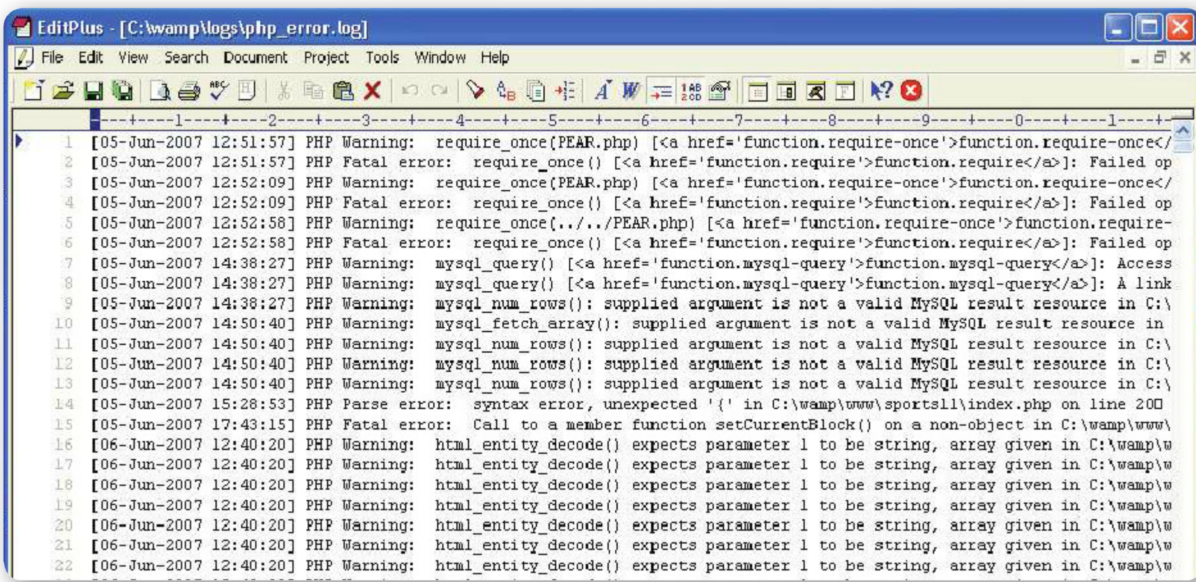


Figura 2. El archivo de registro y almacenamiento de errores puede ser de utilidad a la hora de precisar el momento y el lugar de su ocurrencia.

Si necesitamos determinar o limitar el tamaño máximo del registro en bytes, tenemos que definir la directiva **log_errors_max_len**. Su valor por defecto es de **1024 bytes**. Si no queremos imponer ningún límite, simplemente le asignamos **0**.

```
log_errors_max_len = 1024
```

En algunas circunstancias, por ejemplo, durante la ejecución de ciclos **for**, **while**, **do while**, **foreach**, etcétera, un mismo error se producirá un número **N** de veces, y PHP puede ayudarnos a mantener cierta claridad al respecto para no mostrar ni almacenar N mensajes, sino solo uno. Con la opción **ignore_repeated_errors** habilitada, PHP nos mostrará los mismos errores solamente una vez. Para considerarse repetidos, deben ocurrir en la misma línea, en el mismo archivo y, normalmente, estarán dentro de ciclos.

```
ignore_repeated_errors = Off
```

Con relación a esto, la directiva **ignore_repeated_source** modifica la definición de lo que es un error repetido, no tiene en cuenta el origen de los errores, es decir, el archivo en el que se produjeron.

```
ignore_repeated_source = Off
```

La directiva **track_errors** almacena el último mensaje de error o advertencia en la variable **php_errormsg**. Está deshabilitada por defecto en el archivo **php.ini**:

```
track_errors = Off
```

CAMBIOS EN EL ARCHIVO PHP.INI

Debemos recordar que, para hacer valer los cambios realizados en el archivo de configuración **php.ini**, tendremos que reiniciar el servidor web. En los servicios de alojamiento compartido, esta opción no estará disponible, pero sí podremos aplicarla cuando trabajemos con un servidor local.

En el siguiente ejemplo, intentamos realizar una división por 0:

```
<?php

@$variable = 100 / $null;

if ($php_errormsg) {
        echo $php_errormsg;
        exit;
}

//salida: Division by zero

?>
```

Observemos también este código de ejemplo:

```
<HTML>
<HEAD>
<TITLE>ejemplo de track_errors</TITLE>
</HEAD>

<BODY>

<?php

echo $matriz_no_definida[34];
echo "<br>--------------------------";

echo '<br><br>Mensaje del ultimo error cometido: '.$php_errormsg;

echo "<br><br>--------------------------";
```

```
?>

</BODY>
</HTML>
```

Su salida será:

```
--------------------------
Mensaje del ultimo error cometido: Undefined variable: matriz_no_definida
--------------------------
```

En ocasiones, PHP brinda más información sobre un error en particular en su manual oficial, para obtener una visión completa de lo que está sucediendo y, de esta manera, poder resolver sin demoras la falta cometida. Para que el lenguaje nos muestre un mensaje con un enlace al detalle, debemos tener activada la directiva **html_errors**:

```
html_errors = On
```

Puede tomar dos valores: **On** o **1** para habilitar mensajes HTML u **Off** o **0** para deshabilitarlos. El valor por defecto es **Off**, y, desde el archivo **php.ini**,

HOSTING

Es habitual que, en los servicios de alojamiento web, no contemos con la posibilidad de acceder y modificar el archivo **php.ini**. En este caso, realizar las configuraciones necesarias a través de **error_reporting** e **ini_set** nos será de gran ayuda para lograr el nivel de reporte de errores buscado.

se recomienda mantenerlo así, al menos durante la etapa de producción. Observemos el siguiente ejemplo:

```
<HTML>
<HEAD>
<TITLE>ejemplo de html_errors</TITLE>
</HEAD>

<BODY>

<?php

mysql_query($var);

?>

</BODY>
</HTML>
```

También es posible definir la ruta al manual oficial y su extensión por medio de las directivas **docref_root** y **docref_ext**. Si utilizamos una copia local del manual (podemos descargar la última disponible desde el sitio **www.php.net/docs.php**), es posible referenciarla tomando como base el directorio raíz del servidor:

```
docref_root = "/manual/"
```

En la extensión del manual siempre debe incluirse el punto:

```
docref_ext = .html
```

Para anexar contenidos antes y después de los mensajes de error, existen las directivas **error_prepend_string** y **error_append_string**, respectivamente, que nos permitirán personalizar el formato de la presentación de los mensajes de error. Aquí podemos incluir cualquier fragmento HTML, como por ejemplo:

```
error_prepend_string = "<font color='FF0000'>"

error_append_string = "</font>"
```

La directiva **error_prepend_string** nos servirá para mostrar contenidos antes de los mensajes de error. Si escribimos textos en formato HTML, esta los soportará y mostrará la salida correspondiente. Es una directiva interesante en el sentido de que nos permite personalizar el formato o la presentación de los mensajes de error para adecuarlos a nuestras necesidades. No hace falta habilitar o deshabilitar expresamente esta directiva: bastará con asignarle valores o no. En el caso de asignarle algún texto, se considerará habilitada y, en el caso de no asignarle nada, se considerará deshabilitada. Un ejemplo es:

```
error_prepend_string = "<b>Error!</b><br><font color=FFCC00>"
```

VERIFICACIÓN

Siempre es recomendable, en los equipos que utilizamos para desarrollar sistemas, mantener el nivel de errores configurado para que se muestren por todos los medios posibles. Una de las tareas de la etapa de pruebas consiste en encontrar y resolver cualquier clase de inconveniente.

En este caso, cada vez que se produzca un error, antes de mostrarlo por pantalla, se imprimirá el texto **Error!** en negrita. El mensaje se imprimirá con sus caracteres en naranja, porque abrimos la etiqueta **<font color=FFCC00>** en la directiva. Probemos el siguiente código y observemos el resultado:

```
<HTML>
<HEAD>
<TITLE>ejemplo de error_prepend_string</TITLE>
</HEAD>

<BODY>

<?php

echo $matriz[34;

?>

</BODY>
</HTML>
```

MÁS ACERCA DE EXCEPCIONES

PEAR (**http://pear.php.net**) es el repositorio oficial de extensiones y aplicaciones para el lenguaje PHP y ofrece una clase que nos permite implementar excepciones. Funciona para las versiones 5 y superiores del lenguaje, y se denomina **PEAR_Exception**. Para mayor información, debemos ingresar a su sitio web.

La directiva **error_append_string** es similar a la anterior, pero con la diferencia de que el contenido se imprimirá luego del mensaje de error. Supongamos que tenemos la siguiente directiva:

```
error_append_string = "</font><br>Final del mensaje de error."
```

Probemos este código y observemos el resultado:

```
<HTML>
<HEAD>
<TITLE>ejemplo de error_append_string</TITLE>
</HEAD>

<BODY>

<?php

echo $matriz[34;

?>

</BODY>
</HTML>
```

Hay lenguajes, cuya sintaxis es muy parecida a la de PHP, que utilizan el signo más (**+**) en lugar del punto (**.**) para unir cadenas de caracteres o variables dentro de un script. La directiva **warn_plus_overloading** nos informa, en caso de estar habilitada, a través de una advertencia, cuando usamos el signo más (**+**) para concatenar cadenas en lugar del utilizado por PHP, que es el punto (**.**). Como vimos, muchas de las directivas están relacionadas con el

informe de errores para que el desarrollador pueda tomar nota y corregirlos sin demoras. Esto, que sin dudas es de suma utilidad y absolutamente necesario durante el ciclo de desarrollo de un sistema, no lo es para la etapa de producción, en la cual se deberán deshabilitar todas las opciones vinculadas a la exhibición de errores por pantalla. Esto tiene que ver, principalmente, con dos cuestiones: por un lado, preservar la estética de las páginas del sitio y, por otro, prevenir la exhibición de información que eventualmente puede ser tomada por usuarios que busquen explotar los posibles puntos débiles del sitio, con mala intención o simplemente por curiosidad. La exhibición de mensajes de error puede brindar información sensible, por ejemplo, la ubicación de determinados archivos o, peor aún, la presentación de datos reservados.

Funciones del lenguaje

Como hemos visto en la sección anterior, el archivo **php.ini** admite múltiples directivas para manipular todo lo vinculado a los mensajes de advertencia en caso de surgir errores en los scripts. Para complementar este panorama y disponer de herramientas en caso de no tener acceso directo a la modificación de este archivo, PHP mantiene una serie de funciones incorporadas en la distribución estándar del lenguaje. A continuación, veremos cuáles son y cómo funcionan.

Tomaremos como primer caso **ini_set**, que recibe como argumentos el nombre de la directiva del archivo **php.ini** que queremos modificar y el valor correspondiente para asignarle:

```
ini_set('display_errors', 'On');
```

Al momento de utilizar esta función, deberemos tener en cuenta dos cuestiones importantes: primero, que los cambios realizados solo tendrán validez durante la ejecución del script desde el cual se invoque y, segundo, que por cuestiones de seguridad no todas las directivas pueden modificarse. Encontraremos más información sobre las directivas permitidas, en el manual de PHP.

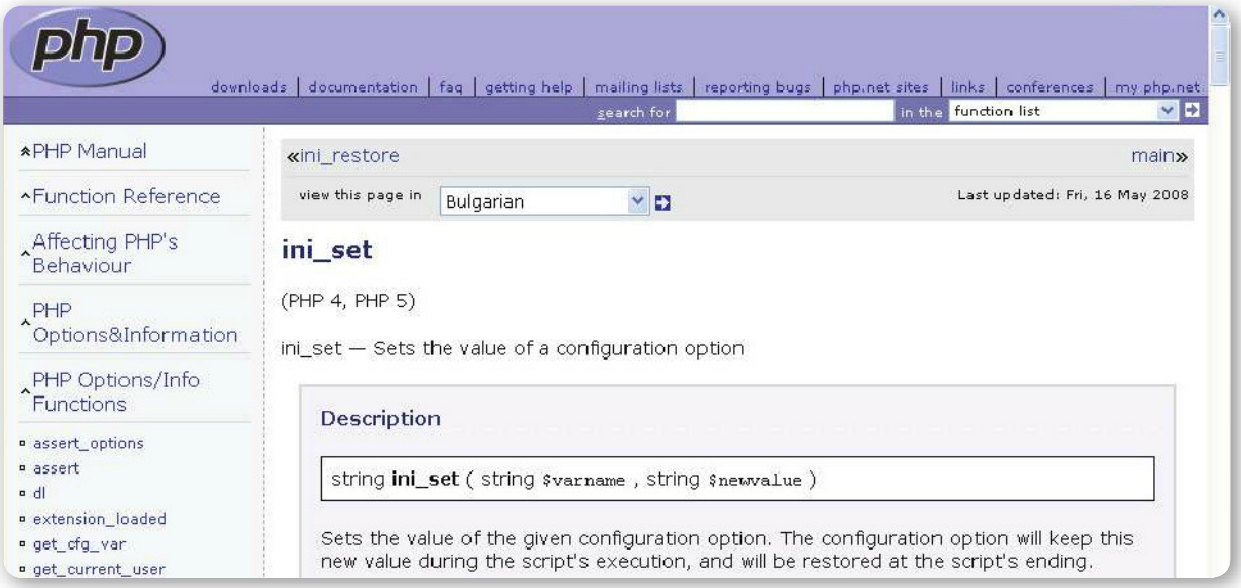


Figura 3. La función **`ini_set`** puede modificar en tiempo de ejecución los valores de algunas directivas del archivo **`php.ini`**.

Esta función no modifica el archivo, sino que reemplaza o sobrescribe de manera temporal sus valores con relación al script actual.

Otra función disponible es **error_reporting** (se denomina igual que la directiva vista anteriormente), que recibe como argumento el nivel de error que queremos establecer. Los valores aceptados son:

ARGUMENTOS DE LA DIRECTIVA ERROR_REPORTING

VALOR	CONSTANTE DE LA DIRECTIVA
1	**E_ERROR**
2	**E_WARNING**
4	**E_PARSE**
8	**E_NOTICE**
16	**E_CORE_ERROR**

ARGUMENTOS DE LA DIRECTIVA ERROR_REPORTING (CONTINUACIÓN)	
32	**E_CORE_WARNING**
64	**E_COMPILE_ERROR**
128	**E_COMPILE_WARNING**
256	**E_USER_ERROR**
512	**E_USER_WARNING**
1024	**E_USER_NOTICE**
2047	**E_ALL**
2048	**E_STRICT**

Tabla 2. Opciones para la función **error_reporting**.

Para seleccionar varias opciones al mismo tiempo, como en la directiva del mismo nombre, en lugar de utilizar operadores debemos sumar los valores enteros correspondientes a las constantes. Por ejemplo, si quisiéramos utilizar las opciones **E_ERROR** y **E_WARNING**:

```
error_reporting(3); //1 + 2
```

Esto sería equivalente a la siguiente línea:

```
ini_set('error_reporting', 'E_ERROR & E_WARNING');
```

Esta función devuelve el valor anterior de la directiva y no modifica el archivo **php.ini**, sino que reemplaza o sobrescribe de manera temporal sus valores con relación al script actual.

Excepciones

A partir de PHP versión 5, se incluye soporte para el manejo de excepciones, que pueden explicarse como sucesos o acontecimientos que interrumpen la ejecución de una aplicación. Una **excepción** nos da la posibilidad de capturar esos sucesos y evitar la finalización abrupta de la aplicación.

Contamos con el bloque **try/catch** para intentar capturar y lanzar excepciones para el control de errores. Las funciones disponibles para el tratamiento y el control de excepciones son las siguientes:

- **trigger_error**: genera un mensaje de error personalizado por parte del usuario. Solo algunos niveles de error pueden ser interceptados por esta función (**E_USER_NOTICE**, **E_USER_WARNING**, y **E_USER_ERROR**):

```
<?php

if ($error) {
   trigger_error("Aquí el mensaje", E_USER_NOTICE);
}

?>
```

AYUDA ACERCA DE ERRORES EN PHP

Algunos lenguajes de programación traen, junto con sus entornos de desarrollo, la ayuda del lenguaje. Como PHP no posee un entorno de desarrollo único, es posible que tengamos que descargar la ayuda desde el sitio web de PHP, ya que tampoco viene junto con la distribución oficial. La ayuda está disponible en múltiples formatos.

- **set_error_handler**: permite cambiar el manejador de errores por defecto de PHP por uno personalizado. Tiene relación con **set_exception_handler**, que crea una función de respuesta para aquellas situaciones en las que una excepción se produzca sin haber sido capturada:

```
<?php

set_error_handler("nuevoManejador");

try {
fopen('archivo.txt', 'r');
} catch (Exception $error) {
echo $error->getMessage();
}

restore_error_handler();

function nuevoManejador($codigoError, $mensajeError) {
throw new Exception("No es posible abrir el archivo solicitado ...");
}

?>
```

MANUAL

Para profundizar, podemos encontrar más información sobre las excepciones y su inclusión en scripts en las siguientes secciones del manual oficial de PHP: **www.php.net/exceptions** y **www.php.net/errorfunc**, en donde, además, se incluyen ejemplos de utilización.

```php
<?php

set_error_handler("nuevoManejador");

try {
    fopen('archivo.txt', 'r');
} catch (Exception $error) {
    echo $error->getMessage();
}

restore_error_handler();

function nuevoManejador($codigoError, $mensajeError) {
    throw new Exception("No es posible abrir el archivo solicitado ...");
}
```

Figura 4. Las excepciones nos permiten evitar el colapso de una aplicación, al capturar y solucionar los posibles inconvenientes surgidos durante su ejecución.

Primero definimos que vamos a utilizar el manejador **nuevoManejador** en lugar del de PHP:

```
set_error_handler("nuevoManejador");
```

Esta función crea una excepción que, al lanzarse, evita la interrupción del programa:

```
function nuevoManejador($codigoError, $mensajeError) {
throw new Exception("No es posible abrir el archivo solicitado ...");
}
```

Intentamos (**try**) abrir un archivo. En caso de no ser posible, capturamos (**catch**) la excepción y emitimos un mensaje de error:

```
try {
fopen('archivo.txt', 'r');
} catch (Exception $error) {
echo $error->getMessage();
}
```

Por último, devolvemos el control de errores a PHP:

```
restore_error_handler();
```

Además de **getMessage**, contamos con otros métodos entre los que podemos citar los siguientes:

MÉTODOS PARA EMITIR MENSAJE DE ERROR

▼ MÉTODO	▼ DESCRIPCIÓN
getCode	Indica el código de error.
getFile	Muestra la ruta al archivo en el cual se generó la excepción.
getLine	Devuelve la línea en la cual se generó la excepción.
getTrace	Información acerca del desarrollo de la excepción (array).
getTraceAsString	Información acerca del desarrollo de la excepción (cadena de caracteres).

Tabla 3. Métodos para el tratamiento de excepciones.

Las excepciones están directamente ligadas a la programación orientada a objetos. Cuando una excepción ocurre dentro de un método, este crea un objeto que guarda información acerca de la excepción.

En el siguiente ejemplo, mediante la función **set_error_handler**, se establece una función para gestionar los errores producidos en una división. Al invocar el método **getMessage**, en lugar de mostrar el mensaje predeterminado de PHP, se pasará el control a la función **division_error_handler**. Con **restore_error_handler**, se devuelve el control a PHP. Veamos los variados ejemplos de manejos de excepciones.

A continuación, vemos el ejemplo 1:

```
<?php

$a = 10;
$b = 0;

set_error_handler("division_error_handler");

try {
echo "Resultado: ".($a / $b) ."<br>";
} catch (Exception $e) {
echo $e->getMessage()." ("".$a." / ".$b.")<br>";
}

restore_error_handler();

functiondivision_error_handler($errno, $errmsg) {
throw new Exception("No se puede calcular resultado: ");
}

//salida:
//No se puede calcular resultado: (10 / 0)

?>
```

Ejemplo 2:

```
<?php

$dom = new domdocument();
$dom->loadXML("<a><b>BBB</b><c>CCC</c></a>");

$ne = $dom->documentElement;
```

```
try {
  $ne->appendChild($ne);
} catch (domexception $e) {
echo "Error : ";
echo "No se puede añadir el nodo '$ne->nodeName'";
}

?>
```

Ejemplo 3:

```
<?php

functionnumeroTelefonico($indice = 0) {
set_error_handler("numeroTelefonico_eh");

  $o[] = '0800-23455-1222';
  $o[] = '0800-23455-1223';
  $o[] = '0800-23455-1224';
  $o[] = '0800-23455-1225';

try {
echo "Opcion seleccionada: ".$o[$indice]."<br>";
} catch (Exception $e) {
echo "Error: ".$e->getMessage();
  }
}

numeroTelefonico(1);
numeroTelefonico(10);
numeroTelefonico(2);
numeroTelefonico(0);
numeroTelefonico();

functionnumeroTelefonico_eh($errno, $errmsg) {
throw new Exception("Opcioninexistente.<br>");
```

```
}

//salida:
//Opcion seleccionada: 0800-23455-1223
//Error: Opcion inexistente.
//Opcion seleccionada: 0800-23455-1224
//Opcion seleccionada: 0800-23455-1222
//Opcion seleccionada: 0800-23455-1222

?>
```

Ejemplo 4:

```
try {
if ( !@mysql_connect('localhost', 'user', 'pass'))
throw new Exception (mysql_error());
} catch (Exception $e) {
echo 'no es posible conectarse al servidor MySQL';
}
```

Ejemplo 5:

```
<?php

function eh($excepcion) {
echo "Mensaje personalizado para excepciones no capturadas: #".$excepcion-
>getMessage()."<br>";
}

set_exception_handler('eh');

$temp[] = '';
```

```
if (!is_array($temp))              {
      throw new Exception('1');
}    e      els                                  {
      if (count($temp)<2)          {
      throw new Exception('2');
      }
}

?>
```

Ejemplo 6:

```
<?php

classarchivo {
functioncargar($ruta) {

if (!$ruta) {
throw new Exception("Archivo no especicado!");
   }

if (!file_exists($ruta)) {
throw new Exception("Archivo no encontrado!");
   }

returnfile_get_contents($ruta);
 }
}

try {
 $archivo = new archivo();
$archivo->cargar('archivo.txt');
} catch (Exception $e) {
print $e->getMessage();
}

?>
```

Además de **getMessage()**, es posible especificar:

- **getCode()**: el código de error.
- **getFile()**: la ruta al archivo en el cual se generó la excepción.
- **getLine()**: la línea en la cual se generó la excepción.
- **getTrace()**: array multidimensional con información acerca del desarrollo de la excepción.
- **getTraceAsString()**: como **getTrace()**, pero en formato de cadena de caracteres.

Register globals

> A PARTIR DE LA VERSIÓN 4.2.0, PHP MODIFICÓ EL VALOR POR DEFECTO DE REGISTER_GLOBALS

A partir de la versión 4.2.0, el lenguaje PHP modificó el valor por defecto de la directiva **register_globals**, lo que suscitó problemas de compatibilidad en las aplicaciones escritas que tomaban como referencia el valor anterior.

Siempre existió la posibilidad de configurar esta directiva tal como lo podemos hacer hoy en día. El cambio se produjo a partir de que los desarrolladores de PHP y la comunidad de usuarios notaron que su valor predeterminado podía traer inconvenientes relacionados con la seguridad de los sitios web. Entonces, el paso siguiente fue modificar este valor y recomendar a los usuarios mantenerlo así.

Ahora bien, este cambio tuvo repercusiones en la manera de programar las aplicaciones, por lo que existió la necesidad de modificar algunos aspectos del código programado.

Al usar **register_globals = On**, PHP asume todas las variables como globales y no podrá diferenciar entre los tipos de variables (si usa una variable en su script que tiene el mismo nombre de una variable de sesión, PHP no podrá diferenciarlas y las tratará como a una sola). Para solucionar estos inconvenientes y trabajar con **register_globals = Off**, el lenguaje ofrece los siguientes arrays:

- **$_SERVER**: contiene una serie de variables generadas por el servidor web (distintos servidores pueden admitir valores diferentes para las mismas variables disponibles en **$_SERVER**). Algunas de las variables contenidas en este array son **REMOTE_ADDR** (dirección IP del usuario), **SERVER_NAME** (nombre del servidor web) y **PHP_SELF** (nombre del archivo que se está ejecutando actualmente). Accedemos a ellas de la siguiente manera:

```
echo $_SERVER[PHP_SELF];
```

- **$_GET**: almacena las variables pasadas por URL (por ejemplo, a través de enlaces o de formularios que utilizan el método GET). Quizás sea aquí donde se vea de manera clara cuál es el problema de seguridad que implica usar **register_globals = On**. Supongamos que tenemos un script para autentificar usuarios y pasamos la variable por URL de la siguiente manera:

```
http://www.misitioweb.com?usuario=registrado
```

En este caso, cualquiera estaría registrado (PHP no necesita declarar previamente las variables). Con **register_globals = Off** podríamos especificar el modo en que recibiríamos la variable. Si establecemos que la variable será de tipo sesión (accesible mediante **$_SESSION**), no nos importa el valor contenido en **$_GET[usuario]**, sino el valor que contenga **$_SESSION[usuario]**.

- **$_POST**: idéntica en su uso a **$_GET**. Almacena las variables recibidas por el método **POST**, por ejemplo, a través de formularios.
- **$_COOKIE**: una matriz asociativa de variables pasadas al script actual a través de cookies HTTP.
- **$_SESSION**: almacena las variables de tipo sesión.
- **$_FILES**: aquí se guardan algunas informaciones acerca de los archivos enviados a través de formularios.
- **$_ENV**: las variables que se incluyen en este array dependen del

sistema operativo sobre el que se trabaja. Otras variables de entorno incluyen las variables **CGI**.

- **$_REQUEST**: una matriz asociativa que guarda los contenidos de **$_GET**, **$_POST** y **$_COOKIE**.

Estos arrays fueron implementados recién en la versión 4.1.0 de PHP; anteriormente, se contaba con los siguientes:

- **$HTTP_SERVER_VARS**
- **$HTTP_GET_VARS**
- **$HTTP_POST_VARS**
- **$HTTP_COOKIE_VARS**
- **$HTTP_POST_FILES**
- **$HTTP_ENV_VARS**
- **$HTTP_SESSION_VARS**

No existe el equivalente a **$_REQUEST** en versiones anteriores a la 4.1.0. La idea es identificar el método por el cual recibimos las variables y acceder a ellas mediante el array correspondiente. Por ejemplo, si enviamos variables a una página a través de un formulario, debemos recuperarlas mediante el array correspondiente al método utilizado en el formulario:

```
<!--pagina1.php                                    -->

<form action=pagina2.php method=POST>
<input type=text name=nombre>
<input type=submit>
</form>

<!--pagina2.php                                    -->

<?php

echo "la variable nombre contiene el valor : ".$_POST[nombre];

?>
```

Si trabajamos con los arrays que nos provee PHP a partir de su versión 4.1.0, los scripts funcionarán tanto con **register_globals = On** como con **register_globals = Off**. Para saber qué valor tiene la directiva **register_globals** en nuestro sistema, disponemos de dos posibilidades. La primera es acceder al archivo **php.ini** buscando la línea que contenga algo parecido a lo que muestra la **figura 5**.

```
;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; PHP's initialization file, generally called php.ini, is
responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a number
of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on Windows (As of PHP
5.2.0)
; 4. Current working directory (except CLI)
; 5. The web server's directory (for SAPI modules), or directory
of PHP
; (otherwise in Windows)
; 6. The directory from the --with-config-file-path compile time
option, or the
; Windows directory (C:\windows or C:\winnt)
; See the PHP docs for more specific information.
; http://php.net/configuration.file
```

Figura 5. El valor por defecto de la directiva **register_globals** fue uno de los grandes cambios a partir de PHP 4.1.0.

La segunda es utilizar la función **phpinfo**, como vemos en la **figura 6**.

Hay soluciones para los códigos programados con **register_globals = On**, que recorren todas las variables contenidas en los arrays vistos anteriormente y copian sus valores a variables:

```
$nombre = $_GET[nombre];
```

Otra posibilidad se da mediante la función **ini_set**:

```
ini_set("register_globals","1");
```

```
2  /* vim: set expandtab sw=4 ts=4 sts=4: */
3  /**
4   * phpinfo() wrapper to allow displaying only when configured to do so.
5   *
6   * @package PhpMyAdmin
7   */
8
9  /**
10  * Gets core libraries and defines some variables
11  */
12 require_once 'libraries/common.inc.php';
13 PMA_Response::getInstance()->disable();
14
15 /**
16  * Displays PHP information
```

Figura 6. Mediante la función **phpinfo**, es posible visualizar el valor de la directiva **register_globals**.

Recordemos que el cambio se produjo, esencialmente, por motivos de seguridad y que es necesario trabajar con los arrays provistos por PHP.

Las dos líneas de código anteriores mantienen el funcionamiento de las aplicaciones programadas, utilizando **register_globals = On**, y que desean pasar a **register_globals = Off** sin modificar casi nada del código. Pero el problema de seguridad alertado por PHP seguirá latente.

RESUMEN

Analizamos las principales funciones y directivas de configuración para encontrar y corregir errores y, de esta manera, poder mantener seguras y bien formadas nuestras aplicaciones de PHP. Hicimos un recorrido por las funciones propias del lenguaje y por las directivas disponibles dentro del archivo **php.ini**. También vimos en qué momento nos conviene hacer uso de las directivas configuradas en este archivo y aprendimos cómo cambiar el valor de dichas directivas sin modificar el contenido del archivo principal.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿En qué consiste la etapa de desarrollo de un sistema?
2. ¿Qué es una excepción?
3. ¿Para qué sirve la función **error_reporting**?
4. ¿Para qué sirve la directiva **error_reporting**?
5. ¿En qué casos utilizaría el nivel **E_NOTICE**?

EJERCICIOS PRÁCTICOS

1. Averigüe cuál es el nivel de exhibición de errores de su sistema.
2. Revise su archivo de errores y visualice sus últimas líneas.
3. ¿Cuáles son las directivas modificables a través de la función **ini_set**?
4. Trate de generar de manera intencional un error capturado por medio de la directiva **display_startup_errors**.
5. Modifique su sistema para utilizar las ventajas de la directiva **html_errors.**

PROFESOR EN LÍNEA

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

Generación de gráficos

La inclusión de gráficos en aplicaciones web es habitual, y el número de librerías que nos permiten agilizar el proceso de generación y manipulación de imágenes crece día a día. Observaremos cómo funcionan algunas de ellas y analizaremos en qué aspectos pueden favorecer nuestros desarrollos.

La librería GD

El lenguaje PHP provee una extensión para utilizar la librería **GD** (**www.boutell.com/gd**) a través de funciones predeterminadas. Esta librería se utiliza sobre todo para generar gráficos en tiempo de ejecución. Permite dinamizar las salidas valiéndose de los ingresos de los usuarios o de datos almacenados en bases de datos o archivos de configuración.

Para su correcto funcionamiento, **GD** requiere que contemos con PHP versión 4.3.2 o superior (la librería viene incluida en la distribución estándar). Además, deberá estar habilitada en el archivo **php.ini**:

```
extension=php_gd2.dll
```

Para comprobar si disponemos o no de la librería, tenemos varias opciones:

- Utilizar la función **phpinfo**:

```
<?phpphpinfo(); ?>
```

- Verificar que esté habilitada alguna de las funciones de extensión:

```
<?php

if (function_exists("gd_info")) {
echo "GD esta disponible";
echo '<pre>';
print_r(gd_info());
echo '</pre>';
} else {
echo "GD no esta disponible !";
}

?>
```

En instalaciones que requieren la compilación de PHP, deberán estar incluidas las siguientes opciones:

```
--with-gd
--with-jpeg-dir=/usr
--with-png
--with-ttf
```

Para verificar que nuestro sistema cuenta con estas alternativas habilitadas, nuevamente disponemos de la salida de la función **phpinfo**.

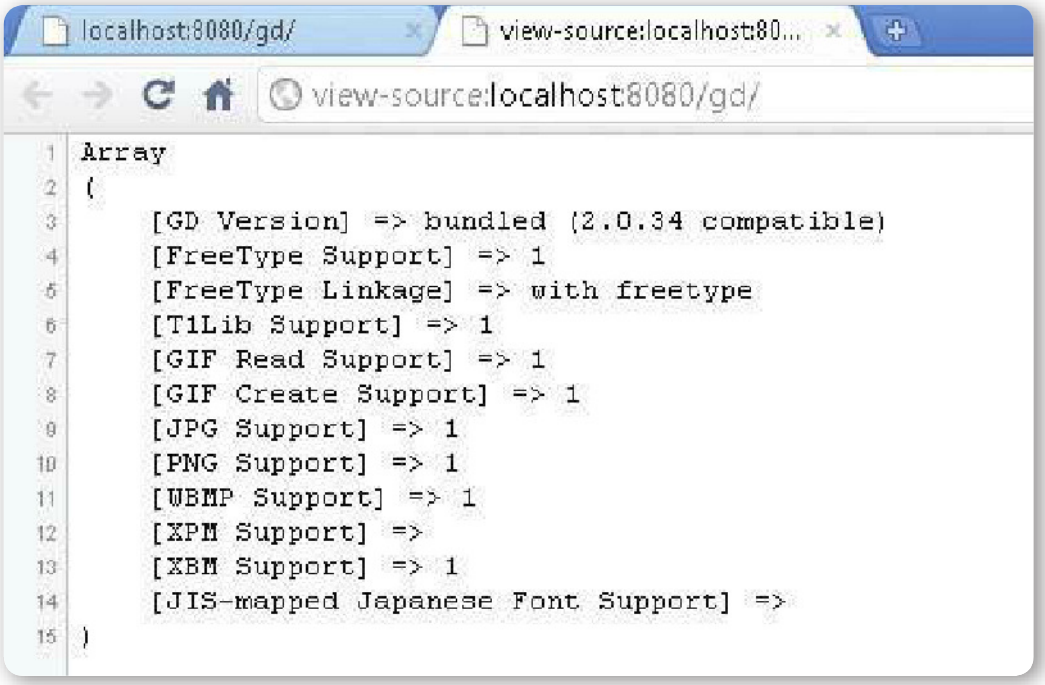


Figura 1. La función **gd_info** devuelve información acerca del soporte brindado por nuestro sistema con relación a **GD**.

GD nos permite manipular diversos formatos de imágenes. Debemos tener en cuenta que esto dependerá de la versión instalada; por ejemplo, las anteriores a la 1.6 admiten **GIF**, pero no **PNG**, y lo inverso sucede con las superiores. Algunos de los formatos más populares soportados por **GD** son los siguientes:

- **GIF** (*CompuServe Graphical Interchange Format*).
- **JPG** (o **JPEG**, *Joint Photographic Experts Group*).
- **PNG** (*Portable Network Graphics* o *PNG is Not GIF*).

En las páginas siguientes, veremos algunas de las múltiples librerías escritas en PHP que hacen uso de **GD** para manipular imágenes.

phpThumb para imágenes dinámicas

La librería **phpThumb** se utiliza para crear y manipular imágenes (**GIF**, **PNG** o **JPEG**) de manera dinámica. Aprovecha las características más avanzadas de la versión 2 de **GD**, aunque también es posible trabajar utilizando versiones anteriores. Trabaja con imágenes **GIF** aun cuando la versión de **GD** instalada en el servidor no soporte este formato (lo hace a través de la clase **GIFutil**). Además, mantiene un mecanismo para evitar el uso de la librería desde sitios externos o mostrar las imágenes almacenadas (**hotlinking**). Podemos obtener más información y descargar esta herramienta desde el sitio web oficial: **http://phpthumb.sourceforge.net**.

Lo primero que debemos hacer, luego de descargar la distribución y copiarla a un directorio del servidor, es renombrar el archivo **phpThumb.config.php.default** a **phpThumb.config.php**. La librería se utiliza a través de una llamada al archivo **phpThumb.php**, que recibe una serie de parámetros y devuelve el resultado correspondiente, es decir, la imagen generada.

DISPONIBILIDAD DE LA LIBRERÍA

Normalmente, los servicios de alojamiento web (**hosting**) dan soporte para la utilización de la librería **GD**, por lo que no deberíamos tener inconvenientes para migrar aplicaciones entre un servidor y otro. De todas maneras y para evitar futuros problemas, es conveniente asegurarnos antes de realizar la contratación.

El caso más común consiste en incluir la llamada dentro del atributo **src** del elemento **img**:

```
<imgsrc="phpthumb.php?src=nombreImagen.jpg">
```

Existe una sintaxis alternativa:

```
phpThumb.php/<parametro1>=<valor1>;<parame-
troN>=<valorN>;<w>x<h>;<imagen>
```

El último argumento es la ruta a la imagen, el penúltimo son las dimensiones (ancho y alto), y lo demás corresponde a valores asignados a parámetros:

```
<imgsrc="phpThumb.php/f=jpeg;q=70;100x150;nombreImagen.png">
```

En el ejemplo anterior, recuperamos la imagen **nombreImagen.png** y definimos que sea de 100px de ancho por 150px de alto. Además, incluimos dos parámetros adicionales: **f** (formato de la imagen devuelta) y **q** (calidad).

Entre los atributos más importantes puestos a disposición por **phpThumb**, se encuentran los siguientes:

VERSIONES DE GD

La librería **phpThumb** trabaja con las distintas versiones de **GD**, sin embargo, su comportamiento puede cambiar según contemos con las últimas disponibles (desde la 2 en adelante) o las no tan nuevas. Algunas de las variaciones pueden estar relacionadas con la calidad en la visualización de las imágenes, por ejemplo.

- **src**: indica la dirección de la imagen por tratar.
- **new**: permite crear una imagen nueva y recibe como valores un número hexadecimal (color) y, opcionalmente, un porcentaje de opacidad. Además, requiere que se definan los atributos **w** y **h**. Con el siguiente código generamos una imagen color amarillo de 200px por 100px:

```
phpthumb/phpthumb.php?new=FFFF00&w=200&h=100
```

Para definir el grado de opacidad (por ejemplo 20%):

```
phpthumb/phpthumb.php?new=FFFF00|20&w=200&h=100
```

- **w**: establece un ancho máximo en pixeles para la imagen.
- **h**: establece un alto máximo en pixeles para la imagen.
- **f**: establece un formato de salida para la imagen, que puede tomar como posibles valores a **JPEG**, **PNG** o **GIF**.
- **q**: define la calidad (nivel de compresión) de la imagen y solo se aplica al formato **JPEG** (1 para baja calidad, 95 para máxima calidad, 75 es valor por defecto).
- **sx**: quita un porcentaje de la imagen a partir del margen izquierdo (toma valores entre 0 y 1). En el siguiente ejemplo, mostramos primero la imagen original y, luego, la misma imagen con un cuarenta por ciento menos:

```
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png&sx=0.4
```

- **sy**: quita un porcentaje de la imagen a partir del margen superior (toma valores entre 0 y 1). En el siguiente ejemplo, mostramos primero la imagen original y, luego, la misma imagen con un diez por ciento menos:

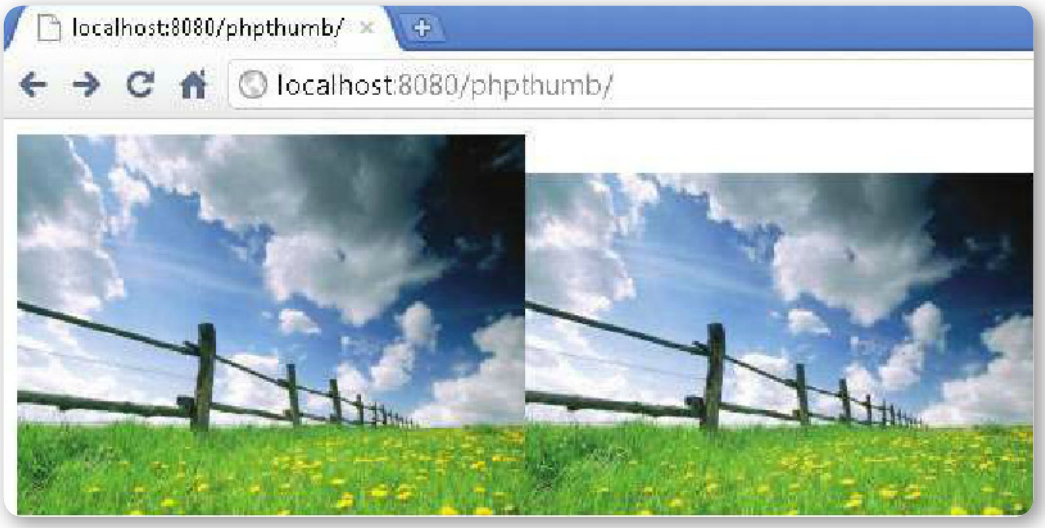


Figura 2. La manipulación de imágenes a través de **phpThumb** se realiza invocando el archivo **phpthumb.php**.

```
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png&sy=0.1
```

- **sw**: es similar a **sx** solo que muestra únicamente el porcentaje a partir del margen izquierdo. En el siguiente ejemplo, mostramos primero la imagen original y, luego, el cincuenta por ciento de la misma imagen:

```
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png&sw=0.5
```

- **sh**: es similar a **sy** solo que muestra únicamente el porcentaje a partir del margen superior. En el siguiente ejemplo, mostramos primero la imagen original y, luego, el setenta por ciento de la misma imagen:

```
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png
phpthumb/phpthumb.php?src=/phpThumb/imagen2.png&sh=0.7
```

- **zc**: devuelve una imagen con las dimensiones definidas cuyo contenido es la imagen original adecuada:

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&w=600&h=300&zc=1
```

- **bg**: permite modificar el color de fondo de una imagen (valor hexadecimal).
- **bc**: representa el borde de la imagen (valor hexadecimal).
- **ra**: rota la imagen de acuerdo con el ángulo dado (positivo, en sentido horario; negativo, en sentido antihorario):

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&ra=90
```

- **ar**: rota la imagen 90 grados:

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&w=200&ar=p
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&w=200&ar=P
```

- **iar** (*Ignore Aspect Ratio*): ignora las dimensiones originales de la imagen sin mantener su aspecto inicial.

VERSIONES DISPONIBLES

Según la versión de **GD** que tengamos instalada en nuestra computadora, las funcionalidades ofrecidas pueden variar y afectar las aplicaciones. En la mayoría de los casos, esto no sucede, debido a que los cambios entre las versiones que se encuentran disponibles, por lo menos hasta ahora, no son radicales.

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&w=420&h=120&iar=1
```

- **far**: crea una imagen con el ancho y el alto especificados, pero mantiene el aspecto original. El valor que recibe indica la alineación (**L** para la izquierda, **R** para la derecha, **T** para arriba, **B** para abajo, **C** para el centro, y las variaciones posibles, por ejemplo, **BL** para abajo a la izquierda):

```
<img src="phpthumb/phpthumb.php?src=/phpthumb/
imagen.jpg&w=200&h=300&far=C" border="1"
```

- **sia** (*Save Image As*): define el nombre que se le dará a la imagen cuando el usuario quiera guardarla. No es necesario incluir la extensión:

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&sia=nombre
```

- **maxb**: nos da la posibilidad de establecer un valor máximo en bytes para la imagen. En el siguiente ejemplo, definimos el peso máximo en 100000 bytes (alrededor de 100 kB):

```
phpthumb/phpthumb.php?src=/phpthumb/imagen.jpg&maxb=100000
```

- **down**: permite forzar la descarga de una imagen, recibe como valor el nombre predeterminado con el cual se guardará. La descarga comienza cuando accedemos a un enlace como el siguiente:

```
phpthumb.php?src=/phpthumb/imagen.jpg&f=png&down=nuevaImagen.png
```

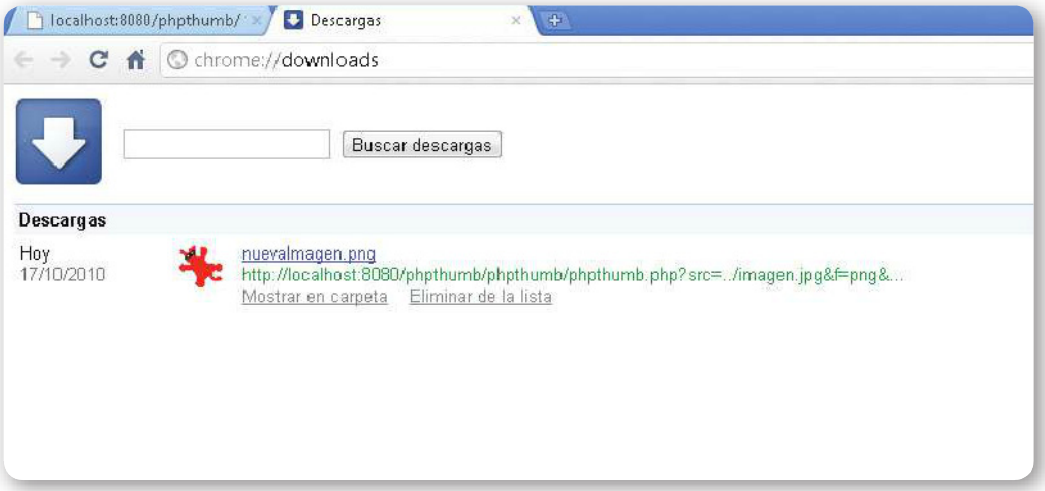


Figura 3. Es posible configurar el nombre de la imagen percibido por el usuario de la aplicación.

A través del atributo **fltr**, **phpThumb** permite la aplicación de filtros sobre las imágenes. Se utiliza de la siguiente manera:

```
phpthumb.php?src=imagen.jpg&fltr[]=opcion|valor
```

El número de valores es variable:

```
phpthumb.php?src=imagen.jpg&fltr[]=opcion|valor1|valor2|valorN
```

Y podemos incluir más de un filtro a la vez:

```
phpthumb.php?src=imagen.jpg&fltr[]=opcion|valor&fltr[]=opcionN|valorN
```

Entre los filtros disponibles se encuentran los que figuran en la tabla:

FILTROS DE IMÁGENES

▼ NOMBRE	▼ DESCRIPCIÓN
Brit	Brillo, toma valores entre -255 y 255.
Cont	Contraste, toma valores entre -255 y 255.
Gam	Corrección de la gama, toma valores positivos.
Sat	Saturación, toma valores entre 0 y –100.
Gray	Convierte a escala de grises, no recibe valores.
Sep	Sepia, toma valores entre 0 y 100.
Blur	Difuminar, toma valores entre 0 y 25.
over	Ubica una imagen por encima de otra. Recibe el nombre de la imagen y la indicación de si se superpone a la original (**0**) o viceversa (**1**).
wmi	Agrega una marca de agua a la imagen. Recibe como valores la ruta a la imagen y la alineación (**L** para izquierda, **R** para derecha, **T** para arriba, **B** para abajo, **C** para centro, y las variaciones posibles, por ejemplo, **BL** para abajo a la izquierda).
wmt	Agrega un texto sobre la imagen. Recibe como valores el texto, el tamaño de la fuente (**1** a **5**), la alineación y un color (valor hexadecimal), entre otras opciones.
Flip	Da vuelta la imagen en sentido horizontal (valor x) o vertical (valor y).
Ric	Redondea los bordes de la imagen. Recibe como valores el radio horizontal y el vertical.
bord	Asigna un borde a la imagen. Recibe como valores el ancho en pixeles, el radio horizontal y el vertical, y el color (hexadecimal).
crop	Extrae una parte de la imagen dadas las coordenadas (izquierda, derecha, arriba, y abajo; si están entre **0** y **1**, se consideran porcentajes).

Tabla 1. Algunas de las opciones disponibles para el atributo `fltr`.

Como vimos, el archivo **phpThumb.php** admite una gran cantidad de argumentos, de los cuales el más importante es **src** que sirve para indicar la ruta a la imagen (también es posible invocar archivos remotos, aunque puede ralentizar la normal ejecución del script). Además, **phpThumb** mantiene un avanzado sistema de caché, que permite reutilizar las imágenes almacenadas en la memoria para evitar procesamientos en el servidor. En el archivo **phpThumb.config.php**, se incluyen ciertas directivas para configurar el funcionamiento de la librería, entre las que se encuentra **cache_directory** (directorio de la caché):

```
$PHPTHUMB_CONFIG['cache_directory']
```

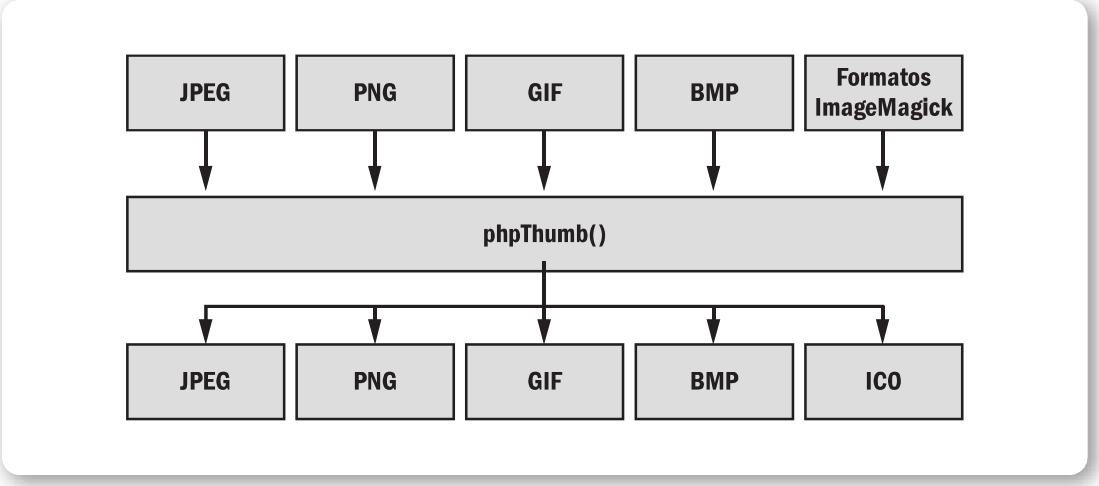


Figura 4. phpThumb permite importar y exportar archivos de imagen en diferentes formatos.

Otras opciones relativas:

- Deshabilitar los mensajes de advertencia ante posibles fallos durante la generación o administración de la caché:

```
$PHPTHUMB_CONFIG['cache_disable_warning']
```

- Número de subdirectorios admitidos para organizar y almacenar los archivos de la caché:

```
$PHPTHUMB_CONFIG['cache_directory_depth']
```

- Eliminar imágenes almacenadas en la caché que no hayan sido utilizadas en los **X** días posteriores al último acceso (**null** para nunca eliminar):

```
$PHPTHUMB_CONFIG['cache_maxage'] = 86400 * X;
```

- Tamaño máximo de la caché (en bytes):

```
$PHPTHUMB_CONFIG['cache_maxsize']
```

- Número máximo de archivos en la caché:

```
$PHPTHUMB_CONFIG['cache_maxfiles']
```

- Directorio temporal (**null** para autodetectar y utilizar el directorio temporal del sistema):

```
$PHPTHUMB_CONFIG['temp_directory']
```

- Formato de salida predeterminado (**JPEG**, **PNG** o **GIF**):

```
$PHPTHUMB_CONFIG['output_format']
```

JpGraph para la generación de gráficos

JpGraph es una librería orientada a objetos, que permite la generación de gráficos, casi siempre de tipo estadístico.

La cantidad de opciones de configuración relativas a la visualización de gráficos es inmensa y se detallan de manera clara y precisa en el manual de la aplicación, que incluye centenares de ejemplos prácticos de uso. Está disponible para todas las versiones de PHP y requiere que se habilite la extensión **GD**, cuyo objetivo es, como hemos visto, la generación de imágenes (la versión instalada en el servidor puede ir desde la 1.8 hasta la 2 y superiores).

En el momento de descargar **JpGraph** (**http://jpgraph.net/download**), deberemos optar por una de las dos distribuciones: la definida para PHP 4 o la disponible para PHP 5 y superiores.

Una vez descargada la librería, descomprimimos el archivo correspondiente y copiamos el directorio **src** (que contiene las clases) a un lugar accesible por las demás páginas. Para utilizar **JpGraph**, tenemos que incluir en nuestras páginas el archivo **jpgraph.php**:

```
include "jpgraph/jpgraph.php";
```

EXPORTACIÓN DE GRÁFICOS

Los gráficos generados a partir de **JpGraph** son, en definitiva, archivos de formatos conocidos y populares, y pueden ser incluidos en documentos **PDF** (PHP brinda librerías especiales para crear esta clase de archivos), por ejemplo, para generar reportes profesionales y personalizados por el propio desarrollador de la aplicación.

Luego, debemos agregar referencias a archivos que contienen las clases correspondientes al tipo de gráfico elegido, como veremos a continuación. Entre los tipos de gráficos soportados es posible mencionar:

- Lineales
- Barra
- Torta
- Anillo
- Mapas HTML
- Gráficos polares
- Radar
- Diagramas de Gantt
- Texto (por ejemplo, para generar imágenes CAPTCHA)
- Figuras
- Leds
- Código de barras
- Rosa de los vientos
- Velocímetros
- Puntos

> JPGRAPH ES UNA LIBRERÍA QUE PERMITE GENERAR GRÁFICOS ESTADÍSTICOS

En cuanto al licenciamiento, **JpGraph** nos ofrece una versión gratuita y otra paga, que agrega ciertas funcionalidades a la primera, entre las que podemos citar:

- Tratamiento de códigos de barra.
- Gráficos tipo velocímetro.
- Gráficos tipo rosa de los vientos.
- Generación de tablas anexas para enumerar los datos ilustrados.

Como podemos observar, las diferencias no son tantas, y la versión base debería permitirnos llevar a cabo la mayoría de los objetivos incluidos en un desarrollo profesional. Podemos obtener más información acerca de **JpGraph**,

incluyendo una gran cantidad de ejemplos de utilización, en su sitio web oficial: **http://jpgraph.net**.

El gráfico generado podrá ser **PNG**, **GIF** (siempre y cuando la versión de **GD** componga este tipo de formato) o **JPG**, por defecto en ese orden de prioridad. Si necesitamos definir otro orden, modificamos la constante **DEFAULT_GFORMAT** del archivo **jpg-config.inc.php**:

```
DEFINE("DEFAULT_GFORMAT","jpg");
```

Este documento contiene múltiples opciones de configuración, aunque los valores de manera predeterminada deberían funcionar para la mayoría de los usuarios.

Los distintos tipos de gráficos mantienen propiedades y métodos comunes, entre los que podemos citar:

PROPIEDADES Y MÉTODOS DE LOS GRÁFICOS

▼ PROPIEDAD	▼ DESCRIPCIÓN
Title	Título del gráfico.
Subtitle	Subtítulo del gráfico.
Subsubtitle	Segundo subtítulo del gráfico.
Legend	Leyenda.
▼ MÉTODO	**▼ DESCRIPCIÓN**
Add	Se utiliza para añadir un elemento al gráfico.
SetMargin	Define el margen del gráfico.
SetMarginColor	Define el color del margen del gráfico.

PROPIEDADES Y MÉTODOS DE LOS GRÁFICOS (CONTINUACIÓN)	
SetColor	Define el color del gráfico.
SetBox	Define si el gráfico estará encerrado en una caja.
SetShadow	Permite asignar una sombra al gráfico.
SetGridDepth	Agrega líneas encima del gráfico o debajo de él.
SetAngle	Define el ángulo de inclinación del gráfico.
SetBackgroundImage	Incluye una imagen de fondo al gráfico generado.
SetAxisStyle	Define el estilo de los ejes horizontales y verticales.

Tabla 2. Descripción de las propiedades y métodos disponibles de los gráficos generados.

A continuación, veremos cómo generar y personalizar algunos de los gráficos disponibles a través de **JpGraph**.

Gráficos lineales

Quizás es uno de los tipos más utilizados y de los más simples de implementar mediante **JpGraph**. Lo primero que debemos hacer es incluir los

FINES

Una de las dudas más frecuentes a la hora de analizar una aplicación web es cómo mostrarle la información generada al usuario. **JpGraph** es una valiosa opción en este contexto, ya que permite, de manera sencilla, generar gráficos intuitivos para exponer los datos almacenados.

archivos **jpgraph.php** (que sirve de base para la generación de los distintos tipos de gráficos) y **jpgraph_line.php** (específico para los lineales):

```
include 'jpgraph/jpgraph.php';
include 'jpgraph/jpgraph_line.php';
```

Luego, creamos un objeto de tipo **Graph** que actúa a modo de contenedor de los diferentes gráficos y recibe como argumentos el ancho de la imagen por generar, el alto y el formato del archivo:

```
$grafico = new Graph(450, 250, "auto");
```

Una opción que debemos definir es el tipo de escala por utilizar en los ejes: lineal (**lin**), logarítmica (**log**), textual (**text**) o entera (**int**). La única que no está disponible para ambos ejes es la textual (solo para X). Mediante el método **SetScale**, indicamos la escala uniendo las constantes; la primera es para el eje X y la segunda, para el Y:

```
$grafico->SetScale("textlin");
```

El método admite cuatro argumentos más, todos ellos opcionales: mínimo y máximo del eje Y, y mínimo y máximo del eje X.

Siguiendo con el ejemplo, generamos un objeto **LinePlot** (que recibe como argumento un array de datos) que será luego agregado al gráfico:

```
$datos = array(10, 12, 3, 14, 21, 13);
$linea = new LinePlot($datos);
$grafico->Add($linea);
```

Por último, enviamos la salida al navegador mediante el método **Stroke**:

```
$grafico->Stroke();
```

Podemos devolver el gráfico generado a un archivo (esto es válido para todas las clases de gráficos):

```
$grafico->Stroke('imagenes/grafico.png');
```

Además de las ya vistas en el ejemplo anterior, existen muchísimas otras opciones para configurar el aspecto de un gráfico. Por ejemplo, el método **SetMargin** aplicado sobre la propiedad **img** del contenedor nos permite establecer el margen del gráfico (izquierda, derecha, superior e inferior, en ese orden):

```
$grafico->img->SetMargin(45, 10, 10, 45);
```

También podemos utilizar el método **SetShadow** para darle una sombra al contenedor del gráfico. Recibe como argumentos **true** (valor predeterminado, para mostrar el sombreado), ancho (en pixeles) y color (un array que contiene los valores **RGB**). Todos son opcionales:

```
$grafico->SetShadow(true, 2, array(23,12,55));
```

A través de las propiedades **xaxis** e **yaxis**, es posible acceder a los ejes del gráfico y modificar sus características, por ejemplo:

```
$grafico->xaxis->title->Set("tituloeje x");
$grafico->yaxis->title->Set("titulo eje y");
```

Con **SetColor**, definimos el color de la línea (formato hexadecimal) y, con **SetLegend**, el título de referencia a ella:

```
$linea->SetColor("#FF0000");
$linea->SetLegend("Leyenda aqui");
```

El método **SetFormat** establece un formato para los valores, permitiendo definir la cantidad de decimales o los enteros por visualizar, por ejemplo:

```
$linea->value->SetFormat("%0.2f");
```

Para que esto tenga efecto, debemos indicar que los valores serán exhibidos, lo que es posible lograr a través del método **Show** de la propiedad **value**:

```
$linea->value->Show();
```

El método **setFont** admite tres argumentos: tipo de fuente, estilo y tamaño:

```
$grafico->title->SetFont(FF_VERDANA, FS_BOLD, 10);
```

Entre las fuentes predeterminadas se encuentran:

APERTURA

GD no es una extensión ligada a un formato específico o a alguna versión del lenguaje en particular. Por el contrario, trata de incluir más y más opciones en cada nueva versión disponible.

FUENTES PREDETERMINADAS PARA UN GRÁFICO LINEAL	
▼ CONSTANTE	▼ FUENTE
FF_ARIAL	Arial
FF_TIMES	Times Roman
FF_COURIER	Courier New
FF_VERDANA	Verdana
FF_BOOK	Bookman
FF_HANDWRT	Handwriting
FF_COMIC	Sans Comic

Tabla 3. Constantes para definir el tipo de fuente.

Y entre los estilos para modificar el comportamiento por defecto de las fuentes, tenemos los siguientes:

ESTILOS DE FUENTE PARA UN GRÁFICO LINEAL	
▼ CONSTANTE	▼ DESCRIPCIÓN
FS_NORMAL	Normal
FS_BOLD	Negrita
FS_ITALIC	Itálica
FS_BOLDITALIC	Itálica y negrita

Tabla 4. Constante para definir el estilo de la fuente.

Es posible incluir más constantes para personalizar la fuente incluida en los gráficos. Podemos obtener información acerca de cómo instalar juegos de caracteres adicionales, en el manual oficial y en la dirección **http://jpgraph.net/doc**.

El método **SetType** nos permite personalizar los puntos remarcados en las líneas del gráfico:

```
$linea->mark->SetType(MARK_UTRIANGLE);
```

Algunas de las constantes definidas para personalizar los puntos remarcados en las líneas de gráficos son las siguientes:

PERSONALIZACIÓN DE PUNTOS REMARCADOS

CONSTANTE	DESCRIPCIÓN
MARK_SQUARE	Cuadrado.
MARK_UTRIANGLE	Triángulo con la punta hacia arriba.
MARK_DTRIANGLE	Triángulo con la punta hacia abajo.
MARK_DIAMOND	Rombo.
MARK_CIRCLE	Círculo vacío.
MARK_FILLEDCIRCLE	Círculo relleno.
MARK_CROSS	Cruz; signo suma.
MARK_STAR	Asterisco.
MARK_X	Cruz; signo multiplicación.

PERSONALIZACIÓN DE PUNTOS REMARCADOS (CONTINUACIÓN)	
MARK_LEFTTRIANGLE	Triángulo con la punta hacia la izquierda.
MARK_RIGHTTRIANGLE	Triángulo con la punta hacia la derecha.
MARK_FLASH	Rayo.

Tabla 5. Constante para definir el estilo de los puntos remarcados.

Es posible incluir más de una línea en un mismo gráfico:

```
<?php

include 'jpgraph/jpgraph.php';
include 'jpgraph/jpgraph_line.php';

$datosLinea1 = array(2, 4, 5, 7.2, 9, 1);
$datosLinea2 = array(2, 3.2, 4, 5, 2, 1);

$grafico = new Graph(500, 300, "auto");
$grafico->SetScale("textlin");

$linea1 = new LinePlot($datosLinea1);
$linea1->SetColor("#66CC00");
$linea1->value->Show();
$linea1->value->SetFormat("%0.1f");
$linea1->SetLegend("Linea 1");
$grafico->Add($linea1);

$linea2 = new LinePlot($datosLinea2);
$linea2->SetColor("#FF6666");
$linea2->value->Show();
```

```
$linea2->value->SetFormat("%0.1f");
$linea2->SetLegend("Linea 2");
$grafico->Add($linea2);

//leyenda en sentido horizontal
$grafico->legend->SetLayout(LEGEND_HOR);

$grafico->Stroke('imagen.png');

?>
```

Los gráficos de líneas permiten representar información de manera sencilla y personalizable, son una opción válida a la hora de implementar maneras alternativas de presentación de datos en aplicaciones web.

Gráficos de barra

Otro de los tipos de gráficos comúnmente utilizados para representar situaciones en aplicaciones web es el de barras. Para implementarlos mediante **JpGraph**, contamos con la clase **BarPlot**, que recibe como argumento un array de datos, al igual que **LinePlot**:

```
$barra = new BarPlot($datos);
```

Lo primero que debemos hacer antes de instanciar la clase será incluir los archivos correspondientes para trabajar barras (**jpgraph.php** y **jpgraph_bar.php**):

```
include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_bar.php");
```

Luego, generamos un contenedor a través de la clase **Graph**:

```
$grafico = new Graph(300, 200, 'auto');
```

Y definimos la escala, al igual que en los ejemplos anteriores, con el método **SetScale**:

```
$grafico->SetScale("textlin");
```

Asignamos valores a algunas de las propiedades generales admitidas por la mayoría de los gráficos, en este caso título, leyenda del eje X y leyenda del eje Y:

```
$grafico->title->Set("Titulo del grafico");
$grafico->yaxis->title->Set("Y");
$grafico->xaxis->title->Set("X");
```

Definimos la fuente de datos y la pasamos como argumento al constructor de la clase **BarPlot**:

```
$datos = array(24, 12, 11, 15, 3, 14);
$barra = new BarPlot($datos);
```

Los objetos de esta clase tienen sus propios métodos disponibles, entre los que podemos citar los siguientes:

- **SetLegend**: define la leyenda o el título para especificar de manera textual qué es lo que se está midiendo.

```
$barra->SetLegend("Valores");
```

- **SetFillColor**: nos permite asignar un color a las barras del gráfico.

```
$barra->SetFillColor("#FFCC33");
```

- **SetAbsWidth**: recibe como argumento el número de pixeles de ancho de cada una de las columnas.

```
$barra->SetAbsWidth(30);
```

- Opcionalmente, podemos utilizar **SetWidth**, que define el ancho con relación al espacio disponible (por ejemplo 90%):

```
$barra->SetWidth("0.9");
```

- El método **Show** aplicado a la propiedad **value** indica que los valores de cada columna serán exhibidos (este no es el comportamiento predeterminado).

REPORTES

PHP no cuenta con un generador de reportes verdaderamente popular y afianzado, sin embargo, mantiene una serie de herramientas que se pueden utilizar en conjunto para generarlos. Una de ellas puede ser, sin dudas, **JpGraph** y otra, **FPDF**, que permite la generación de documentos en formato **PDF**.

```
$barra->value->Show();
```

- En el mismo sentido, podemos establecer el color de la fuente:

```
$barra->value->SetColor('#000000');
```

- Definimos la alineación vertical a través del método **SetValuePos**, cuyos posibles valores son **top**, **middle** y **bottom**.

```
$barra->SetValuePos('bottom');
```

- Por último, añadimos la barra al contenedor y enviamos el gráfico al navegador para poder visualizarlo:

```
$grafico->Add($barra);
$grafico->Stroke();
```

- En lugar de **SetFillColor**, podemos agregar contraste al color de las barras mediante **SetFillGradient**, que recibe como argumentos los valores hexadecimales de los colores componentes (o un nombre reconocido por **JpGraph**) y un estilo definido mediante alguna de las constantes disponibles:

- **GRAD_CENTER**
- **GRAD_HOR**
- **GRAD_LEFT_REFLECTION**
- **GRAD_MIDHOR**
- **GRAD_MIDVER**
- **GRAD_RAISED_PANEL**
- **GRAD_RIGHT_REFLECTION**

- **GRAD_VER**
- **GRAD_WIDE_MIDHOR**
- **GRAD_WIDE_MIDVER**

Por ejemplo:

```
$barra->SetFillGradient("#FFFF00", "#993333", GRAD_VER);
```

Este tipo de gráfico permite variantes, como la inclusión de múltiples barras una sobre otra o alineadas a través de las clases **GroupBarPlot** y **AccBarPlot**.

Gráficos de torta

Los gráficos de este tipo se generan incluyendo los archivos **jpgraph_pie.php** y **jpgraph_pie3d.php** y, en lugar de la clase **Graph**, se toma como base a **PieGraph**:

- **index.html**

```
<html>
<head>
<title>graficos con jpgraph</title>
</head>

<body><imgsrc="grafico.php" border="0" /></body>
</html>
```

- **grafico.php**

```
<?php

include "jpgraph/jpgraph.php";
```

```
include "jpgraph/jpgraph_pie.php";
include "jpgraph/jpgraph_pie3d.php";

$datos[] = 10;
$datos[] = 30;
$datos[] = 32;
$datos[] = 15;

$leyendas[] = "2007";
$leyendas[] = "2008";
$leyendas[] = "2009";
$leyendas[] = "2010";

$grafico = new PieGraph(350, 240, "auto");
$grafico->SetMarginColor('#CCCC99');
$grafico->SetShadow();
$grafico->title->Set("Titulo del grafico");

$torta = new PiePlot3D($datos);
$torta->SetSize(0.5);
$torta->SetCenter(0.40);
$torta->SetLabelPos(0.45);
$torta->SetLegends($leyendas);

$torta->value->SetFont(FF_FONT2, FS_NORMAL);
$torta->value->SetColor("#FFFFFF");

$grafico->Add($torta);
$grafico->Stroke();

?>
```

El constructor **PiePlot3D** recibe como argumento un array que deberá contener los datos por graficar (no porcentajes, sino valores).

El método **SetSize** especifica el tamaño de la torta y puede tomar valores entre **0** y **0.5**:

```
$torta->SetSize(0.5);
```

Por su parte, **SetCenter** define la posición con relación al contenedor (0.5 para ubicarla en el centro):

```
$torta->SetCenter(0.40);
```

El método **SetLabelPos** establece la distancia entre cada título de cada porción y el centro:

```
$torta->SetLabelPos(0.45);
```

Otra opción disponible es **ExplodeAll**, que nos permite separar las porciones de la torta agregando un margen intermedio:

REDIRECCIÓN DE LA SALIDA

El comportamiento predeterminado de la mayoría de las aplicaciones que generan imágenes a partir de otras ya existentes consiste en enviar la salida directamente al navegador, por lo que debemos tener la precaución de no imprimir ninguna información antes, para no interferir el procesamiento por parte del cliente.

```
$torta->ExplodeAll();
```

Al generar el gráfico, los valores pasados como argumento a **PiePlot3D** se convierten en porcentajes.

Anillos

Estos gráficos son una variación de los de torta y se implementan de manera similar. En primer lugar, debemos utilizar la clase **PiePlotC** en lugar de **PiePlot3D**:

```
<?php

include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_pie.php";
include "jpgraph/jpgraph_pie3d.php";

$datos[] = 10;
$datos[] = 30;
$datos[] = 32;
$datos[] = 15;

$leyendas[] = "2007";
$leyendas[] = "2008";
$leyendas[] = "2009";
$leyendas[] = "2010";

$grafico = new PieGraph(350, 240, "auto");
$grafico->SetMarginColor('#CCCC99');
$grafico->SetShadow();
$grafico->title->Set("Titulo del grafico");

$anillo = new PiePlotC($datos);
```

```
$anillo->SetSize(0.4);
$anillo->SetCenter(0.40);
$anillo->SetLabelPos(0.75);
$anillo->SetLegends($leyendas);

$anillo->value->SetFont(FF_FONT2, FS_NORMAL);
$anillo->value->SetColor("#FFFFFF");

$grafico->Add($anillo);
$grafico->Stroke();

?>
```

Un método adicional de utilidad es **SetMidColor**, que permite definir el color de fondo del círculo central del anillo:

```
$anillo->SetMidColor('#CCCC99');
```

Si este valor es igual al de **SetMarginColor**, el fondo del gráfico quedará establecido con un color uniforme.

Leds

En este contexto, un led podría definirse como una serie de puntos iluminados que contrastan con los demás, formando caracteres. Fueron incluidos recientemente en las últimas versiones de la librería. Los signos soportados para incluir en esta clase de gráficos son:

- Numéricos (del 0 al 9).
- Puntos (.).
- Espacios ( ).

- Numerales (#).
- Letras mayúsculas (de la A a la L).

Inicialmente, debemos incluir el archivo **jpgraph_led.php** y generar un objeto **DigitalLED74** (que recibe como argumentos opcionales el radio, de manera predeterminada es **2**, y el margen, por defecto es **0.6**):

```
$led = new DigitalLED74();
```

El método **Stroke**, en este caso, recibe como argumento el texto por generar:

```
$led->Stroke(rand(1000000000, 900000000));
```

Además, podemos incorporar como segundo argumento una constante que representa el color del texto generado:

- **LEDC_RED** (rojo)
- **LEDC_GREEN** (verde)
- **LEDC_BLUE** (azul)
- **LEDC_YELLOW** (amarillo)
- **LEDC_GRAY** (gris)

MÉRITOS PROPIOS

JpGraph es un claro ejemplo de una aplicación que utiliza librerías externas para funcionar (**GD**), pero que, sin embargo, se ha sabido ganar un nombre propio y un prestigio en relación con los desarrolladores de aplicaciones que necesitan incorporar gráficos estadísticos en sus programas.

```
$led->Stroke('DIA #1', LEDC_GREEN);
```

Figura 5. El soporte para leds se incluyó recién en las últimas versiones de **JpGraph** y está en pleno proceso de desarrollo.

CAPTCHA

Las imágenes **CAPTCHA** (*Completely Automated Public Turing test totell Computers and Humans Apart*, prueba pública y automática para diferenciar máquinas y humanos) se utilizan para comprobar si quien está manejando una determina computadora es un ser humano o un robot.
Muestran una cadena de caracteres que el usuario debe ingresar en una caja de texto para acceder a cierto servicio de una página web (descarga de un archivo, etcétera). Si es un ser humano, podrá hacerlo con éxito.

JpGraph incluye una implementación sencilla y potente para, en pocas líneas de código, generar aplicaciones funcionales. El archivo que debemos incluir en este caso es **jpgraph_antispam.php**:

```
include "jpgraph/jpgraph_antispam.php";
```

Y luego inicializar un objeto **AntiSpam**:

```
$spam = new AntiSpam();
```

La cadena de caracteres contenida en la imagen puede ser definida de dos maneras: al azar (mediante el método **Rand** que recibe como argumento la longitud del texto) o de forma predeterminada (a través del método **Set** que recibe como argumento el string):

```
$spam->Rand($longitud);

$spam->Set($cadena);
```

En cualquier caso y para comparar lo que ingresa el usuario con lo que dice la imagen, será necesario guardar el contenido (en una variable de sesión, por ejemplo):

```
session_start();
$_SESSION['cadena'] = $spam->Rand($longitud);

session_start();
$_SESSION['cadena'] = $spam->Set($cadena);
```

Por último, imprimimos la imagen generada:

ORIGEN DE LOS DATOS

Los datos graficados a partir de la librería **JpGraph** pueden ser de origen variado. La única condición que requieren es poder representarse a través de arrays (matrices) PHP, algo que por supuesto de ningún modo es un inconveniente y que podemos generar dentro de la aplicación antes de invocar a la librería.

```
$spam->Stroke();
```

Suponiendo que lo anterior se almacenara dentro de un archivo llamado **grafico.php**, en el formulario de ingreso compararíamos el texto enviado por el usuario con lo almacenado en la variable **cadena**:

```
<?php

session_start();

if (count($_POST)) {
if ($_SESSION['cadena'] == $_POST['texto']) {
die ('<li>Ingreso correcto');
 } else {
echo '<li>Error en ingreso';
 }
}

echo '<br /><imgsrc="grafico.php">';

echo '<form method="post">';
echo '<input type="text" name="texto" id="texto">';
echo '<input type="submit">';
echo '</form>';

?>
```

OPCIONES DISPONIBLES

Es enorme la cantidad de opciones puestas a disposición del desarrollador para modificar la apariencia o el comportamiento de un gráfico generado con **JpGraph**.

Figura 6.
El soporte para generar imágenes CAPTCHA propuesto por **JpGraph** es sencillo de implementar y a la vez muy potente.

Mapas HTML

Un mapa HTML sirve para ligar partes de una imagen a diferentes acciones (enlaces a distintas páginas o archivos, por ejemplo). En **JpGraph**, es posible crear este tipo de elementos que se diferencian del resto de los gráficos, principalmente, porque se genera no solo el archivo correspondiente a la imagen, sino un fragmento HTML que deberá ser incluido. Es posible incluir más de un mapa en una misma página. Suponiendo que lo generemos desde **mapa.php**, se invocaría de la siguiente manera:

- **index.php**

```
<html>
<head>
<title>jpgraph - mapa</title>
</head>
<body><?php include 'mapa.php'; ?></body>
</html>
```

El contenido de **mapa.php** no difiere en cuanto a lo que sería habitual en la creación de gráficos: las únicas diferencias son la utilización de los métodos **SetCSIMTargets** y **StrokeCSIM**. Cada elemento del mapa necesita dos valores (un enlace válido y un título) que se asignan mediante el método

SetCSIMTargets. En lugar de **Stroke**, utilizamos el método **StrokeCSIM**, que debe recibir el nombre del archivo actual:

```
<?php

include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_bar.php");

$grafico = new Graph(500, 200, 'auto');

for ($c=0;$c<12;$c++) {
$enlaces[$c] = '/detalle.php?mes='.($c+1);
  $titulos[$c] = 'Mes numero '.($c+1);
  $datos[$c] = rand(0, 100);
}

$grafico->SetScale("textlin");

$grafico->title->Set("Titulo del grafico");
$grafico->yaxis->title->Set("Y");
$grafico->xaxis->title->Set("X");

$barra = new BarPlot($datos);

$barra->SetLegend("Valores");

$barra->SetFillGradient('yellow', 'white', GRAD_VER);

$barra->SetWidth("0.9");

$barra->value->Show();

$barra->value->SetFormat("%0.0f");

$barra->value->SetColor('#000000');
```

```
$barra->SetValuePos('bottom');

$barra->SetCSIMTargets($enlaces, $titulos);

$grafico->Add($barra);

$grafico->StrokeCSIM('mapa.php');

?>
```

El código fuente de **index.html** contiene el mapa HTML incrustado en el contenido. El gráfico se genera utilizando valores aleatorios, por lo cual la salida será diferente en cada ejecución.

Utilización de gráficos en la caché

En ocasiones, los gráficos generados por un script no varían de manera dinámica, por lo que invocar su creación a través de las clases y los métodos disponibles a partir de **JpGraph** consume recursos innecesariamente. Para administrar el uso de la caché de gráficos, existen las constantes **USE_CACHE** (habilitar la memoria caché), **READ_CACHE** (utilizar los gráficos que están en la caché) y **CACHE_DIR** (ruta absoluta a un directorio temporal, que debe contar con permisos de escritura), todas disponibles en el archivo **jpg-config.php**:

```
DEFINE("USE_CACHE", true);
DEFINE("READ_CACHE", true);
DEFINE("CACHE_DIR", "/imagenes/cache/");
```

Si la caché está habilitada, podremos utilizarla incluyendo dos argumentos en el constructor del objeto correspondiente al contenedor:

```
$grafico = new Graph($ancho, $alto, $nombreImagenCache, $tiempoExpiracion);
```

El tercer argumento es el nombre de la imagen en el directorio temporal (por ejemplo, **grafico.png**), y el cuarto es el tiempo en minutos antes de actualizar la imagen (**0** para no actualizar nunca). Si la variable **nombreImagenCache** es igual a **auto**, se utilizará como nombre de imagen el nombre del script más la extensión que corresponda. Por ejemplo, si el gráfico se genera desde la página **grafico.php**, el archivo almacenado se llamará **grafico.png**, si es que utilizamos **PNG** como formato. Si la imagen no se encuentra disponible en la caché, se ejecuta el script correspondiente para generarla.

En el caso de los mapas, deberemos invocar el método **CheckCSIMCache**, que recibe como argumentos **nombreImagenCache** y **tiempoExpiracion**:

```
$grafico = new Graph($ancho, $alto);
$grafico->CheckCSIMCache($nombreImagenCache, $tiempoExpiracion);

//código

$grafico->StrokeCSIM();
```

EL FORMATO PNG

PNG (**Portable Network Graphics** o **PNG is Not GIF**) es un formato cada vez más popular, impulsado por la comunidad de código abierto como alternativa a **GIF**. Admite la posibilidad de crear imágenes con transparencia, algo que, en su momento, caracterizaba únicamente a **GIF**.

En cuanto a las constantes, contamos con **CSIMCACHE_DIR**, similar a **CACHE_DIR**, pero exclusivo para mapas:

```
DEFINE("CSIMCACHE_DIR", "/imagenes/cache/mapas/");
```

Recordemos siempre que la ruta deberá ser absoluta y que el directorio tendrá que contar con permisos de lectura y escritura.

Las opciones brindadas por **JpGraph** son casi infinitas y, en cada nueva versión, se agregan más y más características que perfeccionan lo que ya se ofrecía o dan nuevas alternativas a la hora de generar gráficos. Se trata, sin dudas, de una herramienta de nivel profesional.

RESUMEN

En este apéndice, analizamos en detalle algunas de las principales aplicaciones que hacen uso de la librería **GD** para generar y manipular diferentes tipos de imágenes (como por ejemplo: gráficos, mapas HTML, imágenes CAPTCHA, entre otras) desde PHP: **phpThumb**, **JpGraphe Image_Graph**.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Por qué considera que **GD** es tan popular?
2. ¿Utiliza directamente la extensión **GD** en sus desarrollos? ¿Por qué?
3. ¿Por qué algunas versiones de **GD** no soportan el formato GIF?
4. ¿En qué casos utilizaría la caché para almacenar imágenes?
5. ¿Cuál es el estado actual de la extensión **GD**?

EJERCICIOS PRÁCTICOS

1. Genere un álbum fotográfico utilizando **phpThumb**.
2. Desarrolle un buscador de imágenes con **phpThumb**.
3. Implemente una encuesta en línea y grafique los resultados parciales y finales con **JpGraph**.
4. Modifique la aplicación anterior utilizando **Image_Graph** en lugar de **JpGraph**.

PROFESOR EN LÍNEA

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

PHP desde cero

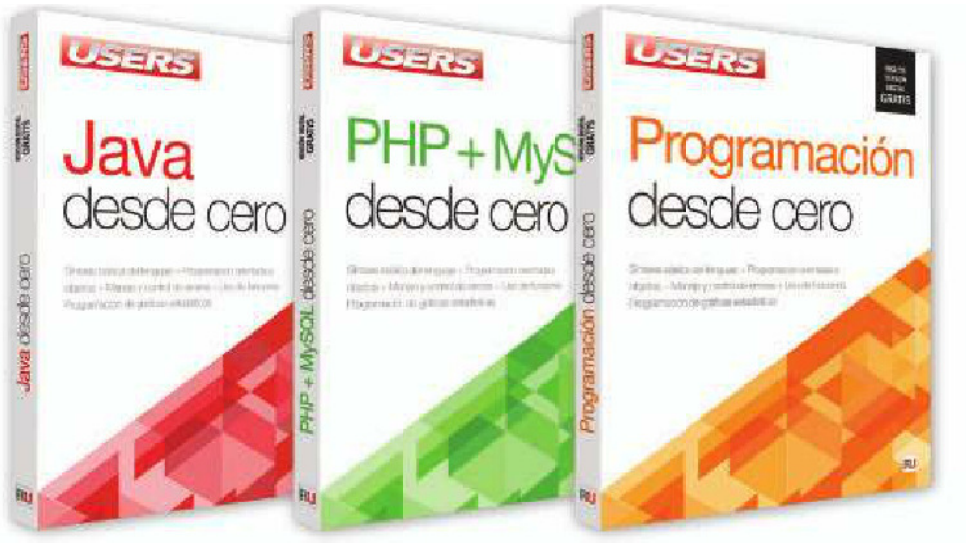
Presentamos un libro para iniciarnos en el mundo de la programación web por medio de un lenguaje flexible y de fácil aprendizaje como PHP. A lo largo de los capítulos desarrollaremos la teoría necesaria y, mediante ejemplos sencillos, comenzaremos a programar sitios web profesionales. La obra incluye ejercicios paso a paso y material extra para descarga, disponible en el sitio web de la editorial.

Dentro del libro encontrará*:

Ámbito de desarrollo y sintaxis básica. / Expresiones, comentarios y funciones. / Instrucciones y manejo de variables. / Opciones para visualizar y solucionar fallas. / Generación de gráficos.

* Parte del contenido fue publicado en *Curso de programación PHP*, *PHP 5* y *PHP Avanzado*, de esta editorial.

Otros títulos de la colección:



ISBN 978-987-1949-58-8