Esta comunicación entre desarrolladores y usuarios del lenguaje hace que el vínculo entre la comunidad de PHP se fortalezca cada día, generando un compromiso que permite la evolución de éste.
Como veremos, en una aplicación web interactúan múltiples actores, y en este libro intentaremos ubicar cada uno de ellos determinando de manera clara y precisa la función que abarcan, las responsabilidades que poseen, y cómo se relacionan entre sí.
Trasmitir un lenguaje en su total extensión es algo poco menos que imposible, pero en lo que sigue trataremos de incluir y explicar los puntos determinantes que constituyen este verdadero suceso, al alcance de quien quiera incursionar en él. En síntesis, lo que buscamos es dar a conocer las características de PHP, los puntos fuertes que lo distinguen y lo posicionan de manera cierta como una de las alternativas más viables en la actualidad en cuanto al desarrollo de aplicaciones web.

PHP 6

Capítulo 1

Introducción al lenguaje

Comenzaremos este capítulo haciendo un recorrido por los principales puntos y características que hacen de PHP un lenguaje popular y fuerte a la hora de emprender el desarrollo de aplicaciones web con calidad profesional.

Aplicaciones web 14
- Arquitectura cliente-servidor 15
- Sitios dinámicos 17
- El lenguaje PHP 18
- Tecnologías del lado servidor 22

Cómo desarrollar sitios en nuestra máquina local 24
- Instalación 24
- Primer ejemplo 28
- Inclusión de código PHP en documentos HTML 32
- Extensiones del lenguaje 35

Resumen 37

Actividades 38

APLICACIONES WEB

En la actualidad, podríamos llegar a diferenciar los distintos tipos de aplicaciones en dos grandes grupos: las de escritorio y las web. Aunque este límite sea cada día más difuso (una aplicación de escritorio puede llegar a tener una interfaz web y, a través de un mismo lenguaje, se puede desarrollar una aplicación y luego definir si va a ser accesible por medio de de un navegador o si se va a instalar en el equipo personal del usuario) servirá para que podamos comprender cuales son los alcances de PHP como lenguaje.

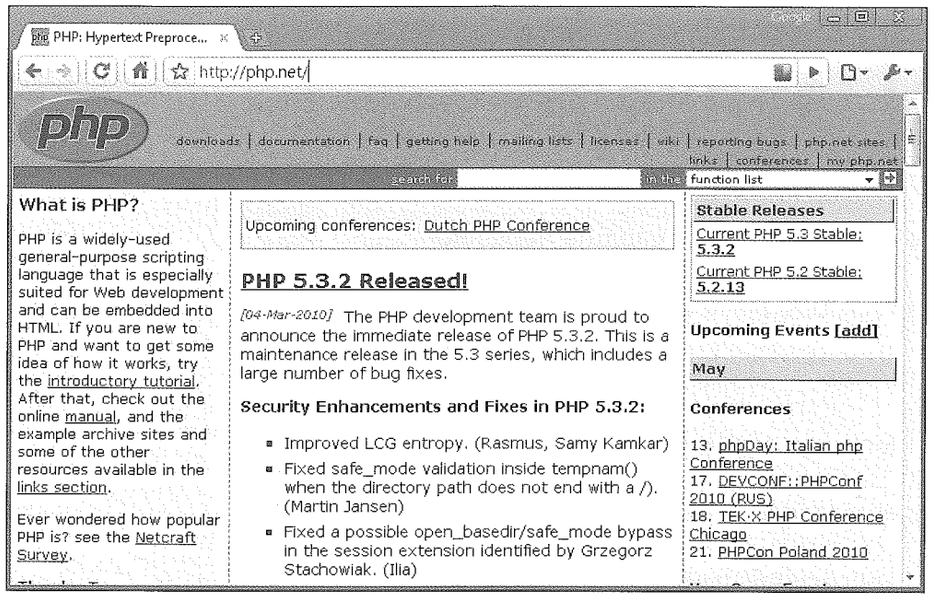

***Figura 1.** PHP es uno de los lenguajes multiplataforma más robustos de la actualidad. Cuenta, además, con una gran cantidad de adeptos.*

Las aplicaciones web son aquellas accesibles, en general, a través de un navegador web. El usuario ingresa la dirección de ubicación, conocida como URL, y comienza a interactuar con ella, tal cual como si se tratara de una aplicación de escritorio.

MANUAL OFICIAL DEL LENGUAJE

El manual oficial de PHP es una muy completa fuente de recursos que nos permitirá obtener respuestas inmediatas a muchas de nuestras preguntas. Podemos acceder a su versión en línea en **www.php.net/manual**, desde donde también podremos descargar versiones en distintos formatos.

A excepción de algunas tecnologías que necesitan determinadas características con relación al cliente, no habrá requerimientos específicos con él (para ejecutar ciertas aplicaciones, por ejemplo las desarrolladas en Flash, deberemos contar con el plug-in correspondiente). Esto es así por el paradigma sobre el que se basan las aplicaciones web: la arquitectura cliente-servidor.

***Figura 2.** Los navegadores son la puerta de entrada habitual para interactuar con aplicaciones web.*

Arquitectura cliente-servidor

Este concepto manejado en muchos tipos de aplicaciones y particularmente en las de interfaz web, podría ser definido como un juego de peticiones y respuestas. Un cliente requiere determinada acción (por ejemplo, a través de un enlace) y el servidor deberá, por medio de un procesamiento, resolver la demanda y devolver una respuesta. En general, podríamos decir que la aplicación cliente por excelencia es el navegador web: desde esta clase de programas accedemos a la interfaz del sistema para interactuar con él aunque existen otros tipos de aplicaciones cliente.

En el lado del servidor pueden darse una serie de alternativas que desembocarán, finalmente, en construir una respuesta que sea claramente comprensible para el cliente. Entre estas alternativas, podemos incluir el tratamiento de esta respuesta a través de un lenguaje de programación, por ejemplo, la extracción de información desde una base de datos, entre muchas otras posibles.

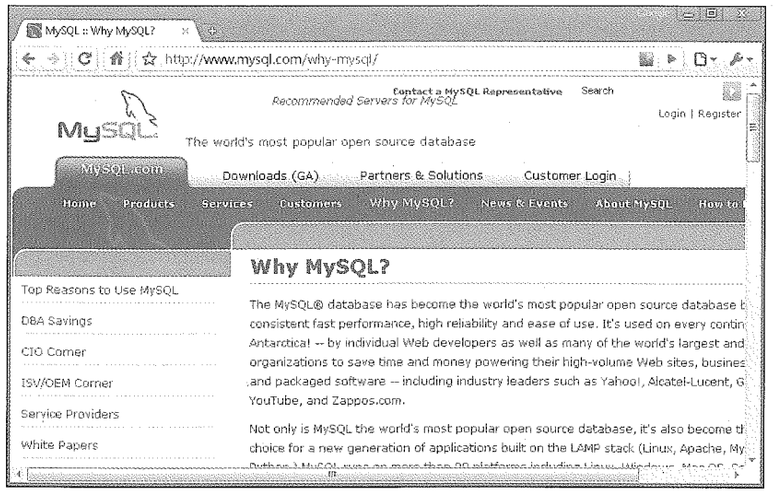

***Figura 3.** Las aplicaciones web pueden recuperar información desde distintas fuentes, entre ellas, las bases de datos.*

Podríamos definir las peticiones desde un navegador como peticiones web, y éstas son resueltas por los llamados servidores web. Las aplicaciones (**Apache** e **IIS** son de 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 anteriormente: servidores de bases de datos (**Oracle**, **SQL Server**, **MySQL**, **PostgreSQL**, por ejemplo) y lenguajes de programación (**PHP**, **ASP.net**, **JSP**, o **PERL**, entre otros).

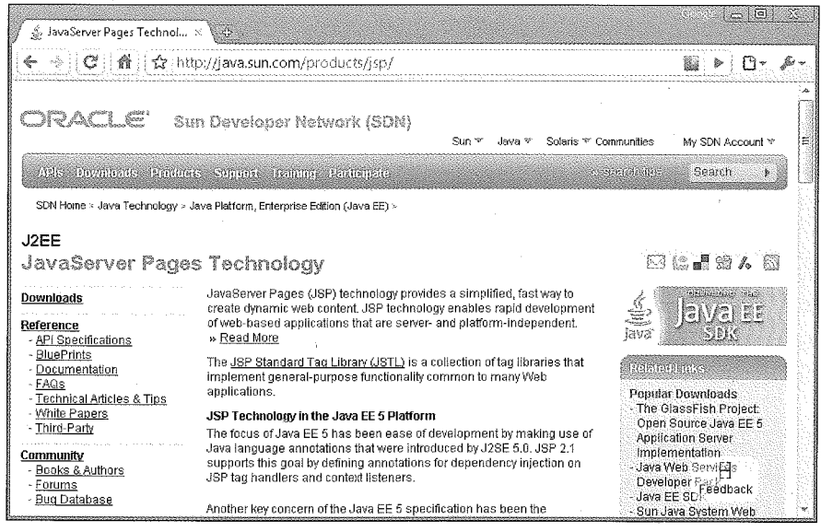

***Figura 4.** Los lenguajes de programación permiten generar aplicaciones flexibles y otorgan mayor poder de control a los desarrolladores de sitios.*

Sitios dinámicos

Ya dentro de las aplicaciones web, otra distinción posible podría ser aquella que está dada entre lo que serían sitios dinámicos y sitios estáticos.

Un lenguaje de programación como alguno de los citados anteriormente, nos daría 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. Con la utilización de lenguajes de programación y con la obtención de la información particular de cada ítem desde una fuente determinada (por ejemplo, una base de datos), sólo necesitaríamos contar con un archivo cuyo contenido dinámico (nombre del producto, foto, descripción, etcétera) sería modificado tomando como referencia la petición del usuario:

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

Sitios estáticos vs. sitios dinámicos

Un sitio estático es aquel que no utiliza lenguajes dinámicos y devuelve la misma respuesta siempre, más allá del tipo de petición. Un sitio dinámico es aquel que modifica su comportamiento sobre la base de los ingresos del usuario.

Esta es una de las ventajas principales relativas a la utilización de herramientas como lenguajes de programación del lado servidor, más bases de datos o cualquier fuente de información externa (documentos XML, servicios web, archivos, etcétera): 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.

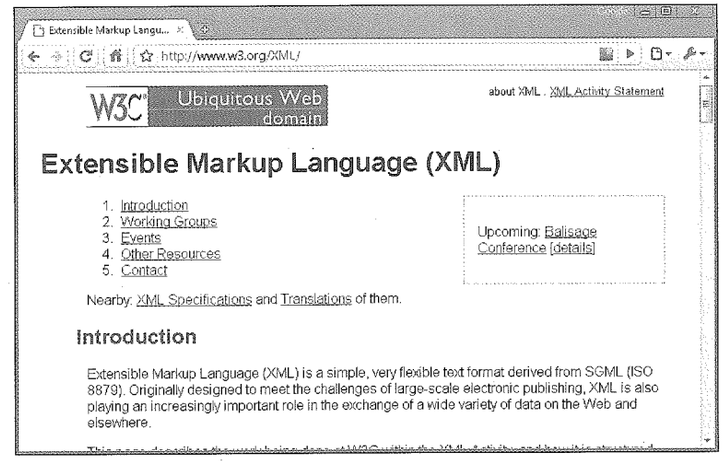

***Figura 5.** Las bases de datos tradicionales no son la única fuente de información: XML es una alternativa distinta, pero válida.*

El lenguaje PHP

PHP (*PHP Hypertext Preprocessor*) es uno de los lenguajes de programación más utilizados en la actualidad (se utiliza mayormente para desarrollo de sitios web pero para muchos es ya un lenguaje de propósito general). Esto se debe a múltiples factores, entre los cuales podemos citar los siguientes:

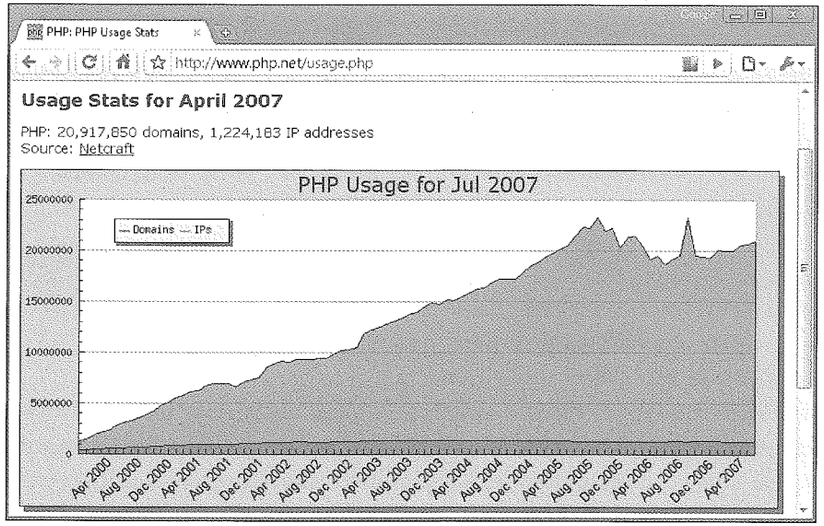

***Figura 6.** El grado de utilización de PHP en aplicaciones web lo posiciona como uno de los lenguajes del momento.*

- Es libre y gratuito. PHP, al igual que muchos otros lenguajes y entornos de programación, está amparado bajo el movimiento **open source** (código abierto), que permite a los programadores de aplicaciones poder sacar provecho de sus beneficios de manera totalmente gratuita, sin la necesidad de pagar licenciamientos de uso ni actualizaciones. La licencia consta principalmente de tres puntos: libertad para utilizar el programa (PHP), posibilidad de modificar el programa si se accede a su código fuente, distribuir el programa modificado o no. La licencia de PHP está disponible en **www.php.net/license**.

APLICACIONES DE ESCRITORIO

Uno de los proyectos mantenidos por los mismos responsables del lenguaje es PHPGTK, que permite generar aplicaciones de escritorio a través del intérprete PHP. Su uso no está del todo extendido, pero el proyecto sigue en pie y evoluciona de manera notable en cada nueva versión.

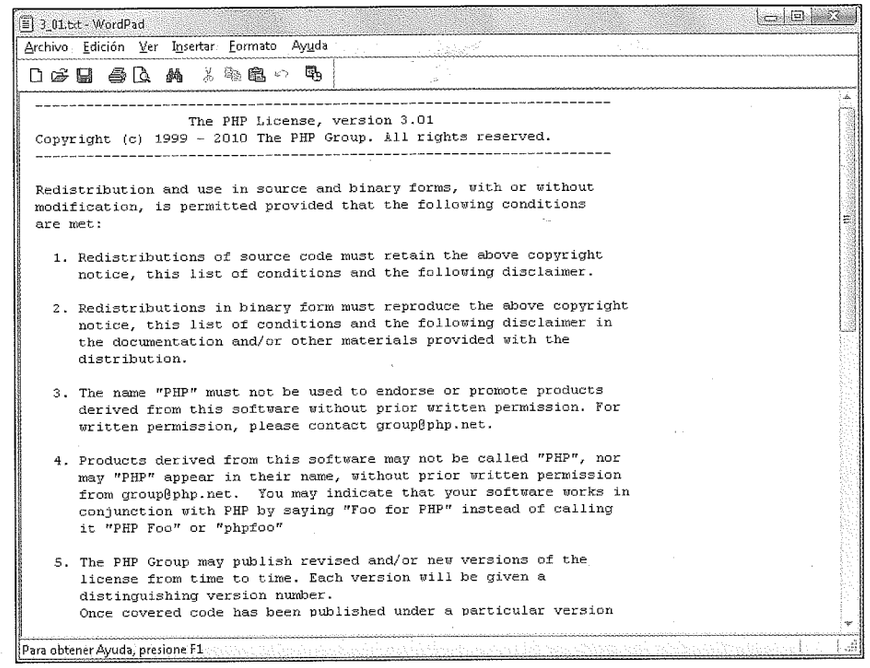

***Figura 7.** El modo licenciamiento de PHP nos permite sacar provecho de sus funcionalidades sin tener que pagar dinero por su uso o actualizaciones.*

- Disponibilidad. Evidentemente, PHP no es la única alternativa a la hora de desarrollar aplicaciones web, pero sin duda es la más popular. Al momento de contratar un servicio de alojamiento, con seguridad, 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 PHP está instalado en nuestro servidor y listo para ser utilizado. PHP se encuentra disponible para los siguientes sistemas operativos:

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

Hay que aclarar que PHP funciona en todas las versiones de Microsoft Windows, excepto PHP version 5 y superiores, ya que no son soportadas por Windows 95.

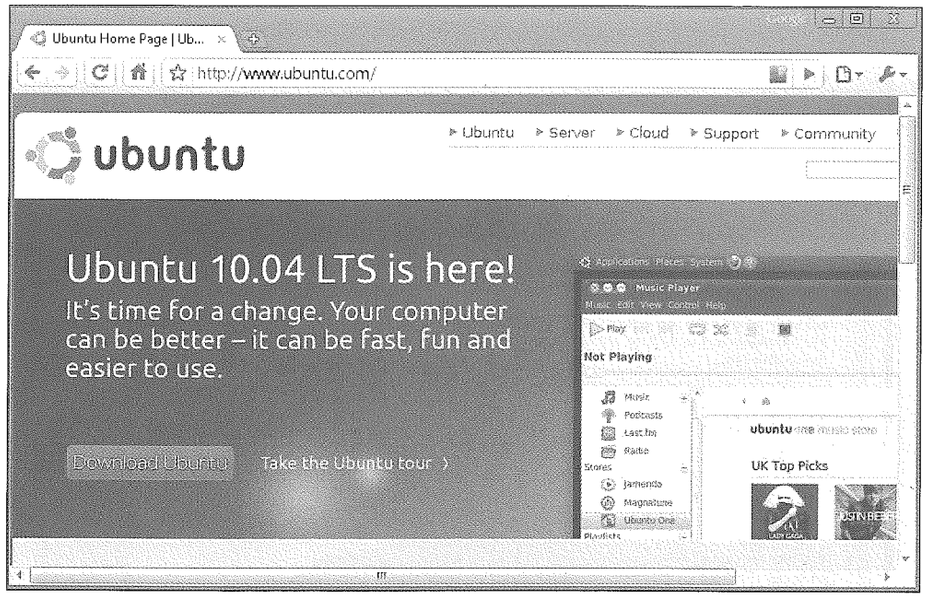

Figura 8. *Uno de los puntos fuertes que explica la popularidad de PHP es que se trata de un lenguaje multiplataforma.*

En este sentido, es importante recalcar que la migración de una aplicación desde un servidor que cuenta con un determinado sistema operativo a otro que cuenta con uno distinto, no es un problema: más allá de las cuestiones específicas no habrá diferencias en cuanto al comportamiento de nuestras aplicaciones. Esto es importante al momento de desarrollarlas: 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 cual esta característica del lenguaje es de suma importancia.

- Soporte para múltiples bases de datos. PHP tiene extensiones para soportar, entre otras, las bases de datos que enumeramos a continuación:

  - DBase
  - Informix
  - Interbase/Firebird
  - Microsoft SQL Server
  - msql
  - MySQL
  - Oracle
  - PostgreSQL
  - SQLite
  - Sybase

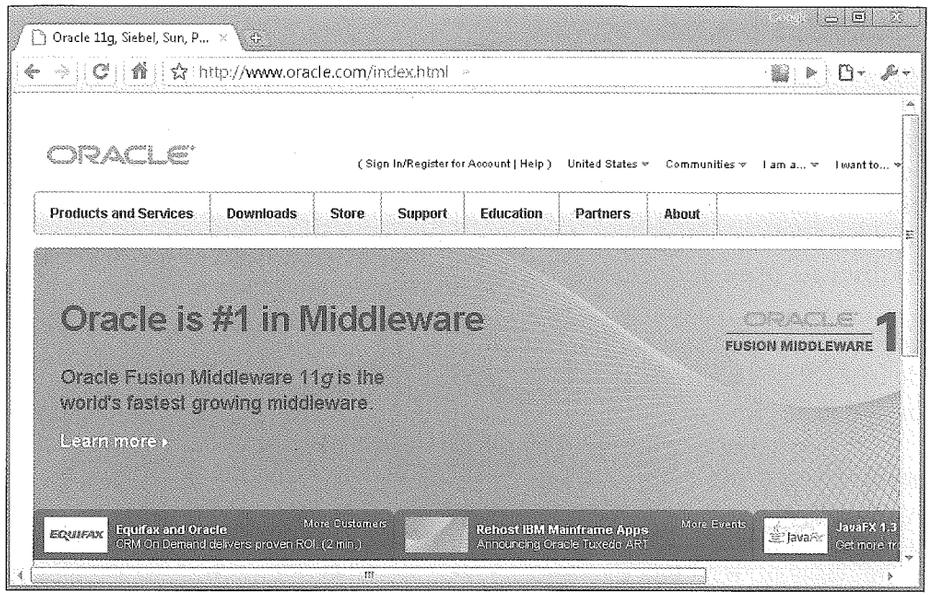

Figura 9. *PHP posee una amplia variedad de extensiones (conjuntos de funciones) para acceder a prácticamente cualquier base de datos disponible en el mercado.*

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.

- Evolución. Otra característica notable es que PHP no es propiedad de ninguna empresa comercial: las continuas mejoras y avances dentro del lenguaje son gracias a una gran comunidad de desarrolladores que contribuyen y opinan acerca de cuales podrían ser los avances incluidos en las próximas versiones y qué cosas deberían ser corregidas en los siguientes lanzamientos.
- Facilidad de aprendizaje. A diferencia de otros lenguajes, PHP se caracteriza por su simpleza: la curva de aprendizaje nos indica que aprender los fundamentos requiere mucho menos tiempo en comparación con otras tecnologías, y que profundizar sobre algunas cuestiones específicas no es tan difícil si contamos con una base teórica sólida y una cierta experiencia a nuestras espaldas. En PHP es más importante saber con precisión qué se quiere hacer que el 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.

Podemos obtener más información acerca del lenguaje (incluida una extensa y sumamente útil documentación a la que recomendamos acudir de manera frecuente) en el sitio web oficial **www.php.net**.

Tecnologías del lado servidor

Como observamos, dentro del ámbito de desarrollo de aplicaciones web contamos con dos conceptos básicos: el de cliente y el de servidor. Fundamentalmente, un cliente es el que realiza peticiones al servidor para que éste, luego de un procesamiento, devuelva un resultado. Por ejemplo, al tipear una URL en la barra de direcciones de nuestro navegador web (un navegador es una aplicación cliente), lo que se está haciendo es enviar una petición o requerimiento al servidor.
Hay distintos tipos de servidores, entre otros:

- web (por ejemplo Apache, IIS, etcétera)
- de bases de datos (por ejemplo MySQL, SQL Server, Oracle, etcétera)
- de correo electrónico (por ejemplo sendmail, qmail, etcétera)

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**, **Oreilly Website Pro**, **Sambar**, **Xitami**, **Caudium**, **Netscape Enterprise Server**, y **THTTPD**, por nombrar algunos ejemplos.

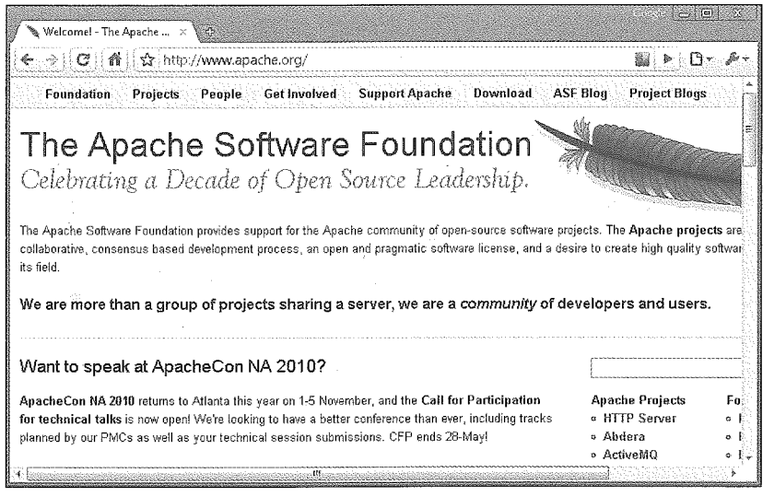

***Figura 10.** Apache es uno de los servidores web más estables hoy en día y está instalado en la gran mayoría de los servidores.*

Todos estos servidores, o por lo menos los web, están instalados en un equipo remoto que es el que recibe las peticiones y devuelve las respuestas. Por este motivo, no es necesario que una máquina cliente (es este contexto un equipo que se utilice 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 desarrollo de aplicaciones web, también podemos categorizarlos: Por un lado, los del lado servidor y, por otro, los del lado cliente. En el primer grupo podemos incluir las siguientes alternativas:

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

***Figura 11.** Perl es un histórico lenguaje de programación que tiene mucho en común con PHP: ambos son libres, multiplataforma, y poseen una gran cantidad de adeptos.*

Con relación a los del lado cliente se incluye, principalmente, el lenguaje de programación JavaScript. Nada nos impide, y de hecho es usual, incluir o utilizar dentro de una misma página lenguajes de distinto tipo.

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.

CÓMO DESARROLLAR SITIOS EN NUESTRA MÁQUINA LOCAL

Al momento de 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, ya que pueden convivir los servidores de prueba en la misma PC. Lo mínimo que necesitaremos será contar con un servidor web (optaremos por uno de los más utilizados y estables de la actualidad, como por ejemplo Apache) y un servidor de bases de datos (en este caso nos inclinaremos por MySQL, por su rapidez, estabilidad, fácil administración, popularidad y gratuidad), además de un lenguaje de programación que será, por supuesto, PHP.

Instalación

Fundamentalmente, hay dos maneras de instalar estas herramientas en nuestro equipo: una es en forma artesanal, es decir, con la descarga, la instalación y la configuración de cada opción por separado; y otra es a través de los llamados **paquetes de instalación**. Estas aplicaciones están disponibles para distintos sistemas operativos (principalmente **Windows**, **Linux** y **MacOs**) y se encargan de automatizar el proceso de instalación liberando al usuario de posibles errores e incompatibilidades y, además, le hace ahorrar tiempo. Se realizan instalaciones estándares, y luego podemos configurar cada herramienta según nuestras necesidades.

Además, ofrecen como valor agregado la instalación de herramientas de administración (por ejemplo **PHPMyAdmin** y **SQLiteManager**, ambas para bases de datos MySQL y SQLite, respectivamente) y programas para poder controlar el funcionamiento de los distintos servidores (inicio, apagado, reinicio, etcétera).
Al descargar esta clase de programas, obtenemos las últimas versiones de cada herramienta, más un instalador que nos permitirá configurar las distintas opciones. Algunas de las alternativas disponibles para realizar esta clase de trabajos son:

HERRAMIENTA	SISTEMA	DIRECCIÓN
AppServ	Windows	www.appservnetwork.com
EasyPHP	Windows	www.easyphp.org
MAMP	OS X de Apple	www.mamp.info/es/home
VertrigoServ	Windows	vertrigo.sourceforge.net
WAMP Server	Windows	www.en.wampserver.com
Server2Go	Windows	http://www.server2go-web.de/
XAMPP	Linux, Windows, MacOS y Solaris	www.apachefriends.org/en/xampp.html

***Tabla 1.** Paquetes de instalación disponibles.*

En el siguiente apartado tomaremos como ejemplo **WAMP**, una de las herramientas más utilizadas por su calidad y sencillez. Al momento de concluir con la instalación de las herramientas, contaremos con todo lo necesario para comenzar a desarrollar aplicaciones en nuestro propio sistema.

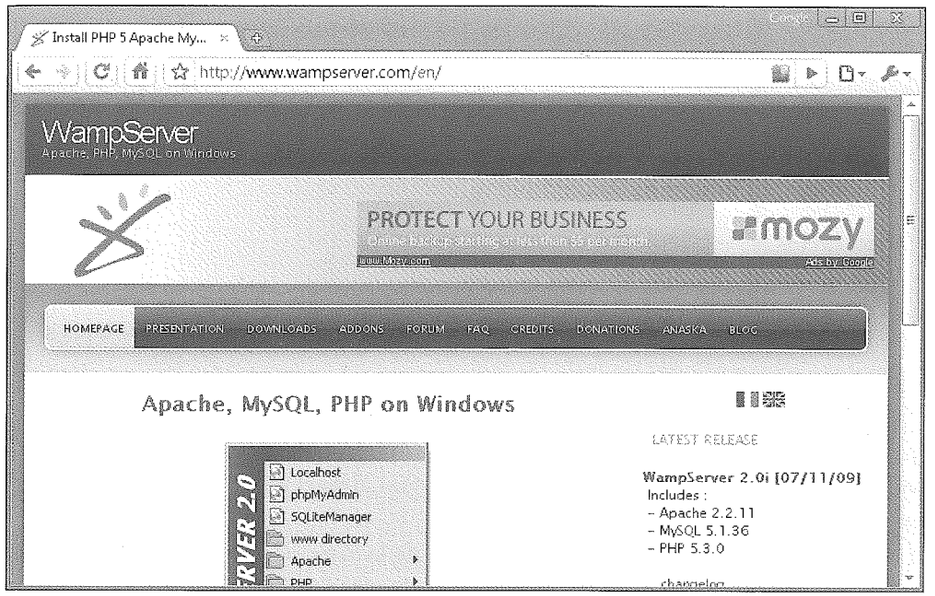

***Figura 12.** Las aplicaciones para instalar las distintas herramientas de desarrollo de manera automática facilitan en gran manera el proceso de armado del entorno de programación de sitios.*

Lo primero que debemos hacer será descargar la versión de WAMP que más se adapte a nuestras necesidades. Lo más usual es optar por la última versión estable disponible. WAMP provee distintas distribuciones, y la diferencia principal entre ellas radica en las herramientas que cada una incorpora. Lo mínimo que tendremos que tener para comenzar a desarrollar aplicaciones será una distribución que cuente con Apache, MySQL, y PHP, la combinación perfecta para desarrollo.

Instalar WAMP

La instalación de este paquete de aplicaciones es igual a cualquier instalación bajo Windows, por lo cual no deberíamos tener inconvenientes. Luego de la pantalla de bienvenida deberemos aprobar las condiciones de licenciamiento (WAMP es una aplicación gratuita) y seleccionar el directorio de instalación (por defecto, **c:\wamp**, en los siguientes apartados asumiremos que se seleccionó este directorio). Depués, un nombre para los accesos directos y, por último, la opción de iniciar o no WAMP de manera automática al comienzo de Windows.

***Figura 13.** Con tan sólo responder algunas simples preguntas, en unos minutos 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** (esto es el directorio en el cual almacenaremos nuestros sitios). Por defecto es **c:\wamp\www**.
En la siguiente sección se nos pregunta acerca de la dirección de nuestro servidor de correo. Por el momento esto no será absolutamente imprescindible para nuestros primeros desarrollos, por ende dejamos el valor por defecto: **localhost** (lo mismo para la dirección de correo, **you@yourdomain**).

En unos momentos veremos cómo la interfaz gráfica de WAMP nos puede permitir acceder de manera rápida e intuitiva a diferentes opciones relativas al manejo y a 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, si es que está instalado (**c:\archivos de programa\Mozilla Firefox**). Una vez concluida la instalación, tendremos la opción de iniciar la aplicación. En el área de notificación de nuestro sistema (esto es, por defecto, en la parte inferior derecha) observaremos un nuevo icono: al hacer clic sobre él podremos acceder a las opciones puestas a disposición por WAMP. Observemos a continuación cuáles son las principales:

- **Localhost** nos da la posibilidad de acceder (a través del navegador por defecto 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.

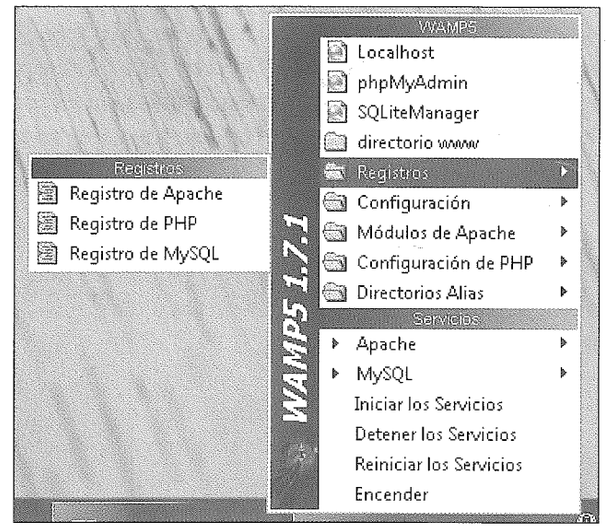

***Figura 14.** El acceso a las funcionalidades provistas por WAMP es sencillo e intuitivo, por eso nos permite, de manera rápida, controlar todo lo relacionado con las herramientas de desarrollo web.*

- **phpMyAdmin** nos permite ingresar, por medio del navegador, a esta popular aplicación que nos habilitará a administrar nuestro servidor MySQL, y así poder manipular las distintas bases de datos que utilizarán nuestros sistemas.

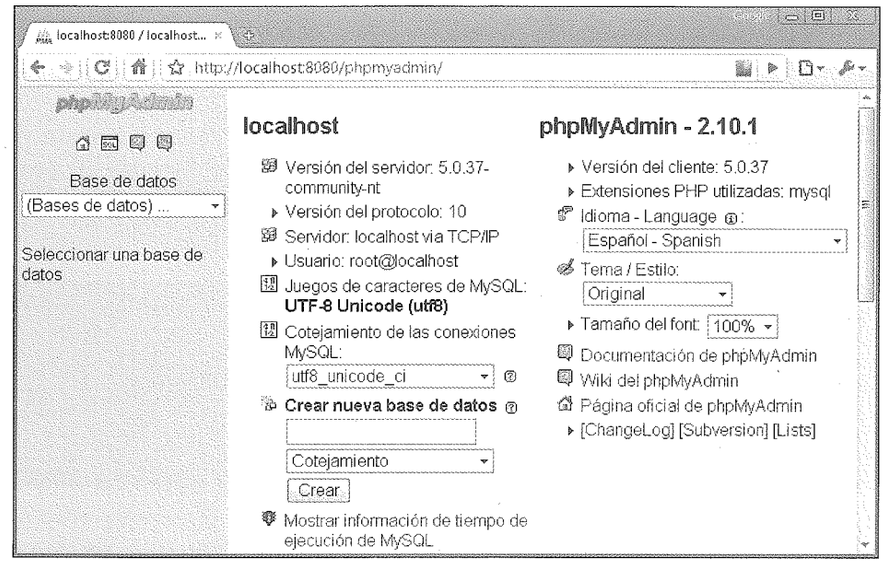

***Figura 15.** phpMyAdmin es una herramienta web que nos permite administrar servidores de bases de datos MySQL de manera grafica.*

- por su parte, **www directory** es un acceso directo al directorio **DocumentRoot** de nuestro sistema, desde el cual tendremos acceso a todos nuestros sitios.
- **Log files** y **Config Files** nos listan los distintos archivos correspondientes al registro de errores y a los archivos de configuración. Desarrollaremos más información acerca de esto en los próximos apartados.
- Tanto **Apache modules** (módulos del servidor web Apache) como **PHP settings** (opciones y extensiones PHP) permiten controlar características específicas y avanzadas acerca de estas herramientas, y habilitar o deshabilitar funcionalidades.
- **Alias directories** nos permite crear accesos directos a determinados sitios locales, por ejemplo **http://localhost/sitio**.

Dentro de la sección **Services**, en el mismo menú, contamos con opciones concernienetes al manejo de los distintos servicios. Al entrar en funcionamiento, tanto Apache como MySQL crean sus propios procesos, que pueden ser **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 (**Start All Services**, **Stop All Services**, **Restart All Services**). 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 su sitio web oficial contamos con una sección de preguntas frecuentes desde la cual podremos resolver cuestiones referidas a la configuración e instalación de la herramienta. La dirección a la que tendremos que acceder es **www.en.wampserver.com/faq.php**.

Primer ejemplo

Ahora, observaremos cómo crear páginas con la utilización del lenguaje de programación PHP. Con cualquier editor de textos (**notepad**, **emacs**, **edit plus**, o cualquier otro) ingresaremos el siguiente contenido:

```
<?php

echo "Esta es mi primer pagina utilizando PHP !";

?>
```

No deberemos preocuparnos si no comprendemos el código anterior, simplemente imprime un mensaje y lo envía a la salida del navegador. Veremos mejor la sintaxis utilizada por PHP en este capítulo y en los que siguen.

Lo que haremos como próximo paso será guardar este archivo bajo el nombre **ejemplo.php** dentro del **DocumentRoot** (éste es el directorio en el cual almacenaremos nuestros sitios; recordemos que fue definido durante la instalación de WAMP y que por defecto es **c:\wamp\www**).

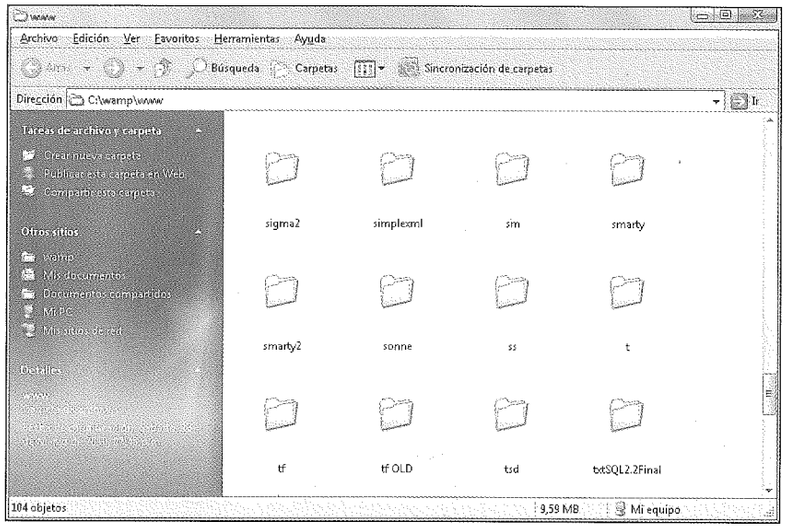

Figura 16. *Para acceder con un navegador a un archivo, éste deberá estar ubicado por debajo del DocumentRoot.*

Una vez hecho esto, verificararemos que el servicio correspondiente a Apache esté activo (en caso de no estar seguros podemos acceder a la opción **Restart All Services**).

Si accedemos desde cualquier navegador a la dirección

```
http://localhost/ejemplo.php
```

podremos observar la salida correspondiente, en este caso, un simple mensaje:

```
Esta es mi primer pagina utilizando PHP !
```

Administrar varios sitios web

En nuestro servidor local podremos mantener más de un sitio, por eso se estila, para mantener un cierto orden y estructurar de manera clara nuestros desarrollos, almacenar cada uno de ellos en un directorio propio, siempre bajo el **DocumentRoot**. Veamos un ejemplo. Primero crearemos las carpetas dentro de **c:\wamp\www**. Cualquier

nombre es válido, pero tomemos **sitio1** y **sitio2**. Dentro de **sitio1** crearemos un archivo al cual denominaremos **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";

?>
```

Y dentro de **sitio2** crearemos otro archivo al cual denominaremos **index.php**, que tendrá el contenido que observamos a continuación:

```
<?php

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

?>
```

La función **date**, vista en detalle en los capítulos que siguen, recibe un formato de fecha hora (en nuestro ejemplo **hora:minutos:segundos** actuales).

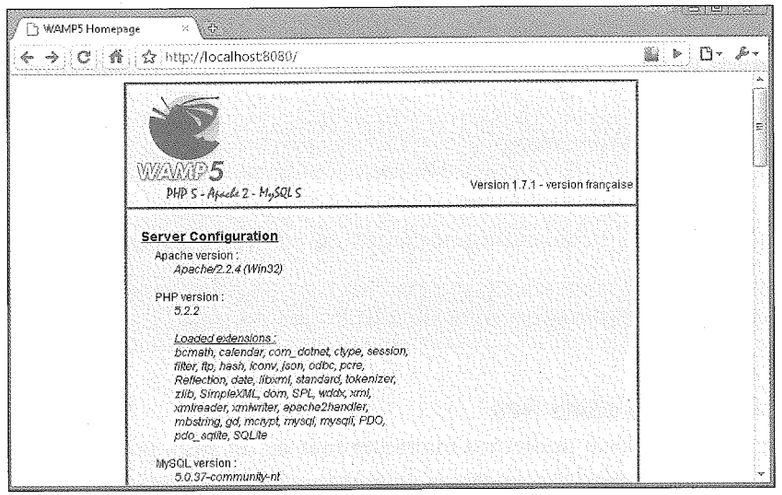

Figura 17. *Con la interfaz de WAMP podemos acceder a todos nuestros sitios locales de manera visual por medio de un listado de directorios.*

Una vez creados y guardados los archivos, podremos ingresar a nuestra página de inicio a través WAMP (opción **localhost**) o simplemente si escribimos la siguiente dirección en la barra de direcciones de nuestro navegador:

```
http://localhost/
```

Allí encontraremos un listado con nuestros sitios disponibles y estaremos habilitados para seleccionar cualquiera de ellos si queremos acceder a su contenido. Notemos que las direcciones son del tipo que aparece a continuación:

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

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

Definir la página principal del sitio

Otro aspecto a tener en cuenta: a diferencia de lo que ocurría en nuestro primer ejemplo (**ejemplo.PHP**) en donde debíamos indicar el nombre de la página, al ingresar tanto a **sitio1** como a **sitio2** 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, en caso de no especificar uno en concreto en las URLs escritas en la barra de direcciones del navegador.

```
DirectoryIndex index.php index.html index.htm
```

Esta directiva es configurable por parte del usuario. En caso de que no exista uno de los archivos especificados 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 por algún motivo no es conveniente que cualquiera que ingrese a nuestro sitio los vea, será necesario ubicar un archivo índice.
Otra opción importante de este archivo es **DocumentRoot**, que permite definir el directorio raíz desde el cual se podrá acceder a los documentos con 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 que esté dentro del directorio **DocumentRoot**. En cada uno de estos ficheros se pueden ubicar las directivas de configuración del **httpd.CONF**, la diferencia es que los valores de las directivas que se encuentran dentro de un fichero **.HTACCESS** prevalecen frente a los valores de configuración especificados dentro del fichero **httpd.CONF**.

Inclusión de 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. Recapitulemos los conceptos principales:
1) Crear o editar un archivo que deberá tener la extensión **.PHP**: esto varía con las versiones, y actualmente puede llegar a ser probable encontrarnos con documentos de extensión **.PHP3** (en este caso se indica que la página está programada utilizando características específicas de PHP versión 3 y que eventualmente habrá conflictos si el servidor sobre el cual trabajamos no cuenta con ella), por ejemplo. Más allá de esta curiosidad, 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 observar líneas como las siguientes:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php3
```

Este caso particular indica que al encontrar un petición que invoque a un archivo de extensión **.PHP** o **.PHP3**, se pedirá al interprete PHP que resuelva el código allí escrito. Este comportamiento puede aplicarse a otros tipos de archivo, no sólo a los que están directamente 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 que vimos antes:

```
<?php

//código

?>
```

Pero, eventualmente, contamos con otras:

```
<?

//código

?>
```

```
//código
```

MÁS INFORMACIÓN

En la actualidad, hay un acceso casi ilimitado a código PHP: sitios web, revistas, libros y cursos son algunas de las fuentes de información que nos permitirán tomar nota sobre técnicas de programación y soluciones que luego podremos aplicar en nuestros propios proyectos.

```
%>
```

Para incluir código PHP en un documento HTML, sólo debemos ubicar nuestro 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 código generado por el preprocesador PHP. La
     hora actual es $hora </b>";

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

Por último, habrá que copiarlo a un directorio que esté bajo el **DocumentRoot** (Directorio raíz), del servidor, e invocarlo desde un navegador web. A continuación, en el ejemplo, veamos cómo quedaría la salida de éste:

```
<!DOCTYPE html
```

VINCULACIÓN ENTRE COMPONENTES

En los próximos capítulos, veremos cómo PHP interactúa de manera natural con los demás componentes que forman parte del entramado de una aplicación web, tal es el caso de las bases de datos (por ejemplo, MySQL o SQLite) o los servidores web (por ejemplo, Apache).


```
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 código generado por el preprocesador PHP. La hora actual es
     20:16:42 </b>
 </body>
</html>
```

En los próximos capítulos, veremos cómo seguir avanzando en el desarrollo agregando más funcionalidades a nuestras páginas a través de las diversas posibilidades que nos brinda PHP como lenguaje de cabecera. Además, aprenderemos sobre uno de los servidores de base de datos más populares de la actualidad.

Extensiones del lenguaje

Las extensiones o bibliotecas son conjuntos de funciones disponibles para programar aplicaciones. Estas extensiones pueden venir incorporadas al lenguaje, o tal vez necesiten ser añadidas de manera explícita.

Una vez que se instalan y se habilitan esas bibliotecas, el comportamiento de las funciones componentes dentro del código de nuestros programas es idéntico al de cualquier función o procedimiento, es decir, la programación se vuelve independiente y transparente al origen de las funciones. Desarrollaremos todo lo referido a la invocación y creación de funciones en los próximos apartados.

Para habilitar o deshabilitar extensiones podemos modificar el archivo **php.ini** o, desde WAMP, con la opción **Configuración de PHP > Extensiones de PHP**. Según la versión que estemos utilizando, PHP nos informará, por medio de mensajes, qué funciones no están disponibles o cuáles lo estarán pronto.

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 mantener desarrollos prolijos y comprensibles.

Ejemplo sobre funciones

Para 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 estuviera deshabilitada podríamos observar algo similar a lo siguiente:

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

Por otra parte, hay que aclarar que, en este caso, para acceder a un servidor de bases de datos a través de estas funciones, éste deberá 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 extras, una gran cantidad de bibliotecas, por lo cual contaremos con múltiples funciones para comenzar a desarrollar sitios profesionales. Entre estas extensiones se encuentran:
Las extensiones se pueden categorizar en estados (estables, obsoletas, y experimentales). Es posible obtener un listado accediendo a http://www.php.net/manual/es/extensions.state.php.

- Funciones para manejo de matrices
- Funciones matemáticas
- BCMath (desde PHP 4.0.4, más funciones matemáticas)
- Para manejo de clases/objectos
- 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 extensiones) tengan efecto, habrá 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(); ?>
```

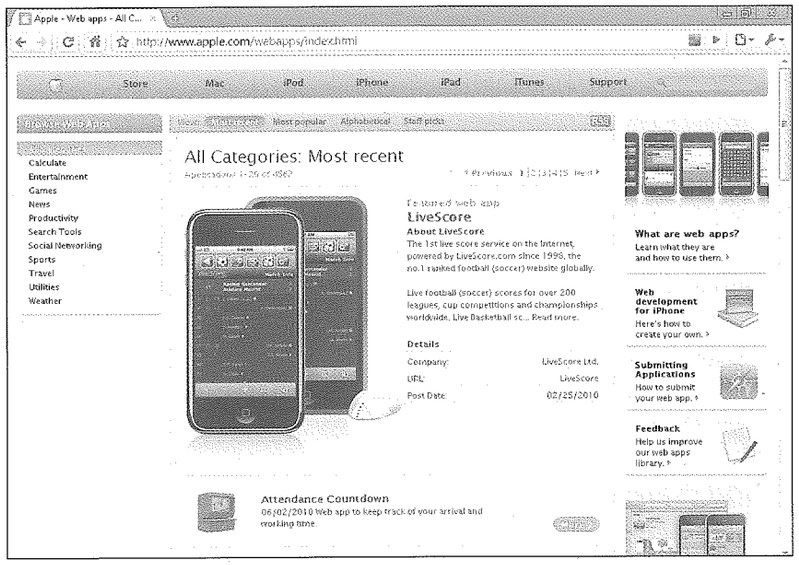

Figura 18. *PHP es un lenguaje multipropósito, y podemos utilizarlo, incluso, para desarrollar aplicaciones web para el iPhone OS.*

RESUMEN

En este capítulo, aprendimos los puntos básicos que nos permitirán, a lo largo de nuestra explicación, avanzar y asimilar la forma de trabajo para poder generar aplicaciones web con la utilización de PHP en conjunto con otras tecnologías, e instalar el paquete WAMP, que incluye las herramientas necesarias para desarrollar en este lenguaje.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Qué es una aplicación web?

2 ¿Cuál es la diferencia con respecto a una aplicación de escritorio?

3 Nombre tres características propias de PHP.

4 ¿Qué es una aplicación cliente-servidor?

5 ¿Qué es un servidor web?

6 ¿Qué es un servidor de bases de datos?

7 ¿Cuál es la relación entre un servidor web, una base de datos, y un lenguaje de programación?

8 ¿Es posible desarrollar aplicaciones web en una máquina sin conexión a Internet?

9 ¿Cómo se llama el archivo de configuración del entorno PHP?

10 ¿Qué carácter debe tener una línea de este archivo por delante para deshabilitar una configuración o característica en PHP?

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 a su directorio DocumentRoot a través de WAMP.

4 ¿Cómo podría comprobar que su servidor web está activo?

5 Busque y cambie el nombre de la página principal de inicio en PHP, para ver qué sucede. Luego restaure al nombre original.

Capítulo 2

Sintaxis básica

En este capítulo, haremos un repaso por las características más distintivas de PHP como lenguaje de programación y nos adentraremos en la forma y en las herramientas que brinda para implementar aplicaciones.

Introducción	**40**
Instrucciones	40
Variables	40
Constantes	44
Expresiones	44
Comentarios	49
Tipos de datos	52
Estructuras de control	**59**
Inclusión de archivos	68
Funciones	68
Resumen	**71**
Actividades	**72**

INTRODUCCIÓN

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

Instrucciones

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

```
<?php

?>
```

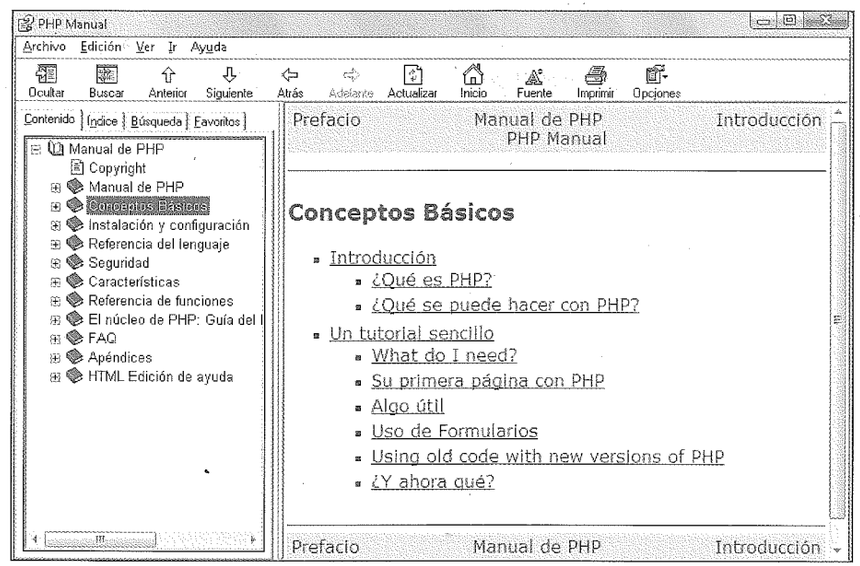

***Figura 1.** El manual oficial de PHP es una fuente indispensable en todo momento ya que nos ayudará a resolver dudas típicas.*

Variables

Una **variable** es un lugar reservado para almacenar valores. Cada una está identificada por un nombre, generalmente referido al tipo de dato que almacenará, y comienzan

con el signo $ seguido de su nombre, el cual deberá empezar con una letra o un guión bajo y, a continuación, cualquier número de letras, guiones bajos, y números. Veamos a continuación algunos nombres válidos para variables. En la líneas de código siguientes, podemos ver algunos nombres válidos para las variables vistas anteriormente.

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

En el siguiente ejemplo vemos nombres no válidos para variables:

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

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

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

FUNCIONES

En los capítulos que siguen veremos las funciones disponibles relacionadas con cada tipo de dato provisto por PHP. Éstas nos permitirán aumentar la capacidad de nuestras aplicaciones y nos darán más herramientas para lograr los objetivos propuestos durante la planificación previa.

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

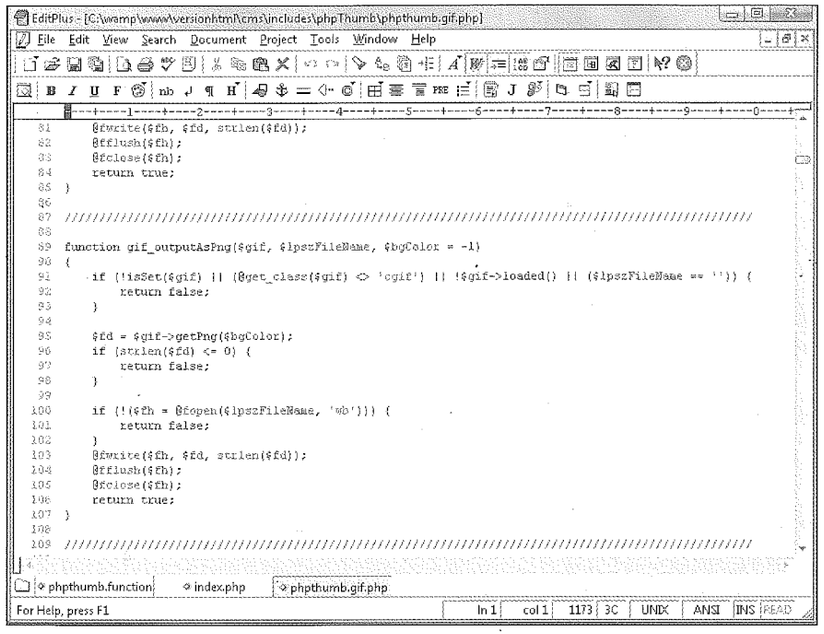

Figura 2. *Utilizar un editor que reconozca código PHP nos permitirá escribir aplicaciones de manera segura y encontrar errores sintácticos rápidamente.*

- La construcción **echo** recibe como argumentos, variables o valores separados entre comas, y los muestra como salida, como en el ejemplo siguiente:

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

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

- Por su parte, **print** recibe como argumento una cadena de caracteres y la muestra en pantalla, podemos ver el ejemplo, a continuación:

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


PHP nos permite "variar el nombre de las variables", algo conocido como **variables variables**. Para lograr este cometido se utilizan dos signos **$** en lugar de uno. Veamos un claro ejemplo de cómo se utiliza:

```
<?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 algunas 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;

echo gettype($var1); //string
echo gettype($var2); //integer

?>
```

También, dentro del lenguaje, 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 caso de no pertenecer:

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

Veremos más acerca de arrays, numéricos, y strings en el próximo capítulo. En el **Capítulo 5** haremos una introducción a la programación orientada a objetos.

Constantes

Las constantes en PHP se generan a través de la función **define**, que recibe como argumentos un nombre y un valor para la constante. 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." días";

?>
```

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

Expresiones

PHP soporta el pre y el post incremento y decremento. Veamos algunos ejemplos, a continuación, 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

?>
```

Otra manera de asignar es a través del operador condicional ternario. La sintaxis es:

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

Veamos un ejemplo de esto a continuación:

```
<?php

$tmp = 10;

$variable = $tmp > 5 ? 'Es mayor a 5' : 'No es mayor a 5';

echo $variable; // Es mayor a 5

?>
```

Además, también tenemos la alternativa de poder hacer uso de las asignaciones múltiples que van de derecha a izquierda:

```
<?php

$var = 10;

$var1 = $var2 = $var + 1;

echo $var1;

?>
```

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

- > (mayor que)
- >= (mayor o igual que)
- == (igual que)
- != (distinto)
- < (menor que)
- <= (menor o igual que)

PHP posee más operadores y podemos acceder al listado completo si visitamos **www.php.net/manual/es/language.operators.php**.

Una expresión se evalúa como verdadera si es verdadera, o distinta de falso. Según PHP, algo es falso cuando es nulo, cero, vacío, etcétera.

Un punto muy importante es que las expresiones se aplican desde la más interna hasta la más menos, por ejemplo, el siguiente código:

```
<?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.
Más allá 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** almacena 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'];

?>
```

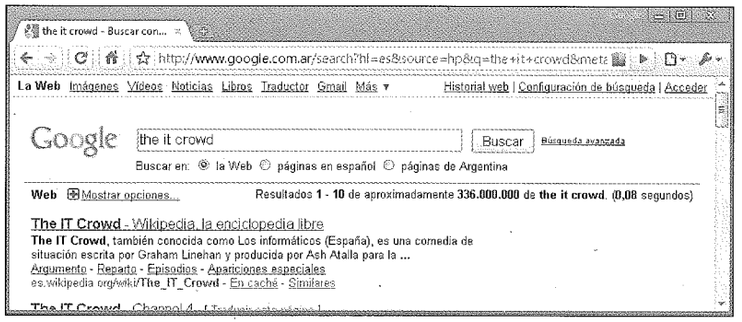

***Figura 3.** El paso de valores a través de páginas es una manera tradicional de construir aplicaciones web ligando sus partes.*

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

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

- **$_POST** almacena las variables pasadas a través del método POST. El caso más común es el envío de formularios, lo podemos ver en la siguiente línes de código:

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

- **$_COOKIE** almacena información acerca de las cookies guardadas en la máquina del cliente. Las cookies son estructuras que las páginas almacenan en archivos de texto ubicados en la máquina del usuario, generalmente dentro de un directorio específico del navegador, y sirven para que los sitios puedan reconocer a los visitantes a través de la identificación de los valores contenidos. Desarrollaremos más acerca de las cookies en el próximo capítulo.
- **$_FILES** guarda información acerca de los archivos subidos por formulario. Mientras que a los campos tradicionales (**text**, **radio**, **checkbox**, **textarea**, etcétera) accedemos a través de los métodos **$_POST**, **$_GET**, o **$_REQUEST**, a los de tipo **file** accedemos al utilizar este array.

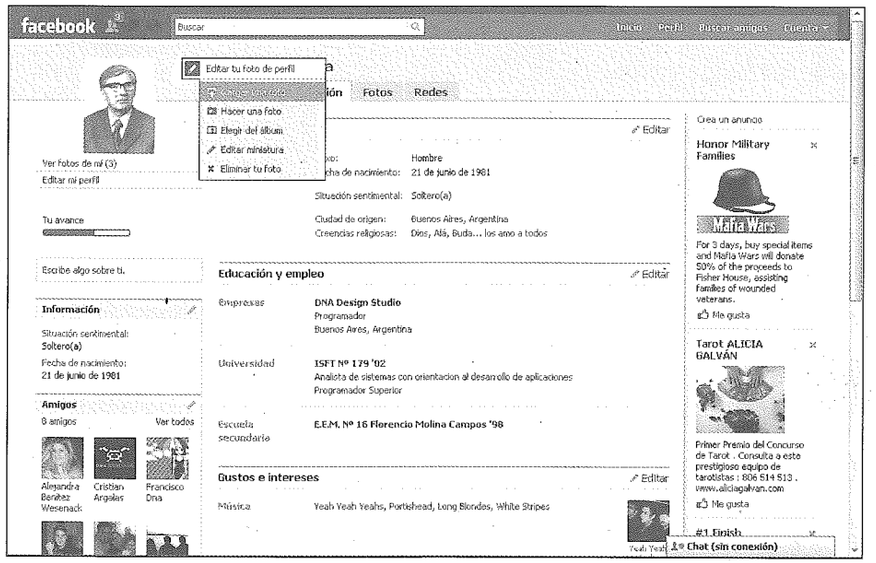

***Figura 4.** La carga de archivos forma parte de aplicaciones de cualquier tipo, dirigidas a toda clase de usuarios.*

- **$_ENV** nos permite acceder a ciertas informaciones relacionadas con la máquina desde la cual se ejecuta PHP, como por ejemplo, nombre del equipo, directorios especiales, sistema operativo, etcétera.
- **$_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 aquellas 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 sólo una vez. Si los datos son correctos, se inicializa una variable de sesión y se consulta su valor cada vez que el usuario intenta ingresar en una de estas secciones. Veremos más acerca de sesiones en el próximo capítulo.

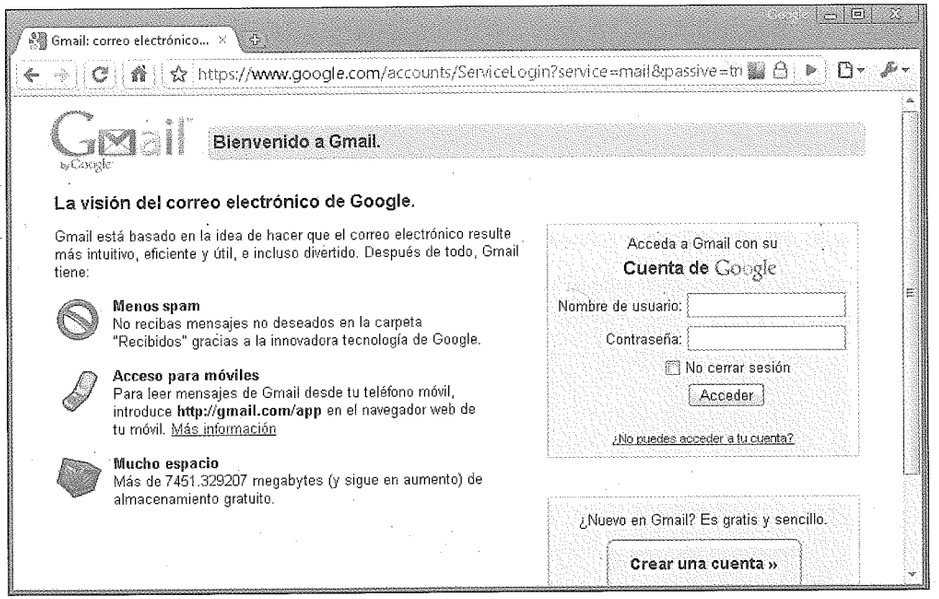

***Figura 5.** El envío de datos por formulario puede ser manejado por las aplicaciones PHP de manera sencilla a través de los arrays prediseñados para tal fin.*

Comentarios

Los comentarios se suelen utilizar para incluir textos que aclaren sobre la composición del programa y 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 está comentado.";

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

?>
```

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 //Aquí se termina el bloque actual ?>
```

Por su parte, los comentarios realizados en múltiples líneas tienen la siguiente sintaxis:

```
<?php

/*

Esto es un comentario
de múltiples
líneas

*/

?>
```

Esto puede usarse para delimitar el comienzo y final de un comentario:

```
<?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
```

PROCEDENCIA DE LOS TIPOS DE COMENTARIOS EN PHP

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

Los únicos caracteres continuos que no podemos incluir en cualquier clase de comentarios son los que se utilizan como delimitador de cierre (*/):

```
<?php

/* Abrimos el comentario

    */

Intentamos cerrar el comentario */

?>
```

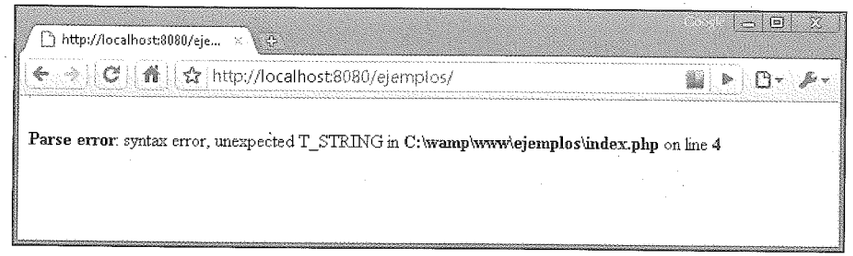

***Figura 6.** La mala implementación de comentarios puede provocar errores en nuestras aplicaciones.*

El **preprocesador PHP** no incluye el contenido de los comentarios en la salida generada, por lo cual al contenido de éstos sólo podrán acceder aquellos que 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:

- 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

?>
```

Estudiaremos los arrays con más detenimiento en los próximos capítulos; por ahora sólo diremos que una matriz es una colección de variables y, en caso de contar con cero elementos, se considera **false**.

- Entre los tipos de datos numéricos contamos con dos opciones: los enteros (**integer**) y los flotantes (**float**). En lo que rspecta a los primeros, PHP pone a nuestra disposición 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, como en el ejemplo a continuación:

```
<?php

$variable = -101;
// Entero decimal negativo

?>
```

Un tipo de dato **float** (también conocido como **punto flotante, doble, número real**) es un numérico no entero que posee 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 máximo de un entero: '.PHP_INT_SIZE;
echo '<br />Valor máximo de un entero: '.PHP_INT_MAX;

?>
```

En caso de que llegemos a sobrepasar estos valores, el tipo entero que almacene el valor que supere su tipo será convertido automáticamente a flotante:


```
<?php

echo var_dump(PHP_INT_MAX);
//tipo integer

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

?>
```

- Las cadenas de caracteres en PHP son asignadas a variables de dos maneras: rodeadas por 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.';

?>
```

Y 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 siguiente 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) lo que hace es buscar un caracter **$** (comienzo de variable) hasta el final del nombre, como ser, un espacio, un punto y coma, dos puntos, otro signo **$**, o cualquier caracter que indique que el nombre de la variable ha terminado. Luego reemplaza el nombre de ésta por el valor correspondiente. Si la variable no existe, 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 hubiera utilizado las llaves 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, como vemos a continuación:

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

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

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

- Una **matriz** podría 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**, como vemos en la línea de código siguiente.

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

El array del ejemplo anterior contiene 3 posiciones. Podemos acceder a los valores de cada posición 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 al comenzar desde el número 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
```

Y también podemos asignar nuevas posiciones:

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

En cuanto a los valores que podemos utilizar, éstos pueden ser de cualquier tipo, hasta inclusive pueden ser otros arrays, lo vemos en las líneas de código:

```
<?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 provistas por PHP para el tratamiento de arrays en el siguiente capítulo de este manual.

ESTRUCTURAS DE CONTROL

PHP cuenta con construcciones que nos permiten realizar tareas variadas, por ejemplo, tomar una accion u otra dependiendo del valor de una expresion, 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.
Comencemos con la construccion **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 sólo con una instrucción. Podemos evaluar más de un expresión si utilizamos la construccion **else if**:

```
<?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**, como podemos ver a continuación:

```
<?php

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

?>
```

En estas situaciones, 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 "Decada del 1900";
} elseif ($anio >= 1910 && $anio < 1920) {
  echo "Decada del '10";
} elseif ($anio >= 1920 && $anio < 1930) {
  echo "Decada del '20";
} elseif ($anio >= 1930 && $anio < 1940) {
  echo "Decada del '30";
} elseif ($anio >= 1940 && $anio < 1950) {
  echo "Decada del '40";
} elseif ($anio >= 1950 && $anio < 1960) {
  echo "Decada del '50";
} elseif ($anio >= 1960 && $anio < 1970) {
  echo "Decada del '60";
} elseif ($anio >= 1970 && $anio < 1980) {
  echo "Decada del '70";
} elseif ($anio >= 1980 && $anio < 1990) {
  echo "Decada del '80";
} elseif ($anio >= 1990 && $anio < 2000) {
  echo "Decada del '90";
} else {
  echo "El año no esta 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**, lo que se conoce generalmente como **if** anidados.

A continuación, podemos ver en las líneas de código una sintaxis alternativa que utilizamos pocas veces, pero válida de todos modos.

CICLOS

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

```
<?php

if (expresion1):
  //instrucciones
  //instrucciones
elseif (expresion2):
  //instrucciones
  //instrucciones
elseif (expresion3):
  //instrucciones
  //instrucciones
else:
  //instrucciones
  //instrucciones
endif;

?>
```

Vemos 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 (}) es reemplazada por la siguiente evaluación: **elseif**, **else**, o **endif**.
Si tenemos la necesidad de ejecutar un mismo fragmento de código un número N de veces, contamos con varias alternativas: una de ellas está dada por la construcción **while** (muy utilizada en casi todos los lenguajes) que recibe como argumento una expresión. Las instrucciones contenidas se ejecutan **mientras** la expresión sea verdadera. La sintaxis que se utiliza para esto es la siguiente:

```
<?php

while (expresion) {
  //instrucciones
}

?>
```

Como en el caso de **if**, si contamos con una sola instrucción podemos omitir las llaves apertura y cierre, una característica única, propia de PHP.
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.';
}

?>
```

Modifiquemos valores para que la expresión sea falsa en algún momento:

```
<?php

$variable = 10;

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

?>
```

—HASTA ACÁ

En las estructuras repetitivas (como **while**, por ejemplo) se habla de **bucles** (ciclos) e **iteraciones** (vueltas). En el bucle **while** anterior se producen 90 iteraciones.
Puede suceder, incluso, que la expresión sea falsa en todo momento (o por lo menos antes del inicio del ciclo) y que entonces no se produzcan iteraciones:

```
<?php

$variable = 10;

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

?>
```

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

```
<?php

$variable = 100;

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

?>
```

while evalúa la expresión antes de ejecutar instrucciones. PHP cuenta con un bucle llamado **do while** que permite ejecutar 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 vimos 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 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 cominezo 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 pueda finalizar en algún momento.

La sintaxis alternativa podemos verla en las siguientes líneas de código:

```
<?php

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

?>
```

Los bucles **foreach** permiten recorrer un array. La sintaxis utilizada es la que sigue:

```
<?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;
}

?>
```

El bucle **switch** equivale a un conjunto de sentencias de tipo **if ... elseif ... else**:

```
<?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 a evaluar por cada **case** puede ser de cualquier tipo. Si un **case** coincide con el valor a 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";
}

?>
```

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 />i igual a $i";

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

?>
```

En caso de querer saltear sólo 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 que tenemos son las siguientes:

- **require** incluye el archivo pasado como argumento. Un error equivale a fatal.
- **include** incluye el archivo pasado como argumento. Ante un error produce sólo una advertencia.
- **require_once** es similar a **require**, pero incluye el archivo solamente una vez, aunque intentemos hacerlo más de una.
- **include_once** es similar a **include**, sin embargo, incluye el archivo solamente una vez, aunque intentemos hacerlo más de una.

```
<?php

// codigo

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

// codigo

require 'docs/pagina1.php';

?>
```

Todas las variables, constantes, funciones, etcétera 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. Una función en PHP tiene la forma:


```
<?php

function nombreFuncion($arg1, $arg2, $argN) {
  // codigo de la funcion
  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. Un 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 dentro 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 radica en 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 !";
}

?>
```

Así como tenemos **Variables variables**, 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 dentro 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 funcion
  return $a * $b;
}

?>
```

Otra opción es utilizar el array **$GLOBALS**. Con el mismo sentido, si anteponemos **static**, conservan su valor ante múltiples llamadas.

A lo largo de los siguientes capítulos, profundizaremos en las herramientas que PHP pone a nuestra disposición para desarrollar aplicaciones.

RESUMEN

En este capítulo, observamos cuáles son las características generales más importantes de PHP como lenguaje: sus opciones, su sintaxis básica y un poco más avanzada, y las diversas alternativas que nos ofrece para llegar a obtener un determinado resultado. También vimos cómo utilizar recursos o archivos externos de script dentro de otros scripts.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Cuál cree que es la principal cualidad de PHP?

2 ¿Cree que se trata de un lenguaje complejo?

3 ¿Cuáles son los tipos de datos disponibles?

4 ¿Cuál es la diferencia entre un tipo entero y un tipo flotante?

5 ¿Para qué se utiliza la construcción echo?

6 ¿Qué es un array?

7 ¿Qué es un ciclo?

8 ¿Qué es una iteración?

9 ¿Qué diferencia hay entre elseif y else?

10 ¿Qué diferencia hay entre while y do while?

EJERCICIOS PRÁCTICOS

1 Desarrolle una aplicación que reciba dos números y devuelva el mayor.

2 Desarrolle una aplicación que reciba una cadena de caracteres y devuelva el número de vocales en la misma.

3 Desarrolle una aplicación que muestre un texto por pantalla 10 veces.

4 Desarrolle una aplicación 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 anterior para controlar que el número ingresado no sea menor a 1 ni mayor a 12.

PHP 6

Capítulo 3

Referencia de funciones

En este capítulo, observaremos algunas de las extensiones disponibles en la versión base de PHP, que utilizaremos de manera frecuente en nuestra vida diaria como programadores.

Extensiones disponibles	**74**
Funciones para el manejo de sesiones	75
Funciones para el manejo de cookies	85
Funciones del sistema de archivos	87
Funciones para el manejo de fecha y hora	103
Funciones para el tratamiento de cadenas de caracteres	112
Funciones matemáticas	**130**
Funciones para tratamiento de matrices	135
Resumen	**145**
Actividades	**146**

EXTENSIONES DISPONIBLES

PHP posee una inmensa cantidad y variedad de extensiones y sin duda, éste es uno de los puntos más fuertes del lenguaje, una de las razones de su popularidad. Para tener una idea veamos la siguiente tabla:

FUNCIONES Y EXTENSIONES DISPONIBLES PARA PHP		
Acceso directo a E/S	Gestión de funciones	OpenSSL
Apache	Haru PDF Functions	Oracle
BBCode	HTTP	Para compilador Bytecode de PHP
Cache	Hyperwave	PDF
Cadenas de caracteres	IBM DB2	PostgreSQL
Calendario	iconv	PostScript
Capa de abstracción de bases de datos	IIS	Rar
Cifrado Mcrypt	Imágenes	Red
Clases / Objetos	Imagick (Image Library)	Secuencias
COM y .Net (para plataformas Windows)	IMAP, POP3 y NNTP	Shockwave Flash
Compresión Bzip2	Impresora	SimpleXML
Compresión Zlib	Informix	Sistema de archivos
Control de procesos	Ingres II	SNMP
Control de salida	Integración de Java y PHP	SOAP
Correo	IRC	Sockets
Cracking	JSON	SPL (Standard PHP Library)
CURL (Client URL Library)	LDAP	SQLite
DB++	libxml	Subversion
dBase	Lotus Notes	Sybase
dbx	Manejo de archivos Zip	TCP
Depurador avanzado de PHP	Manejo de sesiones	Tidy
Directorio	Matemáticas	Tipo de caracter
DOM	Matemáticas de precisión arbitraria BCMath	Tratamiento de etiquetas ID3
DOM XML	Matrices	Unicode
Expresiones regulares	Memoria compartida	URL
Fecha y hora	Mhash	Verisign Payflow Pro
filePro	Microsoft SQL Server	WDDX
Firebird / InterBase	Mimetype	XML
FriBiDi	Miscelánea	XMLReader
FrontBase	mSQL	XML-RPC
FTP	MySQL	XSL
GeoIP	MySQLi	XSLT
Gestión de errores y registros	ODBC	

***Tabla 1.** Listado de algunas funciones o plug-ins para utilizar desde PHP.*

Éstas son sólo algunas opciones: el lenguaje aumenta el número de extensiones en cada versión y perfecciona las ya existentes. 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, aquéllas 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 –debemos tener en cuenta que estamos hablando de páginas de 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 compruebe su propia identidad, solicitándole que ingrese ciertos datos (por ejemplo, **username / password**) para validar su acceso al sistema. Si hay coincidencia (la información podría compararse contra una base de datos, como veremos en los próximos capítulos) se inicializa (es decir, se asigna un valor en particular) a una variable de sesión.

***Figura 1.** En cuanto a prestaciones, la facilidad en el manejo de sesiones diferencia a PHP de otros lenguajes similares.*

El lenguaje PHP almacena las variables de sesión en un array denominado **$_SESSION**. Su forma de utilización 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 la versión 4.1.0 de PHP en adelante se utiliza el array **$_SESSION**, pero en la versión 4.0.6 e inferiores solía emplearse el array especial **$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 que sigue:

```
<?php

session_start();

$_SESSION['nombreVariable'] = 'Este es el valor';

echo $_SESSION['nombreVariable'];

?>
```

Debemos notar, aquí, que 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();

?>
```


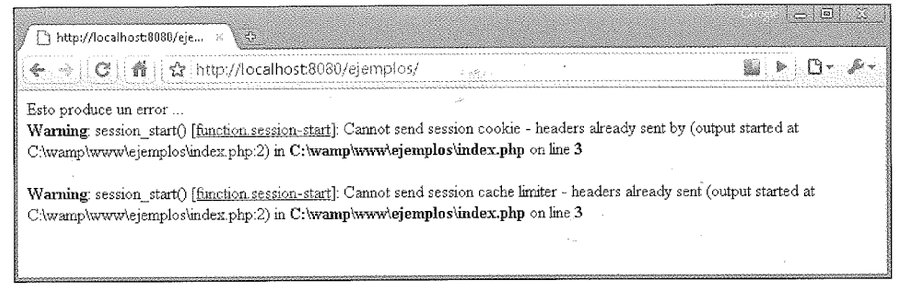

***Figura 2.** El inicio de sesiones no admite ninguna salida anterior a través del navegador web, como vemos en la imagen.*

En algunas configuraciones no es necesario invocar de manera explícita **session_start**, si es que está definida en el archivo **php.INI** la directiva **session.auto_start**:

```
session.auto_start = 1
```

En estos casos no haría falta llamar a **session_start**. Para destruir todos los datos guardados correspondientes a una sesión, contamos con la función **session_destroy**, que no recibe argumentos y devuelve verdadero en caso de éxito y falso si por algún motivo no se logró llevar a cabo la destrucción.

```
session_destroy();
```

Por su parte, 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 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, podemos verlo en el siguiente ejemplo:

```
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**, lo vemos ejemplificado a continuación.

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

PHP le asigna a cada usuario que inicia una sesión, un identificador único denominado "identificador de sesión" (o **session id**). Si queremos conocer nuestro identificador actual o incluso asignarle un nuevo valor tenemos en PHP la función **session_id**, la cual, invocándola con **echo**, nos devolverá su valor:

```
<?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 forma que vemos a continuación:

```
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 la función **session_regenerate_id**.
La función **print_r** puede sernos de utilidad para visualizar todas las variables contenidas en un array específico, no sólo **$_SESSION**.
Veamos un ejemplo para explorar sus posibilidades:

```
<?php

session_start();

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

echoArray($_SESSION);

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

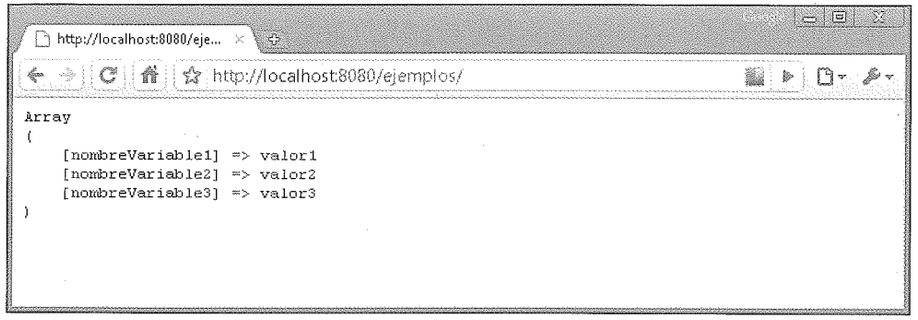

***Figura 3.** Podemos mantener múltiples variables de sesión en un mismo script en forma simultánea, como sucede con las demás clases de variables.*

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, lo hacemos de la siguiente manera:

```
<?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 nuestras páginas, antes de **session_start** incluso(ver línea de código a continuación).

```
session.name = PHPSESSID
```

La función **session_name** devuelve el nombre actual de la sesión, por lo tanto, algo como las siguientes líneas de código son válidas.

```
<?php

$nombreAnterior = session_name('nombreNuevo');

?>
```

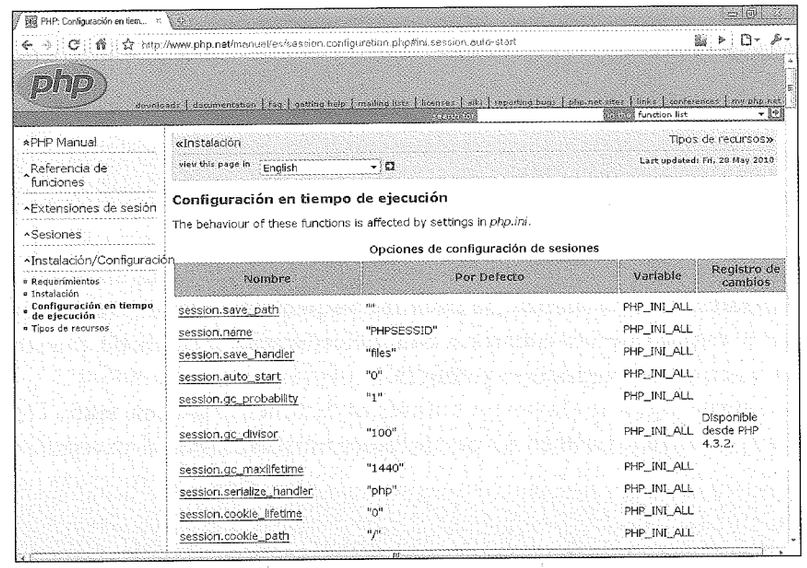

***Figura 4.** El archivo **php.ini** mantiene directivas importantes con relación al trabajo con sesiones en aplicaciones PHP.*

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

```
<?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 una directiva llamada **session.cookie_lifetime** para definir el tiempo de vida (en segundos) de las sesiones. Si se le asigna 0, la sesión no caduca caducará hasta que 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 marcha algunas de las funciones repasadas anteriormente. Se trata de un pequeño script que nos permitirá validar el ingreso de un usuario a una determinada sección de nuestro sitio web (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 que sigue:


```
<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>
```

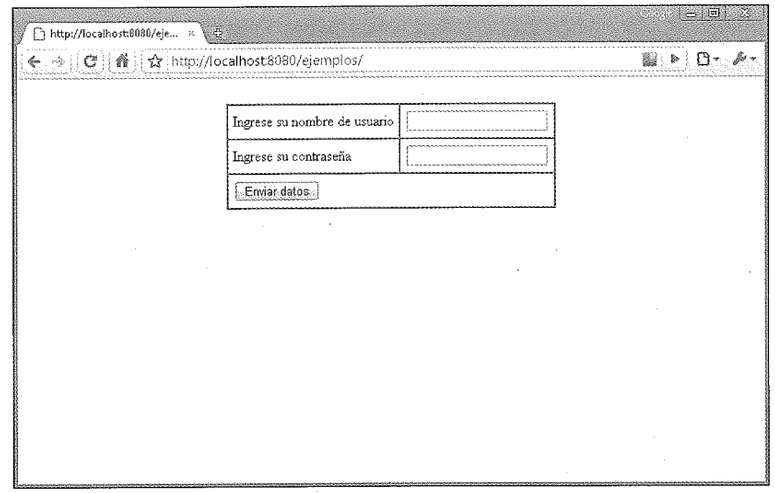

Figura 5. *El trabajo con sesiones nos permite resolver tareas cotidianas, como el acceso de usuarios a contenidos restringidos.*

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"] && $password =
     $_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 (con la función **header**) para que los datos de POST (el método utilizado en el formulario) no se reenvíen si el usuario refresca la página. Si la variable de sesión así lo indica, mostramos un mensaje y un enlace para finalizar la sesión actual y si no, mostramos el formulario inicial:

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

El código para terminar la sesión es el siguiente:

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


Hasta aquí hemos desarrollado nuestra primera aplicación con la utilización de sesiones. Esta técnica es de uso común en sistemas de cualquier alcance, por lo cual nos 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 máquina de un usuario. Éste las recibe sólo 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 sólo 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);

?>
```

USOS

El uso de sesiones no está destinado a todas las situaciones, sino sólo a aquellas que requieran mantener valores de variables a lo largo de la navegación de un usuario por las páginas de nuestro sitio, por ejemplo, como vimos en el acceso a contenidos restringidos.

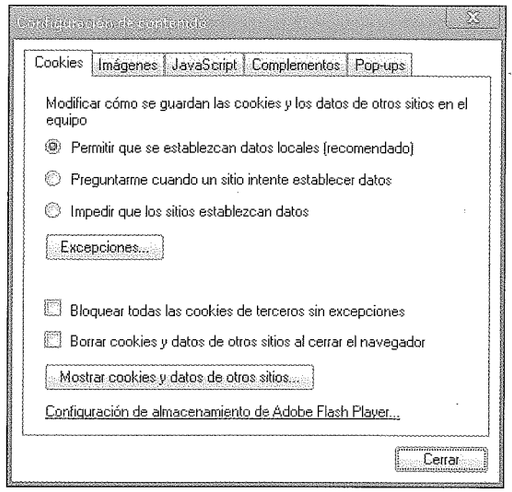

***Figura 6.** Los navegadores actuales permiten al usuario configurar la posibilidad de habilitar el recibimiento de cookies o restringirlo.*

Con este código almacenaremos (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 visualiza de la siguiente manera:

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

INFORMACIÓN SENSIBLE

Cuando almacenamos en cookies cierta información importante a la que preferiblemente no deberían acceder terceros (por ejemplo, contraseñas), existe la posibilidad de aplicar alguno de los métodos de encriptación disponibles. Desarrollaremos más información al respecto en la sección destinada al manejo de cadenas de caracteres.

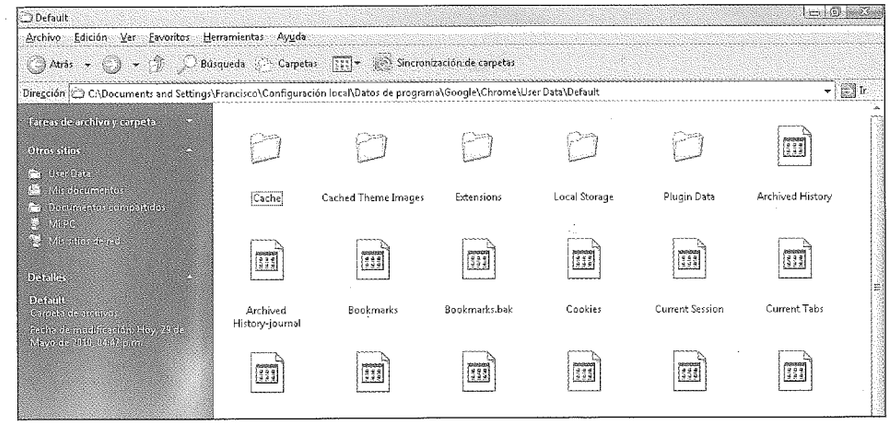

***Figura 7.** Las cookies se almacenan en la máquina del usuario que accede a las páginas del sitio que las implementa.*

En los siguientes accesos del usuario al sitio (hasta los 30 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'];

?>
```

Éste es sólo un caso, entre tantos, de lo que podemos lograr a través de la utilización de cookies en nuestras aplicaciones web.

Funciones del sistema de archivos

En una aplicación web es usual interactuar con archivos y directorios, por ejemplo: galerías de imágenes, videos subidos por los usuarios, generación de gráficos para descarga, todas ellas actividades muy frecuentes.

A continuación, haremos un repaso por algunas de las funciones más importantes que PHP nos brinda para manipular archivos y carpetas desde scripts incluidos dentro de nuestras páginas de Internet.

Hay una directiva en el archivo de configuración **php.INI** que nos será de ayuda al momento de interactuar con archivos remotos, es decir, aquellos que se encuentren en otros servidores, como por ejemplo en un servidor de hosting. Se trata de **allow_url_fopen**, que puede estar habilitada (**On**) o deshabilitada (**Off**).

```
allow_url_fopen = On
```

Veamos como primera medida las funciones habituales para abrir, leer y cerrar un archivo. La función **fopen** recibe dos argumentos principales: el nombre del archivo (incluida la ruta hacia él) y el modo de apertura.

```
<?php

$fd = fopen('archivos/documento.txt', 'r');

?>
```

El segundo argumento nos permite indicar cuál será el tipo de acceso al archivo: sólo lectura, lectura y escritura, etcétera. Observemos cuáles son las posibilidades:

MODO	DESCRIPCIÓN
r	Apertura para sólo lectura.
r+	Apertura para lectura y escritura.
w	Apertura sólo para escritura.
w+	Apertura para lectura y escritura. Si el archivo existe elimina su contenido y si no existe intenta crearlo.
a	Apertura sólo para escritura.
a+	Apertura para lectura y escritura. Si el archivo existe no elimina su contenido (el contenido que luego ingresemos se agrega al final) y si no existe intenta crearlo.
x	Creación y apertura sólo para escritura. Si el archivo existe genera un error y si no existe intenta crearlo.
x+	Creación y apertura para lectura y escritura. Si el archivo existe genera un error y si no existe intenta crearlo.

Tabla 1. Modos de apertura disponibles para fopen.

El archivo **php.INI** mantiene una directiva denominada **include_path**, que es utilizada cuando se incluyen o se abren archivos y/o directorios. En el caso de **fopen**, si intentamos abrir un archivo y no indicamos la ruta, hacia el.

```
<?php

$fd = fopen('documento.txt', 'r');

?>
```

PHP intentará buscarlo en los directorios definidos en esta directiva:


```
include_path = ".;C:\wamp\php\pear"
```

Con el propósito de que **fopen** utilice esta funcionalidad, deberemos ingresar un tercer argumento al asignar el valor **true** ó 1, esto lo podemos ver ejemplificado en las siguientes líneas de código:

```
<?php

$fd = fopen('documento.txt', 'r', true);

?>
```

Esta función devuelve lo que se llama un **descriptor de archivo**, que nos será de utilidad al intentar realizar otras operaciones sobre éste, por ejemplo, leer su contenido. Luego de finalizar el trabajo con el archivo, para cerrarlo utilizamos la función **fclose** de la siguiente manera (ver línea de código a continuación):

```
fclose($fd);
```

Hay varias formas de recuperar la composición de un archivo y una de ellas es por medio de la función **fread**, que recibe como argumentos un descriptor y la longitud, en bytes (ver ejemplo a continuación).

```
<?php

$fd = fopen('archivos/documento.txt', 'r');

while (!feof($fd)) {
```

RUTAS

En caso de que estuviéramos trabajando en plataformas Windows, deberíamos escapar las barras invertidas utilizadas en la ruta al archivo. Para lograr esto podemos utilizar la función **addslashes**, que veremos más adelante en este capítulo, en la sección relativa a las cadenas de caracteres.

```
  $contenido .= fread($fd, 1024);
}

fclose($fd);

echo $contenido;

?>
```

Esta función suele utilizarse para trabajar con archivos binarios. En el ejemplo leemos el contenido del archivo y lo asignamos a una variable. La función **feof** devuelve verdadero cuando se alcanza el final del archivo.
Por su parte, **fgets** nos permite leer de a una línea el contenido de un archivo. Recibe como argumento un descriptor, lo podemos ver a continuación:

```
<?php

$fd = fopen('archivos/documento.txt', 'r');

while (!feof($fd)) {
  $c++;
  echo "<br /> Linea $c: ".fgets($fd);
}

fclose($fd);

?>
```

Esta función también recibe de manera opcional el número de bytes a recuperar (como segundo argumento por defecto 1024 bytes).

PEAR

El repositorio de clases oficial de PHP, Pear, incorpora un paquete con implementaciones para utilizar funciones disponibles en versiones recientes del lenguaje (tal es el caso de **file_get_contents**) en versiones anteriores. Podemos obtener más información en **http://pear.php.net/**.

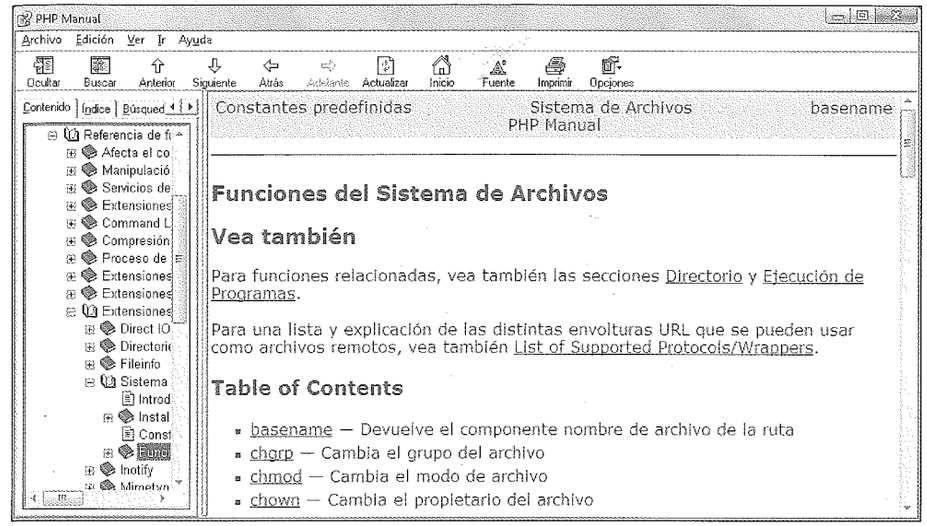

Figura 8. *Las funciones para acceder al sistema de archivos a través de PHP son mejoradas y ampliadas ante cada versión nueva.*

Para realizar funciones similares a las anteriores, contamos con dos funciones: **file_get_contents** y **file**. Ambas reciben como argumento la ruta al archivo y devuelven su contenido sin necesidad de utilizar **fopen**. La diferencia es que **file_get_contents** devuelve el contenido completo, mientras que **file** devuelve un array que contendrá una posición por cada línea del archivo.

```
<?php

echo file_get_contents('archivos/documento.txt');

?>
```

Puede que la función **file_get_contents** no esté disponible en todas las configuraciones.

```
<?php

echo '<pre>';
print_r(file('archivos/documento.txt'));
echo '</pre>';

?>
```

Así mismo, la función **readfile** retorna el número de bytes de un archivo e imprime su contenido, como vemos en el siguiente ejemplo:

```
<?php

readfile('archivos/documento.txt');

?>
```

Pasemos ahora a cómo escribir contenidos en un archivo por medio de la función **fwrite**, que recibe como argumentos un descriptor y la cadena de caracteres a ingresar:

```
<?php

$fd = fopen('archivos/documento.txt', 'a+');

$cadena = "Este es el texto a ingresar - Son las ".date("H:i:s")."\n";

fwrite($fd, $cadena);

fclose($fd);

?>
```

Notemos que, hacia el final de la cadena, incluimos un **\n**, lo que nos permite realizar un retorno y avanzar a la siguiente línea. Nuestro archivo, en el caso de que no tuviera ningún contenido, quedaría constituído de la siguiente manera:

```
Este es el texto a ingresar - Son las 17:56:51
```

PERMISOS

Hay algunas funciones que trabajan sobre los permisos de un archivo: la función **chgrp** nos permite cambiar el grupo de un archivo, la función **chmod** nos da la posibilidad de cambiar los permisos de un archivo, y la función **chown** nos habilita a cambiar al propietario de un archivo.


```
Este es el texto a ingresar - Son las 17:56:52
Este es el texto a ingresar - Son las 17:56:53
Este es el texto a ingresar - Son las 17:56:55
```

Una función similar es **file_put_contents**, que recibe como argumentos el nombre del archivo y los datos a grabar:

```
<?php

$achivo = 'archivos/documento.txt';

$cadena = "Este es el texto a ingresar - Son las ".date("H:i:s")."\n";

file_put_contents($achivo, $cadena);

?>
```

Para agregar al contenido existente del archivo sin truncarlo, podemos utilizar como tercer argumento la constante **FILE_APPEND**:

```
file_put_contents($achivo, $cadena, FILE_APPEND);
```

Si accedemos a un archivo que no ha sido creado por nosotros cabe la posibilidad, por ejemplo, de que no sea apto para lectura o escritura. Para saberlo contamos con dos funciones: **is_readable** e **is_writable** (o su alias **is_writeable**):

```
<?php

$archivo = 'archivos/documento.txt';

if (is_readable($archivo)) {
  //podemos leerlo
} else {
  //no podemos leerlo
}

if (is_writable($archivo)) {
```

```
    //podemos escribirlo
} else {
    //no podemos escribirlo
}

?>
```

Además, podemos obtener otras informaciones acerca del archivo, por ejemplo, a través de la función **stat**, que devuelve un array:

```
<?php

$array = stat('archivos/documento.txt');

echo '<pre>';
print_r($array);
echo '</pre>';

?>
```

Debemos saber que entre los datos más importantes, figuran el tamaño en bytes (posición 7), la hora del último acceso (timestamp, posición 8) y la hora de la última modificación (timestamp, posición 9) .
Otra manera de conocer el tamaño de un archivo es con la función **filesize**, que recibe como argumento la ruta hacia él:

```
<?php

echo filesize('archivos/documento.txt');

?>
```

La función **basename** devuelve el nombre del archivo sin tener en cuenta su ruta. Veamos un ejemplo en la siguientes líneas de código:

```
<?php
```



```
$archivo = 'archivos/documento.txt';
echo basename($archivo); //devuelve 'documento.txt'

?>
```

Con relación a lo anterior, contamos con la función **pathinfo**, que devuelve un array con diferentes informaciones: la ruta al directorio (**dirname**), el nombre del archivo (**basename**), su extensión (**extensión**), y su nombre sin extensión (**filename**). Esto lo podemos ver ejemplificado en las líneas de código que se encuentran a continuación.

```
<?php

$array = pathinfo('archivos/documento.txt');

echo '<pre>';
print_r($array);
echo '</pre>';

?>
```

La salida de lo anterior sería similar a lo que mostramos a continuación:

```
Array
(
    [dirname] => archivos
    [basename] => documento.txt
    [extension] => txt
    [filename] => documento
)
```

TRATAMIENTO DE ARCHIVOS

Algunas funciones adicionales para este tema: con **fileatime** obtenemos la hora del último acceso al archivo, con **filemtime**, la hora de su modificación, con **filegroup**, el grupo al que pertenece, con **fileowner**, su dueño, y con **fileperms**, sus permisos.

Antes de ver cómo realizar ciertas operaciones como renombrar archivos, copiarlos de un lugar a otro, e incluso eliminarlos, hay una función de utilidad que nos permite saber si un archivo realmente existe. Su nombre es **file_exists** y recibe como argumento la ruta al archivo en cuestión:

```
<?php

if (file_exists('archivos/documento.txt')) {
  // el archivo existe
} else {
  // el archivo no existe
}

?>
```

Pasemos ahora a cómo cambiar el nombre de un archivo a través de la función **rename**, que recibe como argumentos el nombre actual y el nuevo:

```
<?php

$nombreActual = 'archivos/documento.txt';
$nombreNuevo = 'archivos/nuevoDocumento.txt';

if (file_exists($nombreActual)) {
  rename($nombreActual, $nombreNuevo);
}

?>
```

Para copiar un archivo de un directorio a otro tenemos la función **copy**, que recibe como argumento la ruta origen y la ruta destino:

```
<?php

$desde = 'archivos/documento.txt';
$hasta = 'documentos/documento.txt';

if (file_exists($desde)) {
```



```
  copy($desde, $hasta);
}

?>
```

En cuanto a eliminar un archivo, la función **unlink** nos permite realizar esta tarea de manera sencilla. Por ejemplo:

```
<?php

$archivo = 'archivos/documento.txt';

if (file_exists($archivo)) {
  unlink($archivo);
}

?>
```

A continuación, expondremos una serie de funciones muy útiles, que refieren al tratamiento o manipulación de archivos subidos a través de los formularios web. Para comprender mejor esto, veamos el siguiente caso:

```
<form enctype="multipart/form-data" action="?" method="post">
Seleccionar archivo: <input name="fldArchivo" type="file">
<input type="submit" value="Enviar formulario">
</form>
```

Luego de seleccionar un archivo de nuestro sistema local y presionar el botón para enviar, desde un script PHP podemos manipular el archivo cargado. Todo lo referente a estos archivos se almacena en un array especial llamado **$_FILES**, desde el cual podemos recuperar ciertas informaciones muy importantes, como por ejemplo las siguientes:

- el nombre original del archivo:

```
$_FILES['fldArchivo']['name']
```

- el tipo de archivo (el tipo MIME, por ejemplo 'image/png'):

```
$_FILES['fldArchivo']['type']
```

- el tamaño del archivo (medido en bytes):

```
$_FILES['fldArchivo']['size']
```

- el nombre temporal dado al archivo utilizado:

```
$_FILES['fldArchivo']['tmp_name']
```

Sobre esto último debemos tener en cuenta que, al cargar un archivo, PHP lo almacena de manera temporal en un directorio especial. Luego, como veremos, podremos copiar el archivo temporal a otra ubicación definida por nosotros.

En todos los casos reemplazamos **fldArchivo** por el nombre que le hayamos dado al campo en el formulario de carga. La función **is_uploaded_file** nos permite saber si el archivo al que hacemos referencia en el argumento fue realmente cargado a través de un formulario. Esta función trabaja de la siguiente manera:

```
<?php

if (count($_POST)) {
  $temp = $_FILES['fldArchivo']['tmp_name'];
  if (is_uploaded_file($temp)) {
    //el archivo fue cargado
  } else {
    //el archivo no fue cargado
  }
```

FTP

PHP mantiene una extensión especialmente dedicada al acceso a directorios y archivos por medio del protocolo FTP. Esto puede llegar a ser un complemento en casos en los que, por algún motivo, los alcances de la extensión para el acceso al sistema de archivos no nos basten.


```
}

?>
```

Figura 9. *La carga de archivos es algo habitual en las aplicaciones web actuales.*

Una vez que verificamos la existencia del archivo, podemos copiarlo a otra ubicación con la función **copy** (ver ejemplo en la siguientes líneas de código).

```
$temp = $_FILES['fldArchivo']['tmp_name'];
$destino = "imagenes/".$_FILES['fldArchivo']['name'];
copy($temp, $destino);
```

Los dos pasos anteriores (verificar que el archivo haya sido cargado y luego copiarlo a otra locación) pueden ser reemplazados por la función **move_uploaded_file**, que trabaja de la siguiente manera:

```
$temp = $_FILES['fldArchivo']['tmp_name'];
$destino = "imagenes/".$_FILES['fldArchivo']['name'];
move_uploaded_file($temp, $destino);
```

Estas funciones nos serán de utilidad y haremos uso de ellas en muchas ocasiones, puesto que el trabajo con archivos subidos por medio de formularios es algo que se presenta en muchos tipos de aplicaciones.

Hasta ahora, observamos cómo utilizar las funciones más importantes relacionadas con el tratamiento de archivos. A continuación, veremos qué funcionalidades nos provee PHP para trabajar con directorios.
Comencemos con la función **dirname**, que nos permite conocer el directorio al que pertenece un archivo, pasándole como parámetro la ruta completa:

```
<?php

$archivo = 'archivos/documento.txt';
echo dirname($archivo);

?>
```

La función **getcwd** nos devuelve el directorio de trabajo actual:

```
<?php

echo getcwd();

?>
```

Así como vimos los descriptores de archivos, también contamos con descriptores de directorios. La función **opendir** nos permite obtener uno de ellos, que luego utilizaremos como argumento en otras funciones:

```
<?php

$dd = opendir("archivos");

//...

closedir($dd);

?>
```

Con **closedir** cerramos el descriptor abierto. Para leer el contenido de un directorio contamos con la función **readdir.** Prestemos atención a la forma en que utilizamos esta función en el ejemplo que sigue:


```
<?php

$dd = opendir("archivos");

while (false !== ($archivo = readdir($dd))) {
  echo $archivo."<br />";
}

closedir($dd);

?>
```

La salida de lo anterior sería semejante a lo siguiente:

```
.
..
documento1.txt
documento2.txt
documento3.txt
imagenes
```

El punto (.) representa el directorio actual (en realidad, es un enlace al directorio actual) y los dos puntos (..), el directorio anterior que también es un enlace. Al respecto, existen dos funciones que nos permiten saber si una entrada es un archivo (**is_file**) o un directorio (**is_dir**). Presentemos un ejemplo para observar cómo funcionan:

```
<?php

$dd = opendir("archivos");

while (false !== ($archivo = readdir($dd))) {
  $ruta = "archivos/$archivo";

  if (is_file($ruta)) $leyenda = ' (archivo)';
  elseif (is_dir($ruta)) $leyenda = ' (directorio)';
  else $leyenda = ' (otros)';

  echo $archivo.$leyenda."<br />";
```

```
  clearstatcache();
}

closedir($dd);

?>
```

PHP almacena los resultados devueltos por algunas funciones. Utilizando **clearstatcache** no tendrá en cuenta los valores anteriores y obtenendremos resultados actualizados. La salida de lo anterior sería similar a lo que vemos a continuación:

```
. (directorio)
.. (directorio)
documento1.txt (archivo)
documento2.txt (archivo)
documento3.txt (archivo)
imagenes (directorio)
```

Otra opción para recorrer el contenido de un directorio es a través de la clase **dir**, que funciona de este modo:

```
<?php

$dd = dir("archivos");

while (false !== ($archivo = $dd->read())) {
  echo $archivo."<br />";
}

$dd->close();

?>
```

Por su parte, la función **scandir** recibe como argumento una ruta a un directorio y devuelve un array con las entradas encontradas:

```
<?php
```



```
echo '<pre>';
print_r(scandir("archivos"));
echo '</pre>';

?>
```

La salida de lo anterior seria semejante e esto:

```
Array
(
    [0] => .
    [1] => ..
    [2] => documento1.txt
    [3] => documento2.txt
    [4] => documento3.txt
    [5] => imagenes
)
```

Para finalizar, veamos en acción a **mkdir**, que se utiliza para crear un directorio, y a **rmdir**, que se utiliza para eliminarlo. En ambos casos deberemos contar con los permisos suficientes para escribir en el directorio. Al momento de eliminar el directorio, deberá estar vacío:

```
<?php

for ($c=1;$c<4;$c++) {
  mkdir('directorio'.$c);
}

rmdir('directorio2');

?>
```

Funciones para el manejo de fecha y hora

La funciones relativas al tratamiento de fechas y horarios no son demasiadas, pero sólo 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 actuales 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 marcas de tiempo Unix**. Este formato es utilizado por múltiples funciones, ya sea como argumento o como resultado devuelto. Una marca de tiempo Unix (o **timestamp**) es el número de segundos transcurridos entre el 1ro de enero de 1970 y la hora especificada.
Para graficar esta situación, comencemos con la función **mktime**, que devuelve la marca de tiempo actual, lo podemos ver en las líneas de código siguientes:

```
<?php

echo mktime();

?>
```

El resultado dependerá del momento en que ejecutemos el script, pero podría ser similar a esto (ver ejemplo en la siguientes líneas de código).

```
1215823703
```

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

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

Con respecto a este último, tenemos la posibilidad de indicarlo utilizando dos o cuatro dígitos: los valores ubicados entre 0-69 se convierten a 2000-2069 y los que están entre 70-100, a 1970-2000. En caso de utilizar cuatro dígitos, hay que 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, 2008);

?>
```

Lo anterior equivale al 11 de julio de 2008, 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 poder observar esta clase de situaciones.

- el 31 de diciembre de 2008:

```
<?php echo mktime(0, 0, 0, 12, 31, 2008); ?>
```

- si avanzamos un día (31 + 1) ya pasamos al 1ro de enero de 2009:

```
<?php echo mktime(0, 0, 0, 12, 32, 2008); ?>
```

- el primer día de abril de 2009:

```
<?php echo mktime(0, 0, 0, 4, 1, 2009); ?>
```

- el ultimo día de marzo de 2009 (1 - 1, el día anterior al 1/4):

```
<?php echo mktime(0, 0, 0, 4, 0, 2009); ?>
```

- el penúltimo día de marzo de 2009 (1 - 2):

```
<?php echo mktime(0, 0, 0, 4, -1, 2009); ?>
```

- falta 1 segundo para el 1ro de enero de 2010:

```
<?php echo mktime(24, 0, -1, 12, 31, 2009); ?>
```

- falta 1 segundo para el 1ro de enero de 2010:

```
<?php echo mktime(23, 59, 59, 12, 31, 2009); ?>
```

- es el 1ro de enero de 2010:

```
<?php echo mktime(24, 0, 0, 12, 31, 2009); ?>
```

- es el 1ro de enero de 2010:

```
<?php echo mktime(0, 0, 0, 1, 1, 2010); ?>
```

Como vemos, 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, lo podemos ver en el siguiente ejemplo:

```
echo microtime();
```

Es evidente que **timestamp** no es un formato demasiado comprensible para un usuario común que quiere, simplemente, visualizar una fecha. Para estos casos, disponemos de dos funciones que nos permiten dar formato a una fecha/hora preestablecida: se trata de **date** y **strftime**. La primera recibe como argumento principal el formato (más adelante explicaremos cómo construir uno) y, opcional, un **timestamp**. Si llegásemos a omitir el valor de timestamp, se toman por defecto la fecha y la hora actuales, especificadas por el sistema operativo.
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, los presentados a continuación:

CARÁCTER	DESCRIPCIÓN
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 dese 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 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 (ante meridiano) o pm (post meridiano).
A	Devuelve AM (ante meridiano) o PM (post meridiano).
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.

***Tabla 2.** Constantes para generar formatos con la función date.*

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, 2009)); ?>
```

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

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

Hay dos cuestiones recurrentes 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 especial, simplemente se imprime tal cual, por ejemplo:

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

imprime algo semejante a **Cp 22:21:38**.

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 salida que se imprime es semejante a lo que sigue:

```
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 cómo se compone un formato. Las opciones son las siguientes:

CARÁCTER	DESCRIPCIÓN
%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).
%D	%m/%d/%y
%e	Día del mes en dos dígitos (completa con espacios).
%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 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 (ante meridiano) o pm (post meridiano).
%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.

Tabla 3. Constantes para generar formatos a través de la función strftime.

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:

```
martes 15 de julio 2008
```

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

```
<?php

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

?>
```

VERSIONES

Un punto importante a la hora de migrar aplicaciones de un servidor a otro es la versión PHP con la que cuentan: algunas funciones sólo están disponibles en las últimas versiones y no en las anteriores. Para saber positivamente en cuál funcionan, podemos visitar el manual oficial de PHP.

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

```
<?php

echo date_default_timezone_get();

date_default_timezone_set('Europe/London');

echo date_default_timezone_get();

?>
```

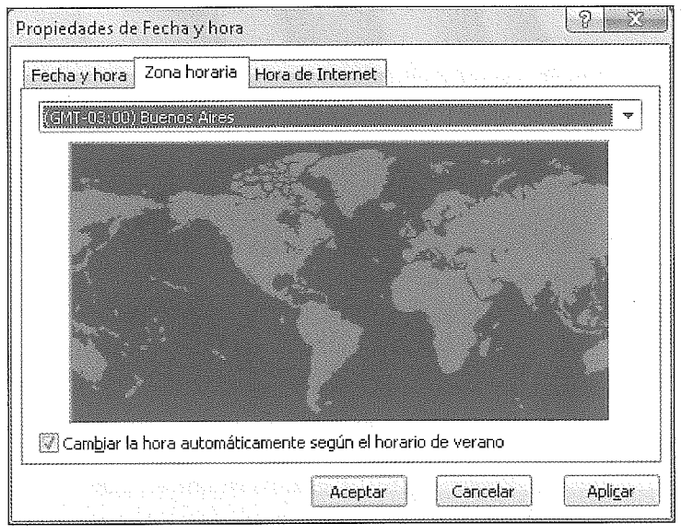

Figura 10. *PHP mantiene funciones específicas para trabajar con zonas horarias puntuales.*

OPCIONES

Si bien podemos conseguir los mismos datos a través de otras funciones ya vistas anteriormente, la función **getdate** nos permite obtener información de fecha/hora actual, mientras que con **gettimeofday** obtenemos, solamente, la hora actual.

Por último, veremos a continuación una función realmente importante, llamada **checkdate**, que recibe como argumentos tres enteros (los cuales son correspondientes al mes, día, y año, en ese orden, formato gregoriano) y devuelve verdadero si la fecha es válida, y falso en caso contrario.

```
<?php

if (checkdate(2, 29, 2008)) {
  echo 'Fecha valida';
} else {
  echo 'Fecha no valida';
}

?>
```

Los puntos que tiene en cuenta esta función 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.

Funciones para el 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 tratamiento de cadenas de caracteres o strings. En los siguientes apartados veremos cuáles son y cómo podemos incluirlas en nuestros scripts.

Comencemos por observar aquellas funciones que cumplen una tarea simple, pero 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 incluir saltos de línea de manera explícita si incluimos un **\n**, o simplemente mientras generamos el valor. Tomemos los siguientes ejemplos para observar concretamente 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 sólo comprende los tags **br** (**
). PHP nos provee una función para anteponer a cada salto de línea encontrado, una etiqueta de este tipo. Se trata de la función **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, más conocidos como caracteres de escape (como por ejemplo, los **\n**), tienen sentido, siempre y cuando los incluyamos entre comillas dobles. Esto vale para cualquier situación, no sólo para ser utilizados con la función **echo**. También podemos incluir el contenido de variables dentro de una cadena, como vemos en el ejemplo:

```
<?php

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

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

?>
```

La sentencia **echo** puede recibir más de un parámetro y tienen que estar separados por comas y/o puntos. Como salida, los concatenará y los mostrará así:

```
<?php

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

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

?>
```


En **php.ini** contamos con la directiva **short_open_tag**, que permite utilizar una sintaxis especial para ejecutar la funcionalidad **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";

?>
```

printf y **sprintf** permiten formatear una cadena de caracteres. La primera devuelve la longitud de la cadena, y la segunda la cadena generada. La idea en ambos casos es definir el tipo de dato que se va a utilizar para mostrar los argumentos.

```
<?php

$mes = 'Julio';
$dia = 19;
$anio = 2008;

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

?>
```

Notemos que dentro del formato establecido (primer argumento a **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 éstos. 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, entre las cuales podemos citar las siguientes:

- **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 = 2008;
```



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

?>
```

La función **fprintf** es similar a **print**, sólo 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 como **printf** nos devolvía la longitud de la cadena pasada como argumento. En caso de querer saber 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.

La función **str_word_count** nos brinda información acerca de las palabras utilizadas en la cadena. Recibe tres argumentos, los últimos dos opcionales: la cadena a 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 a analizar y un modo, que puede ser uno de los siguientes:

- 0, un array cuyas claves serán el número de bytes (de 0 a 255) correspondiente al carácter y a la frecuencia de éste 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, excepto que 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.

Veamos un ejemplo de utilización de esta curiosa función:

```
<?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 a devolver es opcional):

```
<?php

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


Por su parte, **substr_replace** nos da la posibilidad de 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 a 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** realiza una tarea similar y recibe como argumentos la cadena principal, la cadena a 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, sin diferenciar entre mayúsculas y minúsculas. Estas funciones devuelven la aparición de la primera ocurrencia y para encontrar la última aparición podríamos utilizar las funciones **strrpos** y **strripos**.
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):

DISPONIBILIDAD

Recuerde que podrá obtener los códigos completos de éste y todos los capítulos del libro en el sitio oficial de **OnWeb**. Allí podrá, también, encontrar información adicional acerca de los temas tratados, lo que le permitirá complementar de manera específica cada punto.


```
<?php

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


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

?>
```

La función **strtr** es similar, sólo que con ella podemos reemplazar caracteres específicos por otros, lo podemos ver ejemplificado, a continuación:

```
<?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 caracter de la primera palabra a mayúsculas (**ucfirst**) o el primer caracter de todas las palabras (**ucwords**).

```
<?php

$cadena = 'the Long blondes';

echo strtolower($cadena);

echo '<br />';

echo strtoupper($cadena);

echo '<br />';

echo ucfirst($cadena);
```

```
echo '<br />';

echo ucwords($cadena);

?>
```

Como podemos ver en el código siguiente, la salida de lo anterior sería:

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

La función **trim** elimina determinados caracteres encontrados, tanto al principio como al final de la cadena (primer argumento), y 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 a 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, podemos verlo ejemplificado a continuación:


```
<?php

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

?>
```

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

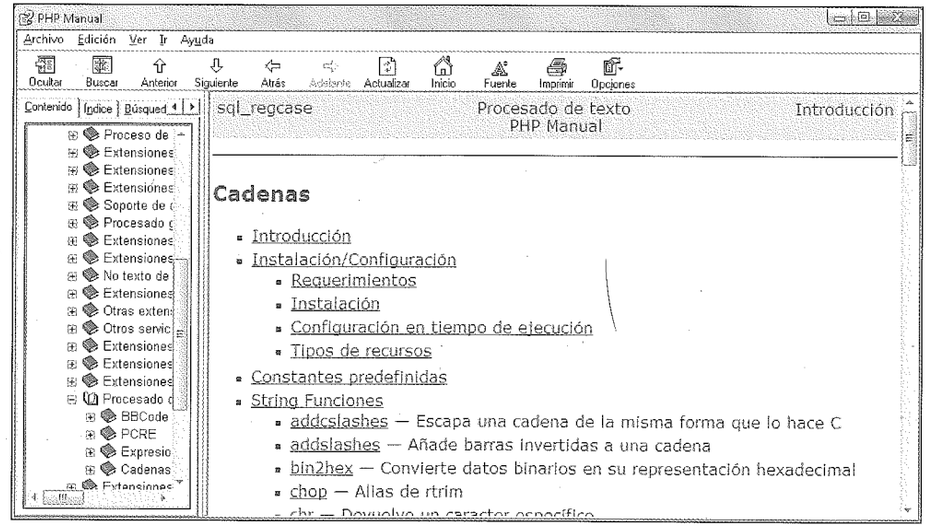

Figura 11. *Las funciones para el tratamiento de cadenas son de las más populares y, sin lugar a dudas, uno de los puntos fuertes del lenguaje.*

La función **str_split** convierte una cadena a una matriz (podemos encontrar más información sobre los arrays en este capítulo). 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, sólo 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), ver líneas de código siguientes.

```
<?php

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

/*
Ejem
```



```
plo dewordw
   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 cantidad de veces a través de 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), podemos verlo ejemplificado a continuación:

```
<?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 (desarrollaremos más información acerca de esto en el **capítulo 7**), las cadenas deben escaparse previamente. Esto significa que a algunos caracteres, por ejemplo, comillas simples, comillas dobles, y 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 un 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 a 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 su opuesta **htmlentities** que vimos en la líneas de código anterior.
La función **htmlspecialchars** es idéntica a **htmlentities** con la diferencia de 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 **>**). Recibe los mismos argumentos que **htmlentities**. 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 como existe la función **html_entity_decode**, también disponemos de **htmlspecialchars_decode**, que convierte desde entidades, pasadas como parámetros, 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 son 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 a la mencionada anteriormente.

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 su utilización en nuestros scripts. Esto es algo fundamental de comprender para el desarrollo de sistemas.
Comencemos por ver a 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.
Entre las diversas funciones geométricas que encontramos en el lenguaje PHP podemos citar las descriptas en la tabla 3:

FUNCIÓN	DESCRIPCIÓN
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.

Tabla 3. Funciones geométricas disponibles en PHP.

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

```
<?php

echo rad2deg(deg2rad(90));

?>
```

Las dos funciones anteriores devuelven 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 disponemos de 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

?>
```

Con relación a esto, 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

?>
```

Para redondear valores con números decimales, contamos con tres funciones: **round**, que recibe el 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

?>
```

* MÁS FUNCIONES

La función **similar_text** nos permite calcular la similitud entre dos cadenas de caracteres dadas como argumento, mientras que la función **soundex** nos devuelve la clave soundex de una cadena de la cadena pasada como argumento (las palabras que se pronuncian de forma parecida tienen la misma clave).

Dentro de las funciones de conversión entre distintas bases numéricas que podemos encontrar en PHP citemos, las siguientes:

FUNCIÓN	DESCRIPCIÓN
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.

Tabla 4. Funciones de conversión entre bases disponibles en PHP.

Con respecto a esto, tenemos la función **base_convert**, que recibe como argumento un número, una base origen (acepta valores entre 2 y 36 inclusive), y una base destino (acepta valores 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 (esto depende de cada plataforma y podemos conocerlo si utilizamos la función **getrandmax**).

```
<?php

echo rand(1, 5);

?>
```

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 según los desarrolladores de PHP 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.

Funciones para tratamiento de matrices

Los arrays son, quizá, la estructura de datos más comúnmente utilizada en PHP, como en la mayoría de los lenguajes de programación y scripts, tanto por su versatilidad como por la gran cantidad de funciones que brinda.
La función inicial es **array** y permite crear una matriz:

```
<?php

$días = array('Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes');

?>
```

Lo anterior es equivalente al siguiente código:

```
<?php

$días[] = 'Lunes';
$días[] = 'Martes';
$días[] = 'Miercoles';
$días[] = 'Jueves';
$días[] = 'Viernes';

?>
```

En el ejemplo visto, a la posición del primer elemento le corresponde el valor **0**, a la del segundo el **1**, y así sucesivamente. Durante la creación del array, podemos definir las claves de cada elemento de esta manera:

```
<?php

$días = array('1' => 'Lunes', '2' => 'Martes', '3' => 'Miercoles', '4' =>
    'Jueves', '5' => 'Viernes');

?>
```

Lo anterior es equivalente al siguiente código:

```
<?php

$días['1'] = 'Lunes';
$días['2'] = 'Martes';
$días['3'] = 'Miercoles';
$días['4'] = 'Jueves';
```



```
$días['5'] = 'Viernes';

?>
```

La función **compact** nos permite crear una matriz de valores a partir de variables previamente definidas. Observemos el ejemplo:

```
<?php

$dia1 = 'Lunes';
$dia2 = 'Martes';
$dia3 = 'Miercoles';

$dia4 = 'Jueves';
$dia5 = 'Viernes';

$día[] = 'dia4';
$día[] = 'dia5';

$días = compact('dia1', 'dia2', 'dia3', $día);

?>
```

La función recibe argumentos y por cada uno de ellos busca una variable del mismo nombre. Si la encuentra, toma su valor y lo incluye dentro del array resultante. La matriz generada en el código anterior es:

```
Array
(
    [dia1] => Lunes
    [dia2] => Martes
    [dia3] => Miercoles
    [dia4] => Jueves
    [dia5] => Viernes
)
```

Otra forma de crear arrays es con la función **explode**, que lo hace a partir de una cadena de caracteres. Recibe como argumentos un delimitador y una cadena:

```
<?php

$cadena = 'Hoy es Domingo 20 de Julio de 2008';
$array = explode(' ', $cadena);

/*

Array
(
    [0] => Hoy
    [1] => es
    [2] => Domingo
    [3] => 20
    [4] => de
    [5] => Julio
    [6] => de
    [7] => 2008
)

*/

?>
```

Para realizar la operación inversa, es decir, obtener una cadena a partir de los valores de un array, tenemos la función **implode**:

```
<?php

$cadena = 'Hoy es Domingo 20 de Julio de 2008';

$array = explode(' ', $cadena);

echo implode(' ', $array); // $cadena

?>
```

La función **array_merge** permite combinar dos o más arrays y obtener uno solo como resultado. En caso de que las matrices componentes tengan claves iguales, se tomará como definitiva la perteneciente al último array ingresado como argumento:


```
<?php

$array1 = array('a', 'b', 'c', 'd');
$array2 = array('e', 'f', 'g');
$array3 = array('h');

$array = array_merge($array1, $array2, $array3);

?>
```

Si se trata de array con matrices componentes podemos utilizar la función **array_merge_recursive**, que trabaja recursivamente.
Para realizar la operación inversa a **array_merge** tenemos **array_chunk**, que genera uno o más arrays a partir de una entrada. La función **array_combine** genera un array a partir de dos datos: un array de claves y otro de valores.
Tanto **array_pad** como **array_fill** respectivamente, completan o generan un array con valores hasta llegar a un límite dado:

```
<?php

$array = array_fill(2, 10, 'valor');

/*
Array
(
    [2] => valor
    [3] => valor
    [4] => valor
    [5] => valor
    [6] => valor
    [7] => valor
    [8] => valor
    [9] => valor
    [10] => valor
    [11] => valor
)
*/

?>
```

En general, será necesario conocer y visualizar la estructura interna de un array, y para eso contamos con **print_r**, que puede ser utilizada no únicamente con matrices:

```
<?php

$días = array('1' => 'Lunes', '2' => 'Martes', '3' => 'Miercoles', '4' =>
    'Jueves', '5' => 'Viernes');

echo '<pre>';
print_r($dias);
echo '</pre>';

/* Salida:

Array
(
    [1] => Lunes
    [2] => Martes
    [3] => Miercoles
    [4] => Jueves
    [5] => Viernes
)

*/

?>
```

Utilizamos la etiqueta **pre** para mantener el formato original de la salida dentro del contexto de una página XHTML. Otra función que puede servirnos es **var_dump**:

```
<?php

$días = array('1' => 'Lunes', '2' => 'Martes', '3' => 'Miercoles', '4' =>
    'Jueves', '5' => 'Viernes');
echo '<pre>';
var_dump($días);
echo '</pre>';

?>
```


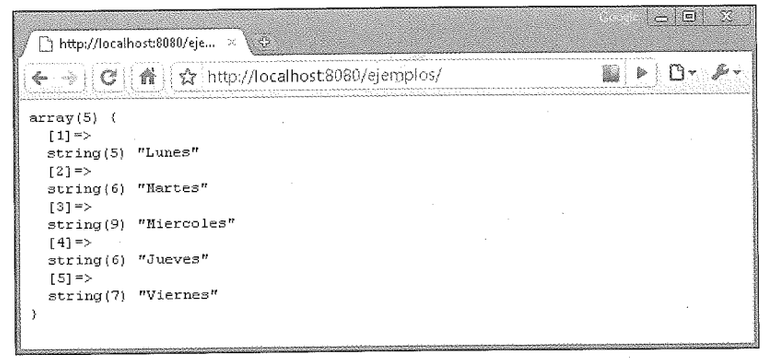

Figura 12. *Las matrices son, quizá, la estructura de datos más popular dentro de PHP.*

La función **count** nos permite obtener el número de elementos de una matriz. La forma en que utilizamos esta función es la que mostramos a continuación:

```
<?php

$m = array('Yeah Yeah Yeahs', 'The Long Blondes');

echo count($m); //2

?>
```

Opcionalmente, recibe un segundo argumento que, en caso de valer **1**, cuenta los elementos internos de la matriz. Recordemos que un array puede contener elementos de cualquier tipo, inclusive otros arrays.
Con **array_count_values** recuperamos un nuevo array que tendrá como clave los valores de la matriz original y como valores las ocurrencias de cada una de ellas.
La función **array_push** nos permite agregar elementos (segundo argumento, puede ser una variable o un array) al final de una matriz previamente declarada, primer argumento. Para añadir al principio contamos con la función **array_unshift**.
Podemos extraer elementos de una matriz con **array_shift** (el primero) o con **array_pop** (el último). Ambas funciones devuelven el elemento quitado y modifican la matriz original. Con **array_slice** extraemos porciones de una matriz y con **array_splice** la sustituimos por otros valores.
Una matriz esta compuesta por pares clave/valor. Para obtener un array con todas las claves, tenemos la función **array_keys**, y para recuperar un array con todos los valores, tenemos la función **array_values**.
Para saber si un valor o una clave existen dentro de un array, contamos con las funciones **in_array** y **array_key_exists** respectivamente. La primera recibe como

argumentos un valor y un array, mientras que la segunda recibe una clave y un array. Si se encontraron coincidencias devuelven verdadero y, en caso contrario, falso.
La función **array_search** busca un valor determinado en una matriz, y si el valor es encontrado en ésta, devolverá su clave o posición.
Una función importante es **array_unique**, que nos permite eliminar los valores duplicados encontrados en la matriz pasada como argumento. No afecta al array original, sino que devuelve uno ya limpio. Hay múltiples funciones que nos permiten ordenar un array según un criterio u otro. Comencemos por ver a la función **sort**:

```
<?php

$valores[] = '1';
$valores[] = 9;
$valores[] = '10';
$valores[] = '2';

sort($valores, SORT_NUMERIC);
/*

Array
(
    [0] => 1
    [1] => 2
    [2] => 9
    [3] => 10
)
*/

?>
```

El segundo argumento indica el tipo de ordenamiento: **SORT_NUMERIC** trata los valores como numéricos, **SORT_STRING** los toma como cadenas de caracteres y **SORT_REGULAR** ordena normalmente sin cambiar los tipos de datos originales. Esta función no mantiene la relación entre valores e índices, como observamos en el ejemplo anterior. En caso de que quisiéramos mantenerla, podríamos utilizar la función **asort**:

```
<?php

$valores[] = '1';
```



```
$valores[] = 9;
$valores[] = '10';
$valores[] = '2';

asort($valores, SORT_NUMERIC);
/*

Array
(
    [0] => 1
    [3] => 2
    [1] => 9
    [2] => 10
)
*/

?>
```

Para ordenar una matriz inversamente disponemos de la función **rsort**, y para mantener la correlación original entre claves e índices podemos utilizar **arsort**. Ambas toman los mismos argumentos que **sort**.
Las funciones vistas hasta ahora nos permiten ordenar según distintos criterios, siempre a partir de los valores de cada posición de un array. En caso de que necesitemos ordenar por clave, existen la funciones **ksort** y **krsort** para orden inverso.
Una función adicional para ordenar arrays es **natsort**. Esta función aplica el llamado algoritmo de orden natural que, básicamente, intenta ordenar de acuerdo con la lógica que aplicaría un ser humano. La función **natcasesort** es similar, excepto que no diferencia mayúsculas de minúsculas. Ambas sólo reciben como argumento un array y mantienen la relación entre claves y valores.
La función **array_multisort**, por su parte, nos permite ordenar un array de más de una dimensión, de manera recursiva.
Al momento de recorrer un array podemos utilizar bucles **for** o **foreach**, por ejemplo. Veamos los dos casos, a continuación:

```
<?php

$array = array('a', 'b', 'c', 'd');
foreach ($array as $clave => $valor) {
  echo "Clave $clave : $valor <br />";
```

```
}

?>
```

```
<?php

$array = array('a', 'b', 'c', 'd');

for ($c=0;$c<count($array);$c++) {
  echo "Clave $c : $array[$c] <br />";
}

?>
```

En este sentido, contamos con funciones para movernos dentro de un array, y para ilustrar esto observemos el siguiente ejemplo:

```
<?php

$letras = array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h');

while ($array = current($letras)) {
  echo key($letras).': '.current($letras).'<br />';
  next($letras);
}

?>
```

La función **current** recibe como argumento un array y nos devuelve el valor de la posición actual, pero falso si ya no hay más posiciones. Por medio de **key** obtenemos la clave actual. A través de **next** avanzamos a la siguiente posición.
Otras funciones relativas son **prev**, que retrocede una posición, **reset**, que retrocede a la primera posición, **end**, que avanza hasta la última, y **each**, que devuelve la siguiente posición y avanza a la siguiente.
La función **array_walk** recorre automáticamente cada posición de un array y le aplica una función especificada por el usuario. **array_walk_recursive** realiza la misma tarea, sólo que también toma los arrays interiores, si es que los hay.
La función **shuffle** reordena aletoriamente los elementos de una matriz:


```
<?php

$letras = array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h');

shuffle($letras);

?>
```

Por su parte, **array_rand** recibe un array y devuelve una cantidad (segundo argumento) de posiciones aleatorias. Si es 1, devuelve una cadena con la clave elegida, y si es más de uno, un array con las claves.
Por último veamos **list**, una interesante alternativa que nos permite asignar valores de un array a variables en una sola línea:

```
<?php

$letras = array('a', 'b', 'c');

list($primeraLetra, $segundaLetra, $terceraLetra) = $letras;

echo $primeraLetra,$segundaLetra,$terceraLetra;

?>
```

RESUMEN

En este capítulo, 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 que casos utilizaría arrays? ¿En que 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 cuantos 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 saber si una frase es un anagrama de otra.

5 Genere un script que cuente las visitas de un usuario a cada página.

Capítulo 4

Manejo y control de errores

En este capítulo, conoceremos de qué manera PHP puede ayudarnos, a través de diferentes funciones y configuraciones, a mantener una observación detallada de los errores que pueden encontrarse dentro de un script.

Introducción	**148**
Tipos de errores	148
Etapas de un desarrollo	148
Opciones del lenguaje	**149**
Configuraciones posibles	149
Funciones del lenguaje	158
Excepciones	160
Resumen	**163**
Actividades	**164**

INTRODUCCIÓN

Es indudable que, para lograr aplicaciones robustas y estables, primero deberemos someterlas a numerosas pruebas para verificar su integridad y corregir aquellas falencias que vayamos encontrando durante las distintas revisiones o durante el período de testeo de la aplicación.
A lo largo de este capítulo, haremos un repaso por las principales alternativas provistas por PHP para obtener información adicional acerca de los errores y advertencias surgidos durante la ejecución de un script.
Uno de los aspectos más importantes a la hora de desarrollar sistemas es la posibilidad de detectar y corregir los distintos errores o fallas que puedan llegar a surgir y, para esto, PHP pone a nuestra disposición una serie de herramientas que nos permitirán mantener un control pormenorizado de cada alternativa.

Tipos de errores

Hay dos clases de opciones: por un lado, las funciones predefinidas del lenguaje y, por otro, las distintas configuraciones posibles especificadas en los archivos disponibles a través de la distribución de PHP. Al mismo tiempo, podemos englobar o categorizar los errores posibles en dos grupos fundamentales: los relacionados con el intérprete y los específicos del código que estamos escribiendo. PHP nos provee funcionalidades para ambos casos. Dentro del primer grupo podemos citar, por ejemplo, los que se producen al intentar cargar una determinada extensión, o 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 que atañen a la sintaxis demandada por el lenguaje en sí.

Etapas de un desarrollo

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 creamos 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.

TIPOS DE APLICACIONES Y ERRORES

Cabe destacar que el manejo de errores está orientado a cualquier clase de aplicación, independientemente de su alcance o de su propósito inicial. La robustez de un sistema no tiene que ver, necesariamente, con la cantidad de usuarios que lo utilizan ni con su finalidad.

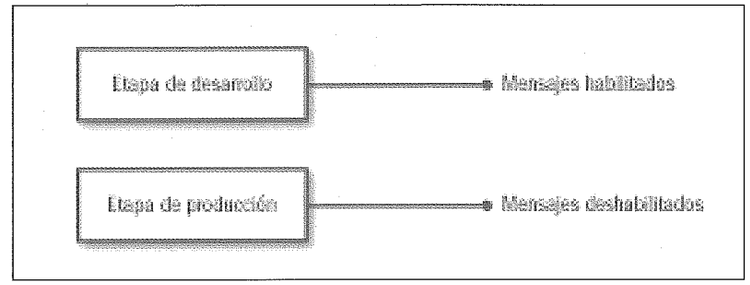

Figura 1. *Según la etapa de desarrollo en la cual nos encontremos, deberemos optar por una política determinada en cuanto a la exhibición de mensajes de error y advertencias.*

OPCIONES DEL LENGUAJE

Dentro de lo que es el lenguaje en sí, PHP tiene una serie de herramientas que nos permitirán aspirar a mantener aplicaciones libres de error a un bajo costo. Esto se debe a la potencia y a la claridad que ya son parte del lenguaje en general y que continúan brindando grandes posibilidades en lo estrictamente relacionado con el control y con la solución rápida de posibles fallas en los sistemas.
Veremos cómo PHP mantiene su filosofía de permitir simplificar los procesos de desarrollo al darle una libertad extrema al programador, pero a la vez, achica 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 situaciones variadas. 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.

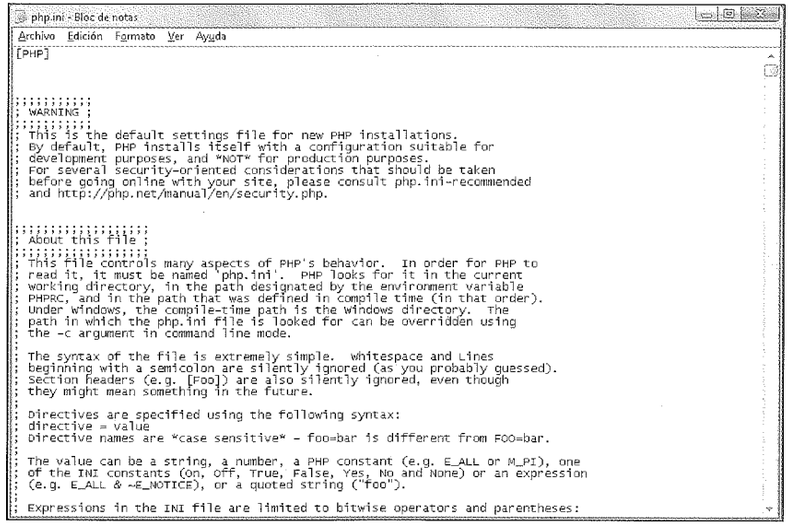

***Figura 2.** El archivo de configuración **php.ini** incorpora opciones generales para modificar el comportamiento de PHP.*

En los servicios de alojamiento (proveedores de hosting), este archivo, en general, no está disponible para ser modificado y/o actualizado por parte de los autores de páginas web, pero, como veremos en breve, por medio de un scipt es posible modificar algunos valores de sus directivas.
Con relación al manejo y el 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 están definidos a través de constantes, entre las cuales podemos citar las siguientes:

CONSTANTE	DESCRIPCIÓN
E_ALL	Nos muestra todos los errores y advertencias, excepto los correspondientes a E_STRICT. En PHP6 se espera que muestre incluso los de E_STRICT.
E_COMPILE_ERROR	Emite mensajes acerca de errores fatales en tiempo de compilación.
E_COMPILE_WARNING	Similar a E_COMPILE_ERROR, sólo que se encarga de mostrar únicamente las advertencias ante eventuales errores.
E_CORE_ERROR	Trata errores fatales producidos durante el inicio de PHP (carga de librerías, directivas, etc).
E_CORE_WARNING	Es similar al anterior, pero se encarga de mostrar únicamente las advertencias ante eventuales errores.
E_ERROR	Muestra mensajes acerca de errores fatales en tiempo de ejecución. Si omitimos esta opción, los errores fatales serán exhibidos de todas formas.
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, sólo que podemos personalizar los mensajes.
E_USER_NOTICE	Es similar a E_NOTICE, sólo que podemos personalizar los mensajes.
E_USER_WARNING	Es similar a E_WARNING, sólo que podemos personalizar los mensajes.
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 con 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, a continuación, 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
```

PHP utiliza una combinación de constantes que nos permite visualizar todos los errores y advertencias (**E_ALL**), excepto aquellas no críticas para el sistema (**E_NOTICE**):

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

La deshabilitación explícita de **E_NOTICE** está dada por el hecho de que generalmente, 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 muy raramente 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 para algunos excesiva, que puede ser restringida a través de este nivel de error.
Ahora 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** ó **1**, para habilitar mensajes, u **Off** ó **0**, para deshabilitar mensajes.

```
display_errors = On
```

El valor por defecto de esta directiva de manera predeterminada es **On**.

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.

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 despliegan este tipo de errores podrían ser la inexistencia de algún directorio especificado en el archivo **php.INI**, o también extensiones habilitadas no halladas.
Puede tomar dos valores: **On** ó **1**, para habilitar mensajes, u **Off** ó **0**, para deshabilitar mensajes. 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 contamos con la fecha, la hora, el nivel, y la descripción del error. Para registrarlos existe la directiva **log_errors**, que por defecto esta 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 con mensajes la salida estándar del navegador. Es especialmente útil durante la etapa de producción de un sitio.

En la instalación clásica de Apache, podremos 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, podrá visualizarse con cualquier editor; para esto, el servidor deberá estar inactivo. Si queremos especificar otro nombre, contamos con 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 el servidor web Apache utiliza el mismo archivo para ubicar y almacenar su propio registro de errores, es decir, aquellos relacionados con las tareas del servidor web.

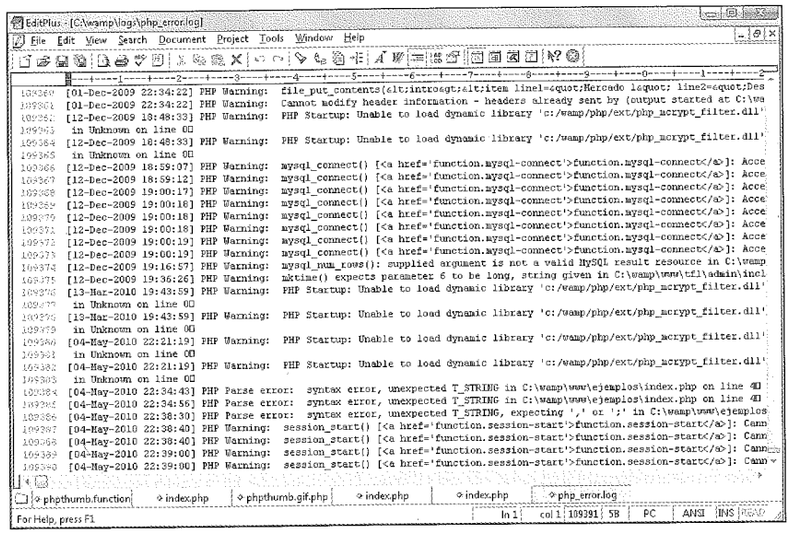

***Figura 3.** El archivo de registro y almacenamiento de errores puede sernos de mucha utilidad a la hora de precisar el momento y el lugar de su ocurrencia.*

Si necesitamos determinar y/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, la ejecución de ciclos **for**, **while**, **do while**, **foreach**, etcétera, un mismo error se producirá un numero N de veces, y PHP puede ayudarnos a mantener cierta claridad al respecto para no mostrar ni almacenar N mensajes sino sólo uno. Con **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: **ignore_repeated_source**, activada, no tiene en cuenta el origen de los errores, el archivo en el que se produjeron.

```
ignore_repeated_source = Off
```


Otra directiva disponible que puede sernos de ayuda es **track_errors**, que permite almacenar el último mensaje de error/advertencia en una variable llamada **php_errormsg**. Está deshabilitada por defecto en el archivo **php.INI**.

```
track_errors = Off
```

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

?>
```

En ocasiones, PHP puede darnos 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 PHP nos muestre un mensaje con un enlace al detalle deberemos tener activada la directiva **html_errors**:

```
html_errors = On
```

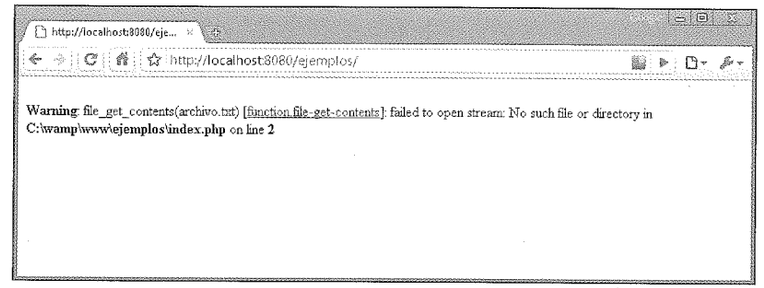

***Figura 4.** PHP proporciona un detalle acerca de las distintas clases de errores a través de enlaces a su manual oficial.*

Puede tomar dos valores: **On** ó **1**, para habilitar mensajes HTML u **Off** ó **0**, para deshabilitar mensajes HTML. El valor que trae por defecto en **php.INI** es **Off** y se recomienda mantenerlo así, al menos durante la etapa de producción.
También podemos 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**), tendremos la oportunidad de referenciarla tomando como base el directorio raíz del servidor:

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

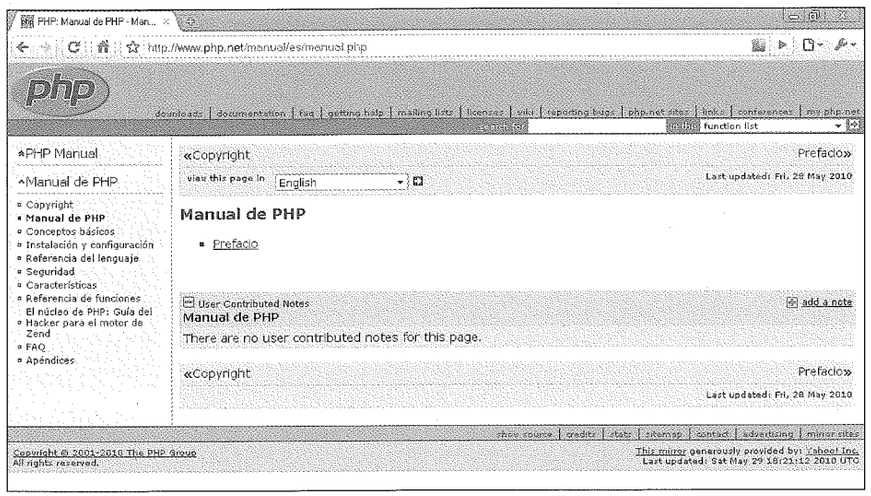

***Figura 5.** El manual de PHP puede ayudarnos a obtener respuestas rápidas ante determinados sucesos relacionados con los errores cometidos en un script.*

En la extensión del manual siempre debe incluirse el punto:

```
docref_ext = .html
```

III MÁS ACERCA DE EXCEPCIONES

PEAR (**http://pear.php.net/**), el repositorio oficial de extensiones y aplicaciones para PHP, ofrece una clase que nos permite implementar excepciones. Funciona para las versiones 5 y superiores, y se denomina **PEAR_Exception**. Estudiaremos con mayor énfasis y detenimiento las características de PEAR en el **Capítulo 9**.

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>"
```

Muchas de las directivas están relacionadas con el informe de errores para que el desarrollador pueda tomar nota y corregirlos de una manera rápida. Esto, que sin dudas es de suma utilidad y 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 mostrar errores por pantalla. Tiene que ver, principalmente, con dos cuestiones: la de preservar la estética de las páginas del sitio y, prevenir exhibición de información que eventualmente pueda ser tomada por usuarios que busquen explotar puntos débiles del sitio, con mala intención o simplemente por curiosidad. Como vimos, la exhibición de mensajes de error puede brindar información sensible, por ejemplo, la ubicación de determinados archivos o, peor aun, la presentación de datos reservados.

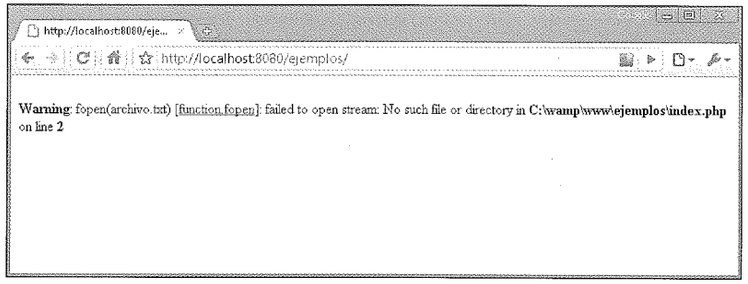

***Figura 6.** Idealmente, a los mensajes de error generados por una aplicación, deberían acceder únicamente aquellas personas que tuvieran los permisos necesarios para poder solucionarlos.*

* MODIFICACIONES EN PHP VERSIÓN 6

A partir de la versión 6 del lenguaje de programación PHP, se producen ciertas modificaciones en lo relacionado con el manejo y el control de errores. Entre ellas podemos citar, además de la inclusión de la constante **E_RECOVERABLE_ERROR**, que el nivel de errores **E_ALL** contendrá el nivel **E_STRICT**.

Funciones del lenguaje

El archivo php.ini admite múltiples directivas para manipular todo lo que atañe a los mensajes de advertencia en caso de surgir errores que no han sido prevenidos 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. Aprenderemos cuáles son y cómo hacer un correcto uso de ellas.

Tomaremos como primer caso **ini_set**, una función que recibe como argumentos el nombre de la directiva del archivo **php.INI** que queramos modificar y el valor correspondiente que deseamos asignarle:

```
ini_set('display_errors', 'On');
```

Al momento de utilizar esta función deberemos tener en cuenta dos cuestiones muy importantes: primero, que los cambios realizados sólo tendrán validez durante la ejecución del script desde el cual se invoque (modificación de valores temporales) y, segundo, que por cuestiones de seguridad no todas las directivas pueden modificarse. Encontraremos más información con respecto al detalle de las directivas permitidas, en el manual de PHP (**www.php.net**).

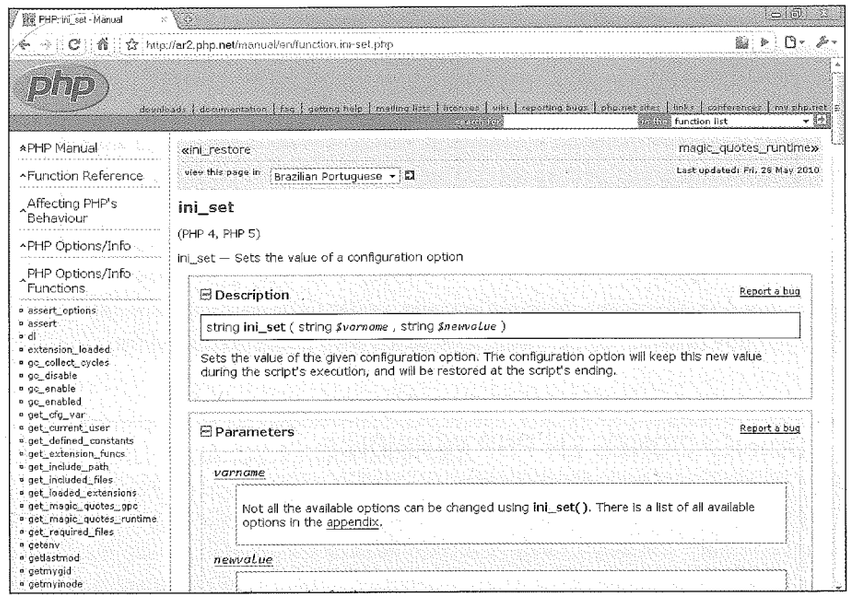

***Figura 7.** La función ini_set puede modificar en tiempo de ejecución los valores de algunas directivas del archivo php.INI.*

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 queramos establecer. Los valores aceptados son:

VALOR	CONSTANTE DE LA DIRECTIVA
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
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 hacer uso de operadores debemos sumar los valores enteros correspondientes a las constantes. Por ejemplo, si quisiéramos hacer uso de las opciones **E_ERROR** y **E_WARNING**:

```
error_reporting(3); //1 + 2
```

Esto sería equivalente a la siguiente línea:

III HOSTING

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

```
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 respecto al script actual, como explicamos al principio de esta sección.

Excepciones

A partir de PHP versión 5 se incluye el soporte para el manejo de excepciones, que podrían llegar a 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.

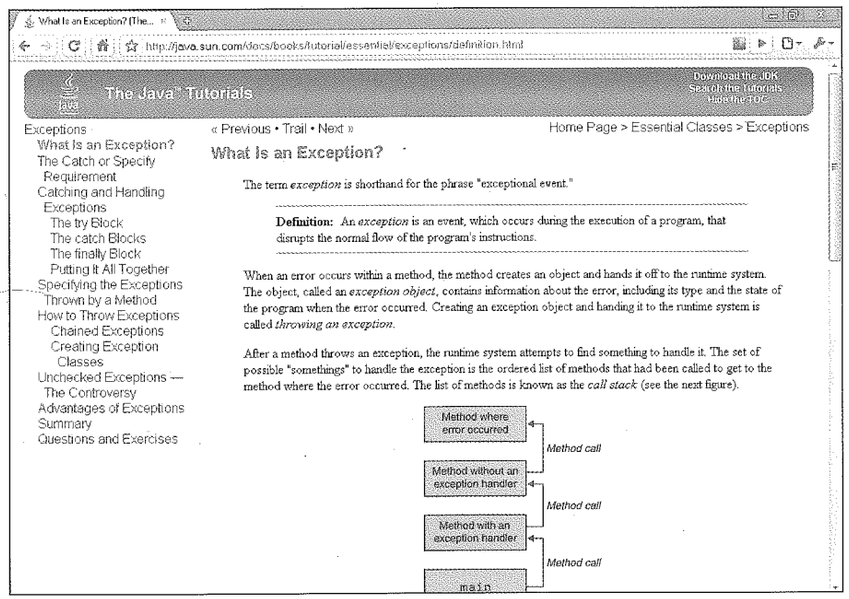

***Figura 8.** El manejo y el control de excepciones es una herramienta relativamente reciente surgida en las últimas versiones del lenguaje.*

En PHP contamos con el bloque **try/catch** para intentar capturar y lanzar excepciones para el control de errores. Las funciones disponibles para el tratamiento y para el control de excepciones son las siguientes:

- **trigger_error** genera un mensaje de error personalizado por parte del usuario. Sólo 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);
}

?>
```

- **set_error_handler** nos permite cambiar el manejador de errores por defecto de PHP por uno personalizado. Tiene relación con la función **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

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.

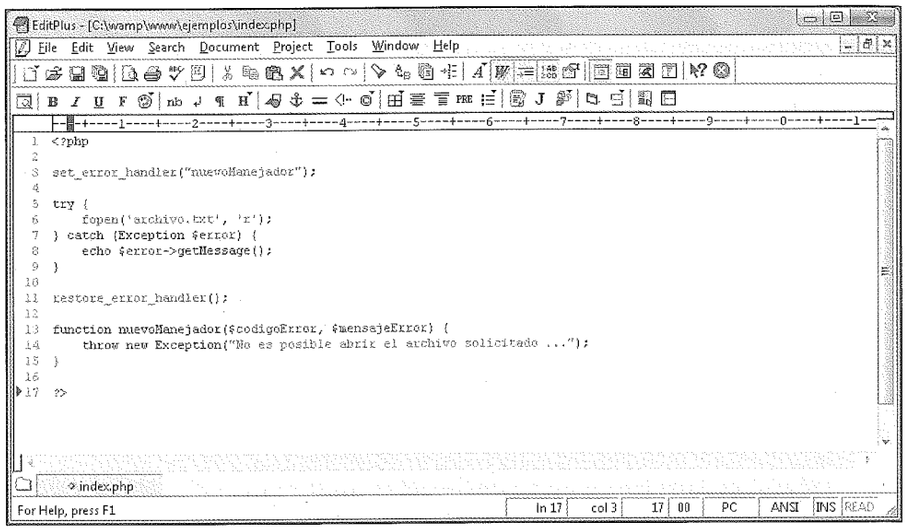

Figura 9. *Las excepciones nos permiten evitar el colapso de una aplicación, al capturar y solucionar los posibles inconvenientes surgidos durante su ejecución.*

Vamos a utilizar el manejador **nuevoManejador** en lugar del de PHP:

```
set_error_handler("nuevoManejador");
```

Esta función crea una excepción que, al lanzarse, evitará 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 la función **getMessage**, contamos con otros métodos entre los que podemos citar como más importantes los siguientes:

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.*

RESUMEN

En este capítulo, observamos cuáles son las principales funciones y directivas de configuración para encontrar y corregir errores para poder mantener seguras y bien formadas nuestras aplicaciones PHP. Hicimos un repaso por las funciones propias del lenguaje y por las directivas disponibles dentro del archivo php.INI. También hemos visto en qué momento nos conviene hacer uso de las directivas configuradas en php.INI, y cómo cambiar el valor de dichas directivas sin modificar el contenido del archivo principal.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Qué interpreta por etapa de desarrollo de un sistema?

2 ¿Qué interpreta por etapa de producción de un sistema?

3 ¿Cuál es la utilidad del archivo de errores generado por PHP?

4 ¿Qué entiende por excepción?

5 ¿En qué casos utilizaría excepciones?

6 ¿Para qué sirve la función error_reporting?

7 ¿Para qué sirve la directiva error_reporting?

8 ¿Cuál es la relación entre las directivas error_reporting y display_errors?

9 ¿Para qué sirve la función ini_set?

10 ¿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.

Capítulo 5

Orientación a objetos

En este capítulo, introduciremos el tema del Paradigma Orientado a Objetos (implementado a través de PHP), una técnica que, utilizada en múltiples desarrollos, será preciso conocer para sacar réditos de las capacidades del lenguaje.

Historia	**166**
Pilares del modelo	166
Implementación en PHP	**167**
Clases, propiedades y métodos	167
Resumen	**183**
Actividades	**184**

HISTORIA

Lo primero que debemos decir es que la orientación a objetos no es una técnica propia de PHP, sino una implementación de un paradigma, sólo una entre tantas: existe una gran cantidad de lenguajes que poseen su propia implementación.
En PHP, a diferencia de lo que ocurre en otros lenguajes, la implementación de la orientación a objetos no fue sólida desde el comienzo, sino que fue moldeándose a medida que fueron surgiendo las nuevas versiones. Recién en la versión 5 del lenguaje se alcanzó el grado de madurez necesaria para que los programadores pudieran aplicar sus conocimientos de la **POO** (Programación Orientada a Objetos) en PHP.

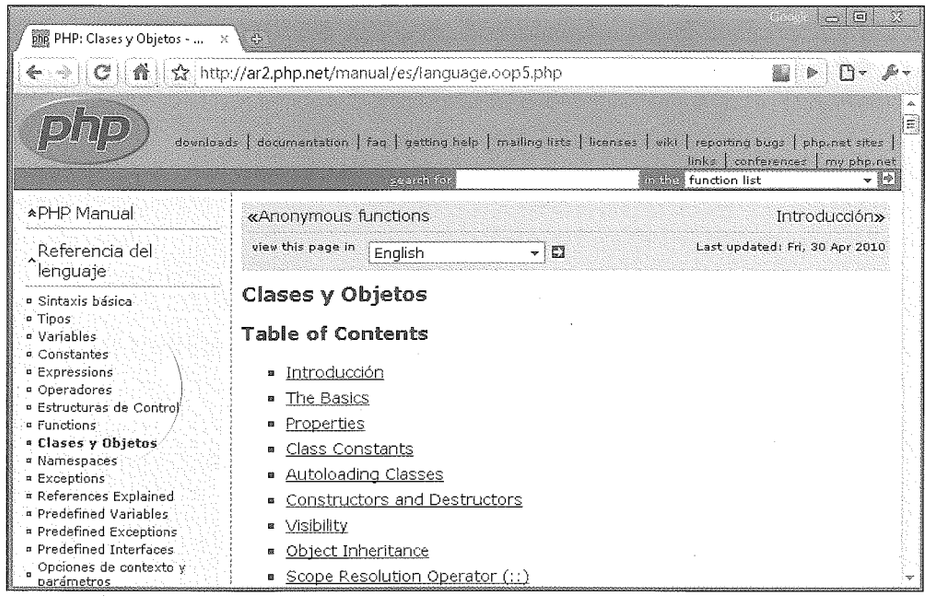

***Figura 1.** La implementación de la Programación Orientada a Objetos ha mejorado de manera notoria en las últimas versiones de PHP.*

Pilares del modelo

Si miramos otros lenguajes con historia en este campo (como por ejemplo, **Java**, **C#**, y **Python**), los desarrolladores a cargo de la ampliación del lenguaje comenzaron a idear y plasmar, poco a poco, la orientación a objetos en PHP: 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 una forma de llevar adelante un desarrollo. Como todo, esto tiene sus ventajas y desventajas, pero es una alternativa a la que

tenemos la libertad de acceder, siempre queda a nuestro criterio, según nuestra inclinación y necesidades, hacer uso o no de éste.
Conocer sus principios básicos nos ayudará a comprender e incluso interactuar con aplicaciones ya desarrolladas que utilizan el Paradigma Orientado a Objetos. Nos referimos a cualquier tipo de aplicación: desde las más simples hasta las más complejas.

***Figura 2.** Java es uno de los lenguajes con historia en lo que atañe a la Programación Orientada a Objetos, y PHP tomó nota de sus virtudes.*

Podemos obtener más información al respecto 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 de este lenguaje de programación.

IMPLEMENTACIÓN EN PHP

Veremos a continuación, cómo implementar la orientación a objetos desde PHP, y tomaremos como base las versiones más recientes del lenguaje.

Clases, propiedades y métodos

Podríamos llegar a definir una clase como una plantilla o molde desde el cual es posible generar objetos, es decir, a partir de un esquema con características bien establecidas, poder 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 cada clase es una abstracción de las instancias particulares y podemos definir sólo una estructura de lugar de N. Cada instancia podrá diferenciarse de las demás con la modificación de las características disponibles en la clase original.
La estructura de una clase contiene dos elementos básicos: variables y funciones (declaradas por medio de la palabra reservada **function**). La definición de una clase se reconoce dentro del código PHP, al iniciar una línea con la palabra reservada **class**. Veamos un ejemplo, a continuación.

```
<?php

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

?>
```

Atributos de clases

Cada clase puede contar con sus propios atributos, por jemplo, un gato puede tener 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 ésta. Veremos más acerca de esto en breve.
Una clase puede contar con métodos. Podemos asumir que un método es una función asociada a una clase y, si seguimos con nuestro ejemplo, diríamos que un gato podría llegar a contar con métodos tales como **calcularFechaNacimiento** u **obtenerNumeroDeHijos**. Para definir un método contamos con esta sintaxis:


```
<?php

class nombreClase {
  public $nombreVariable;

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

?>
```

Un método incluso utiliza la palabra reservada **function**, es decir, la misma que utilizaríamos para definir funciones comunes, fuera del ámbito de una clase.
Por otra parte, observamos una nueva palabra: **$this**, variable reservada especial, que contiene el objeto actual y se utiliza en el ámbito de una función definida dentro de una clase. Puede utilizarse para referenciar otras funciones o variables dentro de la misma clase. Otra opción es utilizar **self**.
En los primeros ejemplos observamos el uso de la palabra **public**. 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:

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

Tabla 1. Modificadores de acceso disponibles.

Hasta antes de la versión 5, 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 para **public**.

```
var $nombrePropiedad;
```

Anteriormente los métodos, por su parte, no contaban con ningún modificador de acceso disponible, únicamente se definían con un nombre y un conjunto de argumentos. Sigamos con el ejemplo que estábamos viendo y repasemos a continuación cómo crear instancias de una clase a través de PHP:

```
<?php

class nombreClase {
  public $nombreVariable;

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

$nombreInstancia = new nombreClase();

?>
```

El operador **new** nos permite crear una instancia de una clase. Como mencionamos, cada instancia de una misma clase puede tomar valores particulares para sus variables:

```
<?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';

?>
```

Veamos la sintaxis a utilizar 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 a partir 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();
?>
```

Podemos notar aquí, que el método **nombreFuncion** devuelve el valor actual de la propiedad **nombreVariable**:

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

el cual mostramos por pantalla:

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

MÓDULOS

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 carga de trabajo: una clase puede ocuparse de una tarea específica y, las demás, de otras.

Permisos de acceso

Pasemos a ver, ahora, cómo funcionan los permisos de acceso y cuál podría llegar a ser la utilidad que nos brinda cada uno de estos, aplicados tanto a propiedades como a métodos dentro de los scripts PHP:

```
<?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 intentado acceder (asignar un valor) a una propiedad protegida. Sin embargo, podemos observar las líneas de código que mostramos, a continuación.

```
<?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 sólo es accesible desde dentro de la clase a la que pertenece. Intente modificar el código anterior para observar el comportamiento y utilice otros modificadores como **public** o **protected**. Analice los resultados.
Los métodos, al igual que cualquier función común y corriente, pueden también recibir 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);

?>
```

¿Cuál piensa que es la salida del código anterior? Ejecute el programa una vez escrito este script para verificar su respuesta.

Constantes

También es posible definir constantes dentro de una clase con la palabra reservada **const**. A diferencia de las propiedades, no se les antepone el carácter $ y siempre son de acceso público, lo podemos ver en el ejemplo que se encuentra a continuación.

```
<?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**. Ahora bien, cada vez que se crea una instancia de una clase se ejecuta una función llamada constructor. El nombre de esta función es **__construct** y podemos redefinirla si deseamos modificar su comportamiento por defecto:

```
<?php

class nombreClase {
  public $nombrePropiedad;

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

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

?>
```

Por motivos de compatibilidad, debemos tener en cuenta que hasta antes de la versión 5, 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**. En el mismo sentido, contamos con los llamados destructores, que son métodos que se encargan de realizar las tareas que se necesitan ejecutar cuando un objeto ya no está siendo referenciado por ninguna variable, es decir, que no cuenta con instancias activas, (esto es el equivalente en otros lenguajes a Garbage Colectors). El nombre de esta función es **__destruct** y podemos redefinirla de la siguiente manera:

```
<?php

class nombreClase {
  public $nombrePropiedad;
```



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

  function __destruct(){
    echo "Terminando ...";
  }
}

$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 ...";
  }
}

$c = new nombreClase();
unset($c);

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

?>
```

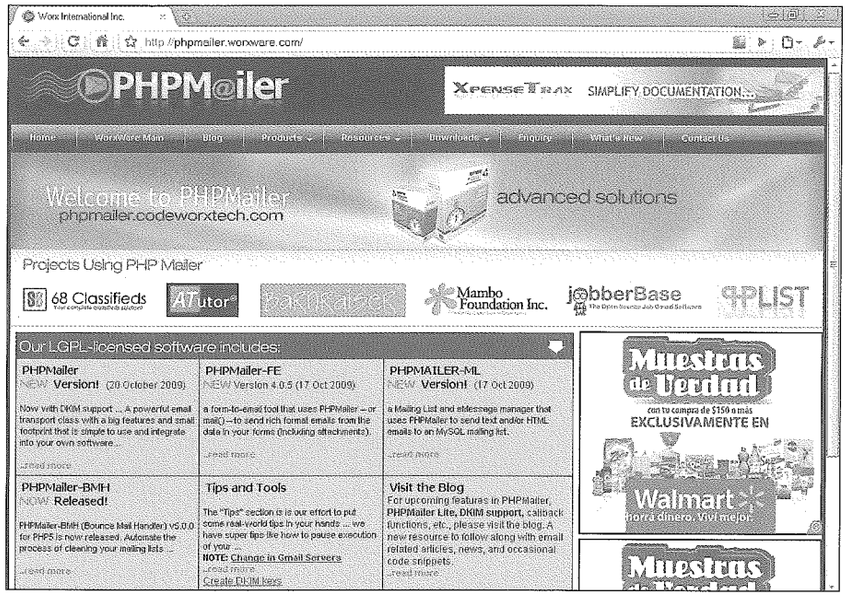

Figura 3. *Hoy en día, cada vez son más las aplicaciones que están desarrolladas con el Paradigma Orientado a Objetos.*

Errores en clases, metodos 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. Recibe como argumentos automáticos el nombre del método y los parámetros pasados, podemos ver un ejemplo, a continuación.

```
<?php

class nombreClase {
  private $nombrePropiedad;

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

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

?>
```


El método **__set** es invocado sólo en caso de intentar asignar valores a propiedades que no han sido 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, podemos verlo en el ejemplo que encuentra a continuación.

```
<?php

class nombreClase {
  private $nombrePropiedad;

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

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

?>
```

En cuanto a las clases, contamos con la función especial **__autoload** que, si es definida de manera explícita, es invocada de forma automática en caso de intentar utilizar clases que no han sido definidas:

```
<?php

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

$c = new claseInexistente();

?>
```

Herencia en PHP

Este es un concepto muy importante en todos los lenguajes que se basan en POO y lo observaremos de manera 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, automáticamente 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 utiliza la palabra reservada **extends**:

```
<?php

class nombreClase1 {
  //...
}

class nombreClase2 extends nombreClase1 {
  //...
}
?>
```

INTERFACES

Esta técnica permite especificar el conjunto de métodos que deberán definir, de manera obligatoria, las clases que la implementarán. Esta opción se introdujo en PHP versión 5 y podemos ponerla en funcionamiento con la palabra reservada **interface**.

La ventaja de esto es que la clase que extiende (**nombreClase2**) hereda y dispone de todos los métodos de la clase base (**nombreClase1**). Incluso puede redefinirlos y modificar su comportamiento. Por supuesto, también puede contar con los suyos propios para extender la propiedades y métodos heredados.
Hay una manera de evitar la redefinición de un método de una clase: a través del modificador **final**. Veamos su implementación:

```
<?php

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

?>
```

Otro modificador de acceso muy útil es **abstract**, que nos permite definir las clases y los métodos como abstractos. Por lo tanto, una clase es considerada abstracta solamente cuando no puede ser instanciada.

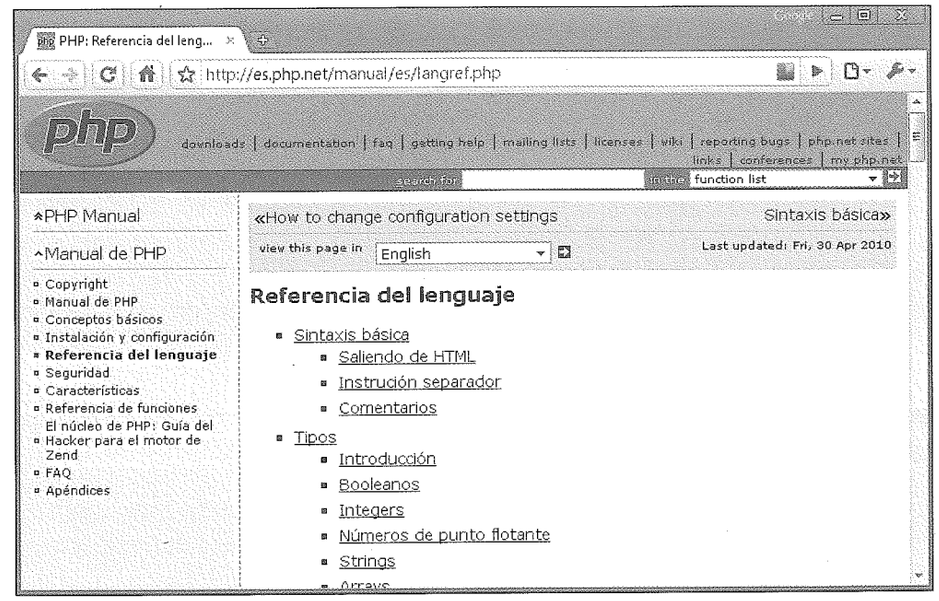

***Figura 4.** En el sitio web http://es.php.net/manual/es/langref.php podemos obtener una ayuda para reforzar conocimientos sobre POO.*

```
<?php

abstract class nombreClase {
  //...
}

?>
```

La finalidad de este tipo de clases es servir como base para otras que sí podrán ser instanciadas (con la utilización de **extends**).
El caso de los métodos abstractos es similar: no se pueden invocar desde una instancia de la clase abstracta, pero pueden ser heredados por las clases que la extienden.

Propiedades y métodos estáticos

Otra opción (además de los modificadores de acceso) 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();
?>
```

CLONACIÓN

En las últimas versiones del lenguaje, 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.

Referenciación de objetos

Tomemos como base el ejemplo que presentamos a continuación, en el cual asignamos un objeto a una variable, y luego esta variable a otras:

```
<?php

$obj1 = new nombreClase();
$obj2 = $obj1;

?>
```

En lo anterior, cualquier operación sobre **obj1** tiene correspondencia en **obj2** y viceversa: el paso de objetos por referencia permite que las modificaciones realizadas tengan repercusión directa sobre el objeto original. Sin embargo, en ocasiones, necesitaremos clonar un objeto para que luego éste pueda mantener un comportamiento propio independiente de los demás. Podemos utilizar la función **clone** que crea un objeto copiando todos los atributos del original:

```
<?php

$obj1 = new nombreClase();
$obj2 = clone $obj1;

?>
```

Comparación de objetos

Para comparar objetos contamos con dos operadores: el operador tradicional de comparación simple (==) y el operador de identidad (===), que nos permitirán realizar distintas evaluaciones acerca del contenido de cada uno. A continuación, podremos conocer una breve descripción de estos operadores:

COMPARACIÓN DE OBJETOS

El operador **==** es muy útil para poder comparar 2 objetos clonados, por ejemplo, si sabemos que tienen clases que se pueden modificar, y que éstas pudieron haber sido cambiadas en cualquier parte de nuestra 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.

- 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, 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);

?>
```

El ejemplo que vimos anteriormente también se aplica a métodos (recordemos que un método no es ni más ni menos que 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.";
}

?>
```

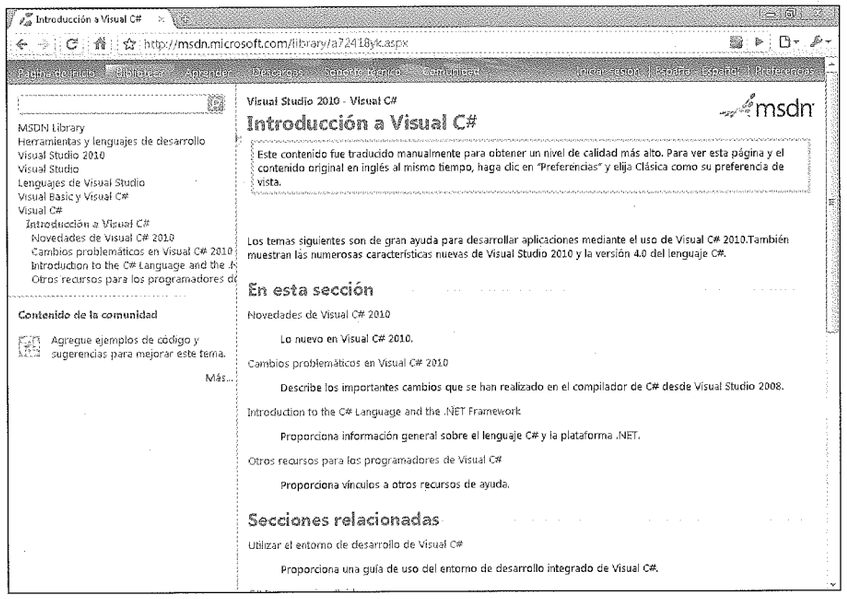

***Figura 5.** El lenguaje de programación C# es uno de los mas populares de la actualidad, y, a diferencia de PHP, es íntegramente orientado a objetos.*

RESUMEN

A lo largo de este capítulo, explicamos las características más salientes en cuanto a la implementación de la Programación Orientada a Objetos en las últimas versiones de PHP. Vimos qué papel juegan GET y SET, la clonación de objetos, y todo lo que hereda por defecto un objeto de su clase principal. También, cómo extender lo heredado en una clase nueva.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Cuál cree que es el motivo de la popularidad de la POO?

2 ¿Qué es una clase?

3 ¿Qué es una propiedad?

4 ¿Qué es un método?

5 ¿Cuál es la diferencia entre una función tradicional y un método?

EJERCICIOS PRÁCTICOS

1 Busque información acerca de la POO en PHP 5.

2 Genere una clase persona con sus características, y luego clónela para poder modificar alguna de ellas.

3 Utilice un método de comparación para obtener las diferencias entre la clase persona y la que usted modificó.

4 Genere una clase para almacenar fechas de cumpleaños y un método para calcular cuantos días faltan.

5 Incorpore las propiedades y los métodos de la clase generada en el ejercicio 4, al ejercicio 3.

PHP 6

Capítulo 6

Imágenes

En este capítulo, haremos un repaso por las opciones relativas al manejo de imágenes a través del lenguaje PHP, y haremos hincapié en las extensiones disponibles y en las herramientas que hacen uso de ellas.

Introducción	**186**
La biblioteca GD	186
Herramientas disponibles	**189**
phpThumb	189
JpGraph	203
Tipos de gráficos soportados	205
Opciones de configuración	222
Resumen	**223**
Actividades	**224**

INTRODUCCIÓN

Debido a que es un lenguaje dinámico, PHP está capacitado para resolver peticiones y actuar en consecuencia a las demandas de los usuarios de las aplicaciones, que de acuerdo con sus necesidades logran dirigir los procesos en distintas direcciones. El trabajo con imágenes no escapa a esta realidad. A continuación, veremos cuáles son las características más salientes de **GD**, la biblioteca más utilizada en la actualidad para resolver esta clase de tareas.

La biblioteca GD

Una de las claves del éxito que ha hecho de PHP un lenguaje tan popular y aceptado por parte de la gran mayoría de los desarrolladores de sitios web, está dada por la inclusión de soporte para la biblioteca **GD**, que permite manipular distintos tipos de imágenes de manera simple y, a la vez, eficaz.

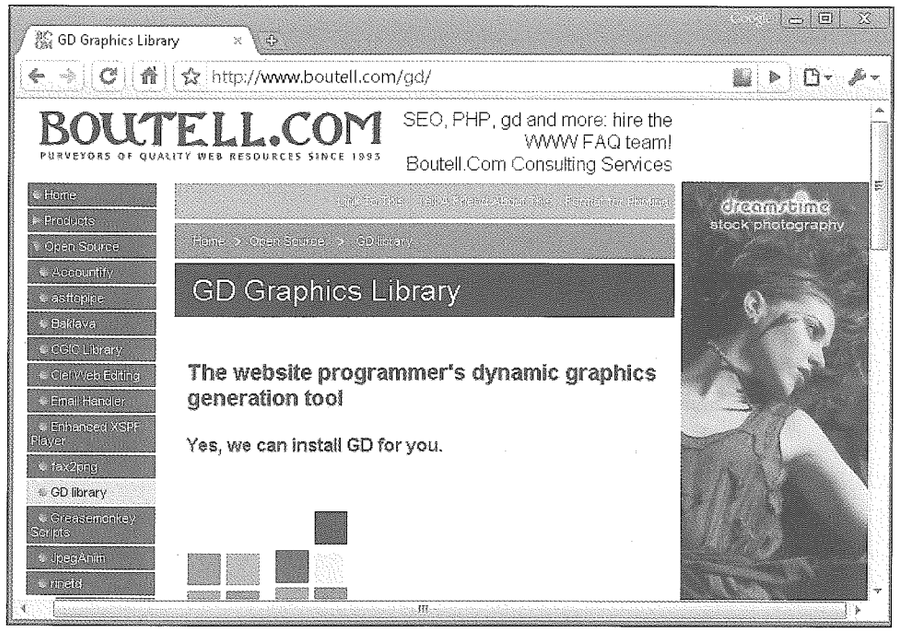

Figura 1. *GD es una de las características disponibles más populares desde el lenguaje de programación PHP.*

En los siguientes apartados veremos cuáles son las características principales de esta biblioteca y, además, algunas de las librerías que utilizan sus servicios para proveer soluciones que nos serán de gran ayuda a la hora de implementar aplicaciones de nivel profesional. La biblioteca GD viene incluida con la distribución estándar

de PHP. Podemos encontrar más información acerca de sus características y funcionalidades en el sitio web oficial, **www.boutell.com/gd**.
La extensión provista por PHP puede ser habilitada desde el archivo de configuración **php.INI** simplemente si descomentamos (esto es, si quitamos el ; que está por delante de la línea) la opción correspondiente:

```
extension = php_gd2.dll
```

Una de las tareas fundamentales que pueden ser llevadas a cabo por esta biblioteca consiste en la composición de gráficos en tiempo de ejecución, ya sea al crear nuevos o al modificar las características de otros ya existentes. En este capítulo, observaremos las cualidades de dos herramientas que hacen uso de GD y brindan justamente estos dos servicios: **JpGraph**, para generación de gráficos estadísticos en tiempo real y **phpThumb**, una librería para modificar diferentes aspectos de una imagen existente.
En cuanto a los requerimientos mínimos para su funcionamiento, para utilizar GD necesitaremos contar con PHP versión 4.3.2 o superiores. Para comprobar si disponemos o no de la librería (normalmente, los servicios de alojamiento web dan soporte para su utilización), tenemos varias opciones. Una de ellas es visualizar la salida de la función **phpinfo**, más precisamente en la sección GD:

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

La otra es verificar que alguna de las funciones disponibles por medio de la extensión esté habilitada. En este caso, tomamos como ejemplo **gd_info**, que devuelve un array con información acerca de la versión de GD disponible:

```
<?php

if (function_exists("gd_info")) {
    echo "GD está disponible";
    echo "<pre>";
    print_r(gd_info());
    echo "</pre>";
} else {
    echo "GD no está disponible !";
}

?>
```

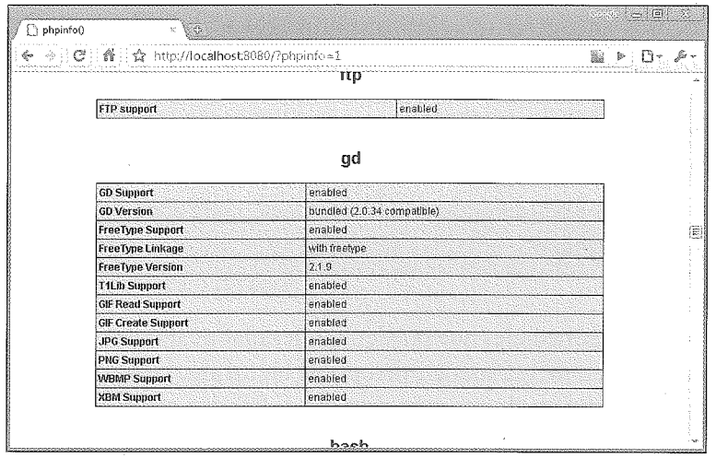

Figura 2. *A través de la función phpinfo podemos obtener información acerca de las características de la instalación de PHP.*

Algunos de los formatos más populares soportados por GD son:

- **GIF** (*CompuServe Graphical Interchange Format*)
- **JPG** o **JPEG** (*Joint Photographic Experts Group*)
- **PNG** (*Portable Network Graphics*, o *PNG is Not GIF*)

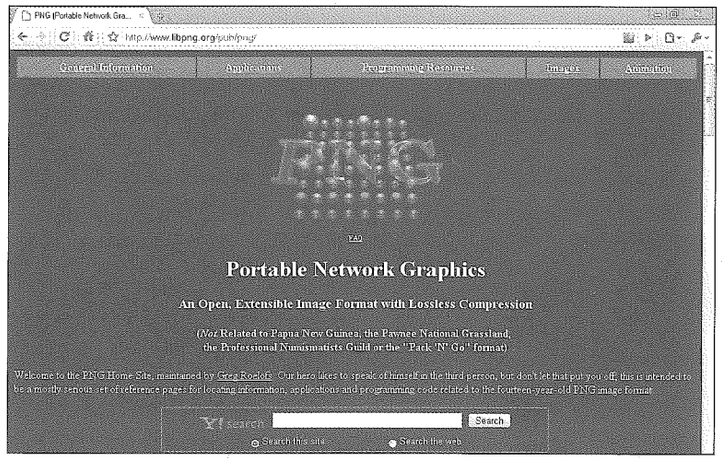

Figura 3. ***PNG*** *es uno de los formatos gráficos más populares de la actualidad, y es apoyado por los que siguen la iniciativa open source.*

Según la versión instalada, la biblioteca GD nos permitirá manipular y procesar distintos tipos de imágenes: las anteriores a la versión 1.6 admiten GIF, pero no PNG, y lo inverso sucede con las superiores.

HERRAMIENTAS DISPONIBLES

La gran popularidad del lenguaje hace que la oferta de aplicaciones y clases que hacen uso de sus características sea realmente inmensa, y lo referido al trabajo con imágenes no es la excepción para este caso.
A continuación, veremos en detalle dos herramientas que nos permitirán, cada una desde su posición, sacar provecho de GD para lograr resultados en un nivel profesional, de manera simple, pero eficaz.

phpThumb

Esta herramienta es de uso libre y nos dará la posibilidad de manipular la visualización de imágenes y, así, poder definir aspectos como el tamaño, el peso, el formato utilizado, y la calidad de la imagen, entre otras muchas opciones. Para funcionar correctamente deberemos contar con GD, que phpThumb la utiliza para trabajar sobre las imágenes.

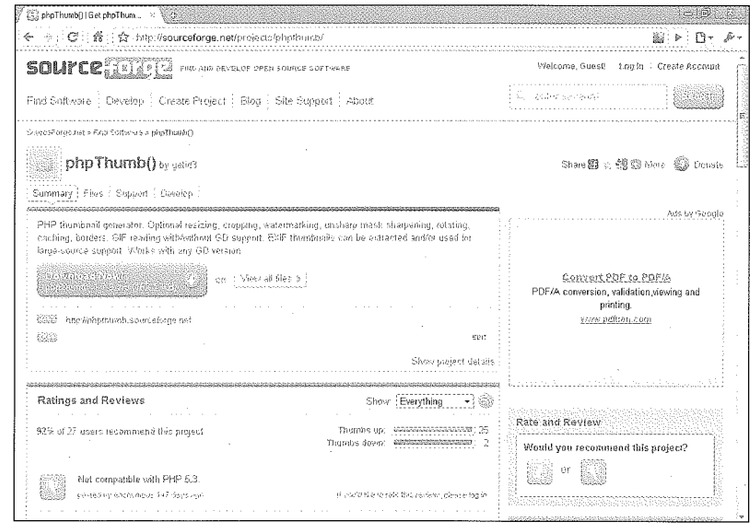

Figura 4. *Las características de GD hacen posible la existencia de herramientas que permiten tratar y generar imágenes de manera simple y profesional.*

Instalacion y forma de uso

Lo primero que deberemos hacer será acceder al sitio oficial de la librería en **http://phpthumb.sourceforge.net** y descargar la última versión disponible. Una vez hecho esto, descomprimimos la distribución y copiamos el directorio a una carpeta accesible a través de un navegador web y renombramos el archivo **phpThumb.config.php.default** a **phpThumb.config.PHP**. La forma de utilización es bastante simple, pero eficaz el archivo **phpThumb.PHP** recibe argumentos, entre

ellos la ruta a la imagen original, y devuelve la imagen modificada. Típicamente se invoca desde el atributo **src** del elemento **img**, disponible en la especificación HTML:

```
<img src="ruta/a/phpthumb.php?a=1&b=2&c=3"/>
```

El argumento principal y básico que recibe este archivo es la ruta (siempre con relación a la ubicación de **phpThumb.php**) a la imagen original, por ejemplo:

```
<img src="ruta/a/phpthumb.php?src=imagen.png" />
```

En este caso, la imagen está en el mismo directorio que **phpThumb.PHP**.
Si suponemos que en nuestro sitio tenemos dos directorios en el mismo nivel, uno denominado **img** (en donde se ubican las imágenes) y otro llamado **phpThumb** (en el cual reside la distribución de la librería, incluido el archivo **phpThumb.php**), la siguiente llamada sería válida:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png" />
```

Opciones disponibles

Además de **src** contamos con otros argumentos que pueden ser incluidos de manera simultánea, entre los que tenemos, por ejemplo, los siguientes:

- El argumento **w** establece un ancho máximo en pixeles para la imagen. Si la imagen original tiene un ancho superior, se reducirá sin perder su aspecto original (**aspect radio**, la relación alto/ancho original) hasta llegar al indicado. Si por el contrario la imagen original tiene un ancho inferior, éste no será modificado:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&w=100" />
```

- El argumento **h** tiene similares características al anterior, el w, y establece un alto máximo en pixeles para la imagen:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&h=90" />
```

Cabe destacar que si phpThumb recibe simultáneamente los argumentos **w** y **h**, buscará generar una imagen que cumpla los dos requisitos sin perder de vista el

aspecto original de imagen, por lo cual puede que alguno de los valores indicados no se cumpla. Hay que recordar que estamos definiendo valores máximos.

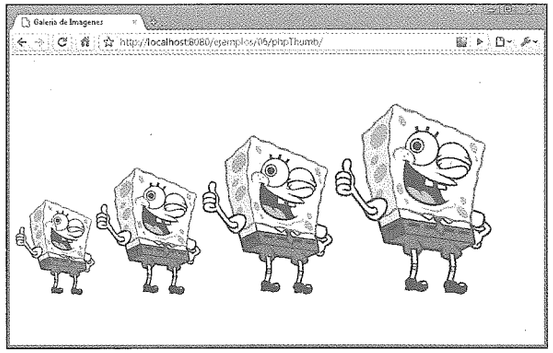

***Figura 5.** La manipulación de imágenes es requerida por una amplia variedad de sitios que necesitan resultados inmediatos en tiempo real.*

- Hasta ahora veníamos utilizando el argumento **src**, pero phpThumb nos permite crear una imagen nueva (desde el argumento **new**) y recibe como valor un número hexadecimal correspondiente al color. Requiere definir los atributos **w** y **h**:

```
<img src="phpThumb/phpthumb.php?new=990000&w=100&h=150" />
```

De manera complementaria, podemos definir un grado de opacidad (porcentaje) de la manera que mostramos a continuación:

```
<img src="phpThumb/phpthumb.php?new=990000|10&w=100&h=150" />
```

En el ejemplo anterior dimos un 10 por ciento de opacidad a la imagen.

- El argumento **f** es uno de los más interesantes y establece un formato de salida para la imagen. Puede tomar como posibles valores **JPEG** (*Joint Photographic Experts Group*), **PNG** (*Portable Network Graphics*, o *PNG is Not GIF*) o **GIF** (*CompuServe GIF o Graphical Interchange Format*):

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&f=jpeg" />
```

Si accedemos al contenido del archivo **phpThumb.config.PHP** que viene junto con la distribución oficial de phpThumb, podremos observar una línea como la siguiente:

```
$PHPTHUMB_CONFIG['output_format'] = 'jpeg';
```

Esta línea indica el formato de salida por defecto (**JPEG**, **PNG** o **GIF**), que tendrá valor siempre y cuando no hayamos definido el argumento **f**. Si no incluimos un valor válido, se tomará por omisión **JPEG**.

- El argumento **q** define la calidad de la imagen y sólo se aplica al formato **JPEG**. Recibe valores dentro de un rango (**1** para baja calidad, **95** para máxima calidad, **75** es el valor por defecto):

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.jpg&q=1" />
```

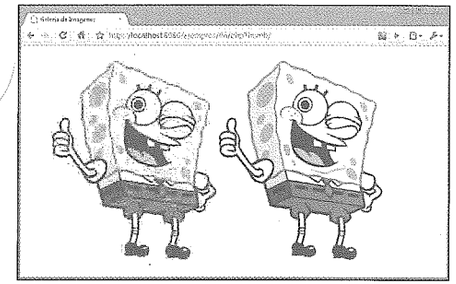

Figura 6. *La disminución de la calidad de las imágenes puede resultar en un menor peso y en una carga más rápida de las páginas que las contienen.*

- Tanto en **w** como en **h**, hablamos acerca de la relación entre el ancho y el alto de la imagen original. El argumento **iar** (*Ignore Aspect Ratio*) ignora las dimensiones originales de la imagen sin mantener su aspecto inicial. Recibe como valor un **1**:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&w=2000&h=20&iar=1" />
```

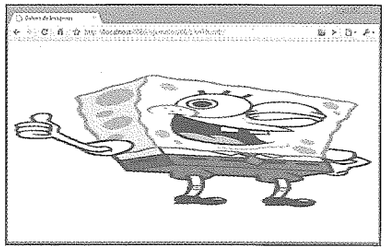

Figura 7. *Las necesidades en cuanto al tratamiento de imágenes son propias de cada aplicación, por lo cual aplicaciones como* ***phpThumb*** *deben brindar servicios variados.*

- La opción **sia** (*Save Image As*) define el nombre que se le dará a la imagen cuando el usuario quiera guardarla:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&sia=nombre" />
```

- El argumento **maxb** puede llegar a ser de mucha utilidad ya que nos da la posibilidad de establecer un valor máximo, en bytes, para la imagen:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&maxb=100000" />
```

En el ejemplo anterior definimos el peso máximo a **100000 bytes**.

- Por su parte, el argumento **down** nos permite forzar la descarga de una imagen. Requiere un comportamiento distinto del visto anteriormente con las demás opciones ya que para forzar la descarga deberemos acceder a un enlace como el siguiente:

```
http://www.nombre-sitio.com/phpThumb/phpthumb.php?src=../img/
    imagen.png&down=nuevaImagen.png
```

Recibe como valor el nombre predeterminado con el cual se guardará.

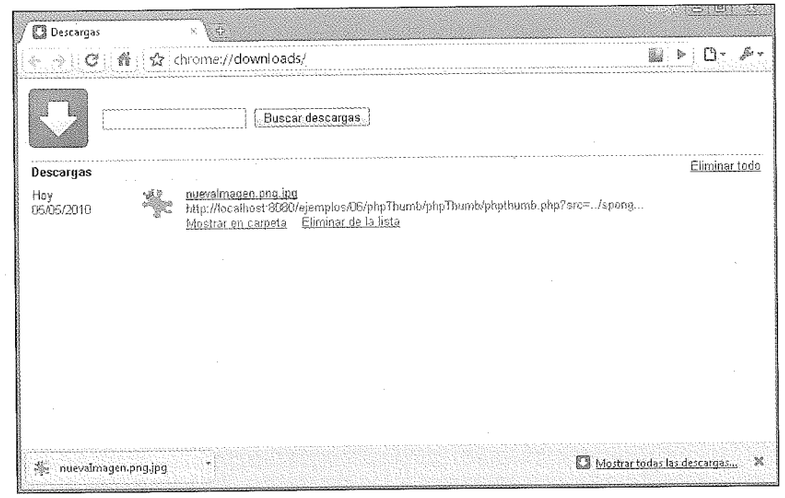

Figura 8. ***phpThumb*** *admite la interacción con los archivos de imagen a través de múltiples maneras, ya sea al exhibir el contenido por pantalla o al enviar la salida a un archivo.*

- Contamos, además, con una serie de argumentos para tomar solamente partes de la imagen original. Todos reciben como valor un número entre 0 (0%) y 1 (100%) y son los siguientes: **sx** (quita un porcentaje de la imagen a partir del margen izquierdo), **sy** (quita un porcentaje de la imagen a partir del margen superior), **sw** (es similar a **sx** sólo que muestra únicamente el porcentaje a partir del margen izquierdo) y **sh** (es similar a **sy** sólo que muestra únicamente el porcentaje a partir del margen superior).

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&sx=0.1" />

<img src="phpThumb/phpthumb.php?src=../img/imagen.png&sy=0.4" />

<img src="phpThumb/phpthumb.php?src=../img/imagen.png&sw=0.3" />

<img src="phpThumb/phpthumb.php?src=../img/imagen.png&sh=0.9" />
```

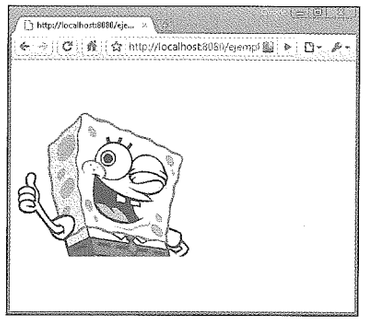

Figura 9. *Tomar partes de una imagen es una de las características más salientes de* ***phpThumb.***

- La opción **zc** devuelve una imagen con las dimensiones definidas (**w** y **h**), pero modifica la imagen para que entre en el marco:

```
<img src="phpThumb/phpthumb.php?src=../img/imagen.png&w=300&h=300&zc=1" />
```

- El argumento **far** crea una imagen con el ancho y alto especificados, pero mantiene el aspecto original, es decir, toma sólo una parte de la imagen si es que las dimensiones de ésta son mayores a las definidas. El valor que recibe indica la alineación (**L** para izquierda, **R** para derecha, **T** para arriba, **B** para abajo, **C** para centro y, también, las combinaciones posibles de valores, como por ejemplo **BL** para abajo a la izquierda):


```
<img src="phpThumb/phpthumb.php?src=../img/
    imagen.png&w=100&h=100&far=B" />
```

Filtros

Además de los argumentos tradicionales vistos hasta ahora, phpThumb nos ofrece la posibilidad de aplicar filtros sobre las imágenes originales para obtener los resultados esperados. Para utilizar filtros contamos con la siguiente sintaxis:

```
fltr[]=opcion|valor
```

De esta manera, podemos aplicar múltiples filtros a la vez, incluso, utilizarlos en conjunto con los argumentos habituales. El atributo **fltr** puede tomar, entre otros, alguno de los valores explicados en la tabla a continuación:

NOMBRE	DESCRIPCIÓN
brit	Brillo, toma valores entre -255 y 255.
cont	Contraste, toma valores entre -255 y 255.
gam	Corrección 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	Blur, toma valores entre 0 y 25.
over	Ubica una imagen por encima de otra. Recibe el nombre de la imagen y 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 un 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.*

Los filtros nos pueden ser de utilidad para observar de manera rápida y precisa cómo las imágenes encajan en la presentación de nuestras páginas web.

Caché

Dentro del archivo **phpThumb.config.PHP** hay disponibles múltiples directivas de configuración de carácter general, y muchas de ellas están referidas al manejo del caché de imágenes por parte de la aplicación.

Al momento de invocar el archivo **phpthumb.PHP**, la aplicación realiza un procesamiento y toma como base la imagen original (si es que existe) y devuelve la nueva salida. Por defecto, cada vez que actualicemos la página el proceso volverá a repetirse, incluso cuando la salida sea la misma. Esto puede evitarse si hacemos uso de la memoria caché que nos permite, básicamente, realizar sólo la primera transformación y almacenar el resultado (esto es, la imagen generada) en un directorio. La siguiente vez que invoquemos **phpthumb.PHP** con los mismos argumentos, la aplicación buscará un resultado y, en caso de encontrarlo, evitará todo el procesamiento. Así se logrará una mayor velocidad en la respuesta.

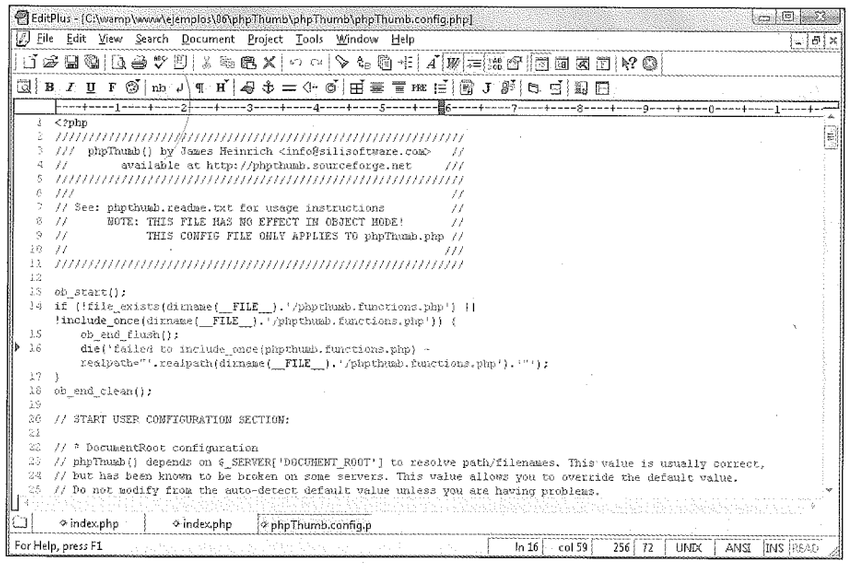

***Figura 10.** El archivo de configuración **phpThumb.config.PHP** provee múltiples opciones para afinar **phpThumb** a lo que realmente necesitamos.*

El directorio en el cual se almacenarán las imágenes cacheadas se define por medio de esta directiva, lo podemos ver en el código siguiente:

```
$PHPTHUMB_CONFIG['cache_directory'] = dirname(__FILE__).'/cache/';
```

La sentencia **dirname(__FILE__)** devuelve el directorio actual, en este caso, el mismo correspondiente al archivo **phpThumb.PHP**. Como vemos, se toma como base el

directorio caché que viene junto con la distribución de la biblioteca phpThumb. Para limitar el uso del caché y no sobrecargar el espacio en disco del servidor, contamos con las siguientes opciones:

- Eliminar imágenes almacenadas en caché que no hayan sido utilizadas en los **X** segundos posteriores al último acceso (**null** para nunca eliminar, en el ejemplo que se encuentra a continuación, indicamos 30 días)

```
$PHPTHUMB_CONFIG['cache_maxage'] = 86400 * 30;
```

- Tamaño máximo del caché (en bytes, **null** para ilimitado, en el siguiente ejemplo limitamos a 10 MB)

```
$PHPTHUMB_CONFIG['cache_maxsize'] = 10*1024*1024;
```

- Número máximo de archivos en caché (**null** para ilimitados)

```
$PHPTHUMB_CONFIG['cache_maxfiles'] = 200;
```

Asimismo, tenemos una opción para deshabilitar los mensajes de advertencia ante posibles fallos durante la generación o administración del caché:

```
$PHPTHUMB_CONFIG['cache_disable_warning'] = true;
```

Al momento de realizar un procesamiento, la aplicación almacena datos en un directorio temporal. Podemos definirlo a través de la siguiente opción (**null** para autodetectar y utilizar el directorio temporal del sistema):

```
$PHPTHUMB_CONFIG['temp_directory'] = null;
```

Ejemplo completo: galería de imágenes

En el siguiente ejemplo desarrollaremos cómo utilizar phpThumb para generar una galería de imágenes. Lo primero que haremos será recorrer un directorio (en nuestro caso, uno llamado **imágenes**) y cargar en un array todas las imágenes disponibles (suponemos que habrá únicamente imágenes, aunque podríamos validar que sólo recupere determinados tipos de archivos como **JPEG**, **GIF**, o **PNG**):

```
$directorio = "imagenes/";

if (is_dir($directorio)) {
    if ($dd = opendir($directorio)) {
        while (($archivo = readdir($dd)) !== false) {
            if (filetype($directorio.$archivo) == 'file') {
                $imagenes[] = $directorio.$archivo;
            }
        }
        closedir($dd);
    }
}
```

Si encontramos archivos (es decir, si el array **imágenes** tiene posiciones) procedemos a generar la salida. Cada imagen será ubicada en una celda de una tabla HTML, y contaremos con cuatro imágenes por fila:

```
echo "<table border='0' cellspacing='0' cellpadding='4' align='center'>";
echo "<tr>";

foreach ($imagenes as $imagen) {
    $contadorImagenes++;
    $contadorColumnas++;

    echo "<td><img src='phpThumb/phpThumb.php?src=../".$imagen."&w=100'
        border='0' /></td>";

    if ($contadorColumnas == 4 && $contadorImagenes != count($imagenes)) {
        echo "</tr><tr>";
        $contadorColumnas = 0;
    }
}

echo "</tr>";
echo "</table>";
```

Para mostrar las imágenes en su tamaño original, al hacer clic sobre ellas haremos uso de la librería **Lightbox**, la cual puede ser descargada desde su sitio web oficial en la dirección **http://www.huddletogether.com/projects/lightbox/**.

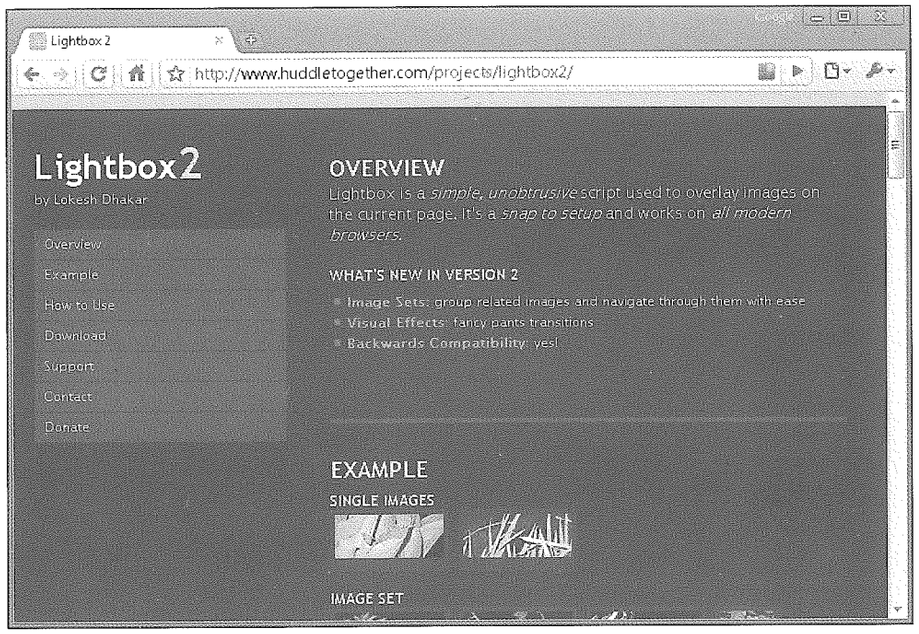

Figura 11. ***Ligtbox2*** *es una popular librería escrita en* ***JavaScript*** *para implementar galerías de imágenes en páginas web.*

Lightbox requiere de otras librerías para su funcionamiento (**Prototype** y **Script.aculo.us**, que vienen incluidas al descargar Lightbox) y para incluirlas en nuestras páginas deberemos añadir las siguientes líneas dentro del **head**:

```
<script src="lightbox/js/prototype.js" type="text/javascript"></script>
<script src="lightbox/js/scriptaculous.js?load=effects,builder"
    type="text/javascript"></script>
<script src="lightbox/js/lightbox.js" type="text/javascript"></script>
```

Será necesario, también, incluir la hoja de estilos CSS utilizada por Lightbox:

EXPANDIR JAVASCRIPT

Prototype es una librería de propósito general escrita en **JavaScript**, que enmascara algunas de las funciones del lenguaje al agregarle más capacidades. Podemos encontrar más información respcto de sus características en la dirección **www.prototypejs.org**.

```
<link rel="stylesheet" href="lightbox/css/lightbox.css" type="text/css"
   media="screen" />
```

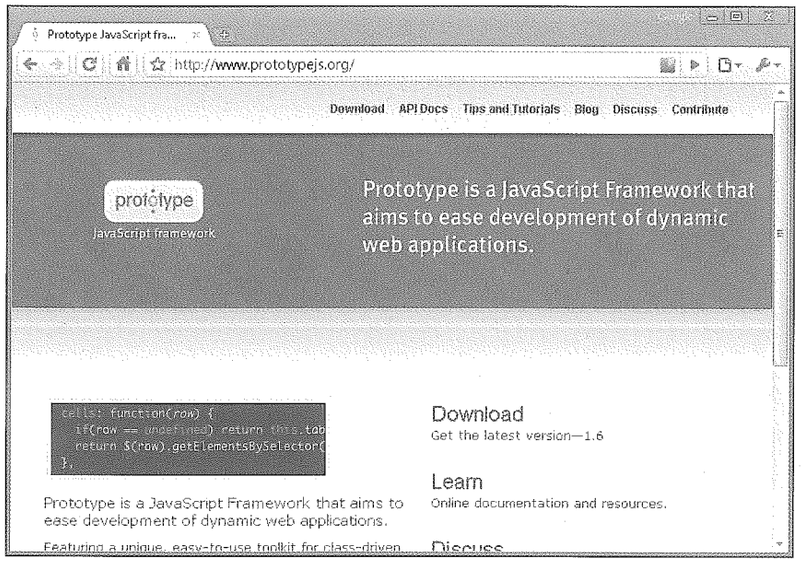

***Figura 12. Ligthbox** es una de las tantas librerías que basa su funcionamiento en **Prototype** para lograr los resultados buscados.*

Para finalizar, dejamos lo más importante, definir los enlaces para invocar Lightbox al momento de hacer clic sobre una de las miniaturas:

```
echo "<td><a href='".$imagen."' rel='lightbox[galeria]'><img
   src='phpThumb/phpThumb.php?src=../".$imagen."&w=100' border='0'
   /></a></td>";
```

Básicamente, son dos puntos:

REINVENTAR LA INTERFAZ DE USUARIO

Script.aculo.us es una librería que nos permite aplicar efectos visuales de manera rápida y profesional, sobre los elementos de una página. Está basada en **Prototype** y podemos obtener más información en su sitio web oficial, en la dirección **http://script.aculo.us/**.

- enlazar a la imagen original,
- incluir dentro del atributo **rel** del link el texto **lightbox[nombre_de_la_galeria]**, lo que nos permitirá navegar entre las diferentes imágenes.

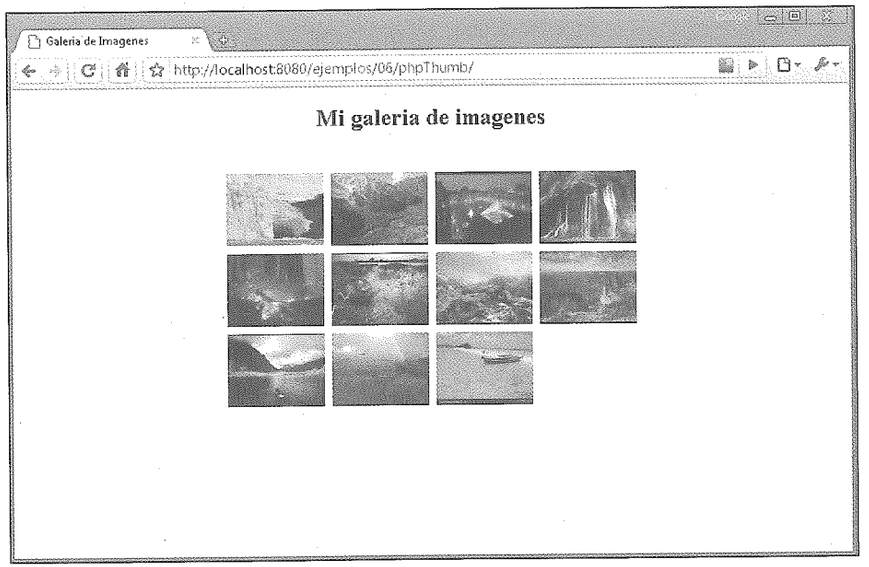

***Figura 13.** Con **phpThumb** podemos obtener resultados profesionales con unas pocas líneas de código.*

El código completo de la aplicación es el 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> Galeria de Imagenes </title>

  <link rel="stylesheet" href="lightbox/css/lightbox.css" type="text/css"
     media="screen" />

  <script src="lightbox/js/prototype.js" type="text/javascript"></script>
  <script src="lightbox/js/scriptaculous.js?load=effects,builder"
     type="text/javascript"></script>
  <script src="lightbox/js/lightbox.js" type="text/javascript"></script>
 </head>
 <body bgcolor="#FFFFCC">
```

```
<?php

$directorio = "imagenes/";

if (is_dir($directorio)) {
    if ($dd = opendir($directorio)) {
        while (($archivo = readdir($dd)) !== false) {
            if (filetype($directorio.$archivo) == 'file') {
                $imagenes[] = $directorio.$archivo;
            }
        }
        closedir($dd);
    }
}

if (is_array($imagenes)) {
    echo "<center><h2>Mi galeria de imagenes</h2></center><br />";

    echo "<table border='0' cellspacing='0' cellpadding='4'
       align='center'>";
    echo "<tr>";

    foreach ($imagenes as $imagen) {
        $contadorImagenes++;
        $contadorColumnas++;

        echo "<td><a href='".$imagen."' rel='lightbox[galeria]'><img
           src='phpThumb/phpThumb.php?src=../".$imagen."&w=100' border='0'
           /></a></td>";

        if ($contadorColumnas == 4 && $contadorImagenes !=
           count($imagenes)) {
            echo "</tr><tr>";
            $contadorColumnas = 0;
        }
    }

    echo "</tr>";
    echo "</table>";
}
```



```
?>

  </body>
</html>
```

JpGraph

Esta librería, que hace uso de la biblioteca GD para su funcionamiento, nos permite la generación de gráficos fundamentalmente estadísticos para representar diversas informaciones de manera visual e intuitiva.

A diferencia de phpThumb, **JpGraph** nos brinda una interfaz orientada a objetos, lo cual implica que para implementar soluciones basadas en esta librería, deberemos acceder a los métodos y a las propiedades que las clases disponibles ponen a disposición.

Para comenzar a desarrollar soluciones tendremos que contar con la biblioteca GD versión 1.8 o superior. En su sitio oficial, accesible desde la dirección **www.aditus.nu/jpgraph**, JpGraph incluye distribuciones para las distintas versiones disponibles de PHP: una para las versiones 4.x y otra para las 5.x y superiores. Una vez descargada la opción seleccionada, procedemos a descomprimir la distribución a un directorio accesible por medio de un navegador web.

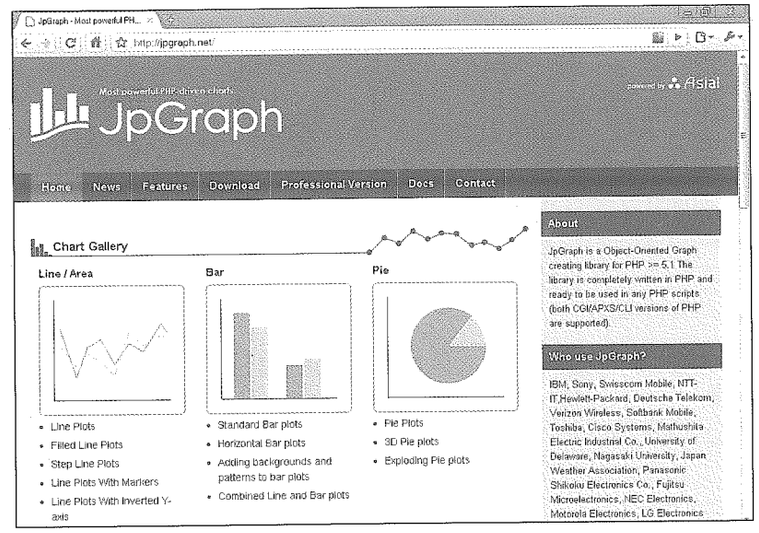

***Figura 14. JpGraph** es una de las librerías para generación de gráficos estadísticos más populares de la actualidad.*

Dentro del directorio llamado **src** encontramos las clases necesarias para incluir en nuestros scripts y comenzar a desarrollar aplicaciones. Para utilizar JpGraph tenemos

que agregar en nuestras páginas el archivo **jpgraph.PHP**. Si suponemos que se encuentra dentro de un directorio denominado **jpgraph**, podríamos escribir la siguiente línea:

```
include "jpgraph/jpgraph.php";
```

Veremos en los próximos apartados que JpGraph pone a nuestra disposición más archivos que deberemos incluir según queramos generar un tipo de gráfico u otro. Entre los tipos disponibles podemos citar:

- Lineales
- Barra
- Torta
- Anillo
- Mapas HTML
- Gráficos Polares
- Radar
- Diagramas de Gantt
- Texto (CAPTCHA)
- Figuras
- Leds
- Código de barras
- Rosa de los vientos
- Velocímetros
- Puntos

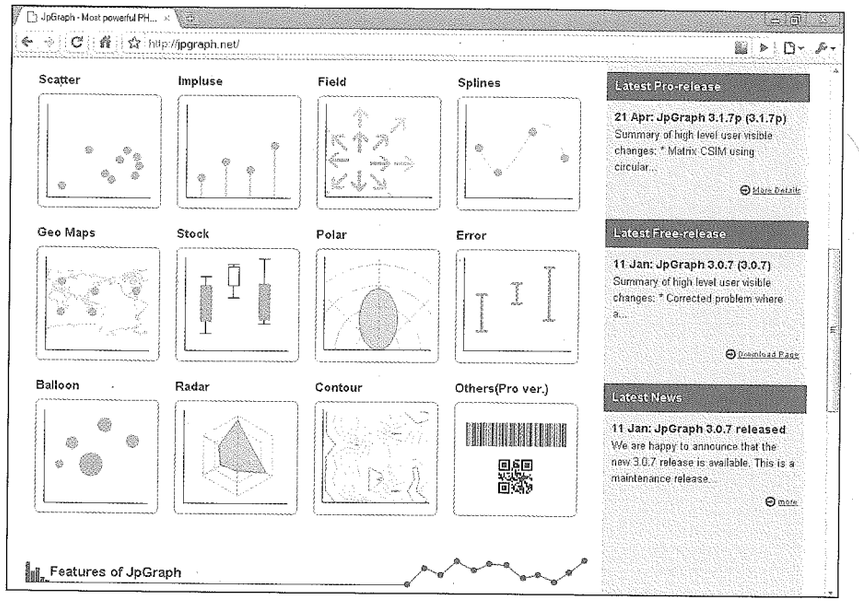

***Figura 15.** La amplia variedad de tipos de gráficos disponibles es una de las mayores virtudes de JpGraph.*

Como veremos, cada uno de estos gráficos posee una clase asociada, pero aun así poseen propiedades y métodos en común. Entre las propiedades podemos nombrar:

- **title**: especifica el título del gráfico.
- **subtitle**: declara el subtítulo del gráfico.
- **subsubtitle**: describe el segundo subtítulo del gráfico.
- **legend**: define la leyenda del gráfico.

Y con relación a los métodos:

- **Add**: se utiliza para añadir un elemento al gráfico.
- **SetMargin**: define el margen del gráfico.
- **SetMarginColor**: especifica el color del margen del gráfico.
- **SetColor**: declara el color del gráfico.
- **SetBox**: define si tanto el gráfico como su leyenda estarán encerrados en una caja.
- **SetShadow**: nos permite asignar una sombra al gráfico.
- **SetGridDepth**: agrega líneas por encima del gráfico o por debajo.
- **SetAngle**: define el ángulo de inclinación del gráfico.
- **SetBackgroundImage**: recibe como argumento la ruta a una imagen que se ubicará como fondo del gráfico generado.
- **SetAxisStyle**: define el estilo de los ejes horizontales y verticales.

JpGraph se caracteriza por poseer una amplia variedad de propiedades y métodos para cada uno de sus gráficos, por lo cual será imprescindible contar con el manual de la aplicación que enumera, con ejemplos, con la mayoría de las alternativas disponibles.

Tipos de gráficos soportados

A continuación, desarrollaremos un sistema que nos muestra como implementar en nuestras aplicaciones algunas de las alternativas relativas a la generación de gráficos dinámicos a traves de JpGraph. Comencemos con los gráficos de tipo lineal, que quizá sean los más simples y los más clarificadores para que podamos comprender el funcionamiento básico de esta librería. Como dijimos anteriormente, y esto vale para la mayoría de los gráficos disponibles, lo primero que deberemos hacer será incluir en nuestros scripts al archivo **jpgraph.PHP**:

```
include 'jpgraph/jpgraph.php';
```

Para disponer de las herramientas necesarias para crear específicamente gráficos lineales tendremos que contar con el archivo **jpgraph_line.PHP**:

```
include 'jpgraph/jpgraph_line.php';
```

JpGraph tiene la clase **Graph**, que nos servirá para contener el gráfico que luego generaremos. El constructor de esta clase recibe como argumentos el ancho de la imagen a generar, el alto, y el formato del archivo (**auto**, explicaremos más acerca de esto en los próximos apartados):

```
$grafico = new Graph(400, 300, "auto");
```

Cabe destacar que JpGraph admite la inclusión de múltiples gráficos, incluso de distinto tipo, en una misma imagen (contenedor).
El siguiente paso será definir los valores necesarios para especificar la escala del gráfico a generar: esto se logra a través del método **SetScale**, que recibe como argumentos el tipo de escala, el mínimo y el máximo del eje **Y**, y el mínimo y el máximo del eje **X**. Estos últimos cuatro son opcionales.
Dependiendo el tipo de escala, contamos con las alternativas:

- lineal (**lin**),
- logarítmica (**log**),
- textual (**text**, sólo para X),
- entera (**int**)

Primero definimos el tipo de escala para el eje **X** y luego para el eje **Y**:

```
$grafico->SetScale("textlin");
```

Los gráficos lineales poseen su propia clase denominada **LinePlot** que recibe como argumento un array de datos:

```
$datos = array(10, 20, 5, 15, 20);
```

```
$linea = new LinePlot($datos);
```

Lo próximo será agregarlo al gráfico:

```
$grafico->Add($linea);
```

Por último, enviamos la salida al navegador por medio del método **Stroke**:


```
$grafico->Stroke();
```

Si en lugar de esto preferimos guardar la imagen en un archivo, podemos hacerlo pasándole al método, como parámetro, la ruta hacia él:

```
$grafico->Stroke('imgs/nombreGrafico.png');
```

Esta característica es válida para todos aquellos gráficos que cuenten con este método.
Para acceder a las propiedades y a los métodos de los ejes del gráfico (X e Y para los de dos dimensiones), JpGraph pone a nuestra disposicion la siguiente sintaxis que podemos visualizar en el código que aparece a continuación:

```
$grafico->xaxis->title->Set("Título del Eje X");
```

```
$grafico->yaxis->title->Set("Título del Eje Y");
```

Para establecer la leyenda de la figura contamos con el método **SetLegend**:

```
$linea->SetLegend("Texto de la leyenda");
```

Eventualmente, un gráfico de este tipo puede exhibir los valores alcanzados y en este sentido podemos dar un formato específico, por ejemplo, si definimos la cantidad de decimales o los enteros a visualizar:

```
$linea->value->SetFormat("%0.2f");
```

III MAPAS HTML

JpGraph nos brinda la posibilidad de generar mapas HTML. Un mapa de este tipo nos permite ligar partes de una imagen a enlaces HTTP, y así poder personalizar la navegación del usuario dentro de nuestras páginas. Además de la imagen, en este caso JpGraph crea fragmentos HTML.

Para que esto tenga efecto debemos indicar que los valores serán exhibidos (por omisión JpGraph no ha de mostrarlos), lo que se logra a través del método **Show** de la propiedad **value**, como podemos ver en la siguiente línea de código.

```
$linea->value->Show();
```

En cuanto a las fuentes utilizadas para formar los textos componentes del gráfico, podemos elegir cuál utilizar. Para esto, el método **setFont** admite tres argumentos: tipo de fuente, estilo y tamaño, en ese orden. Veamos, por ejemplo, cómo definir la fuente del título del gráfico:

```
$grafico->title->Set("Título del gráfico");
```

```
$grafico->title->SetFont(FF_VERDANA, FS_BOLD, 10);
```

Como observamos, los dos primeros están representados por valores constantes. Entre las disponibles para las fuentes se encuentran:

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 2.** Constantes para definir el tipo de fuente de los gráficos.*

En cuanto al estilo aplicado a la fuente contamos con:

CONSTANTE	DESCRIPCIÓN
FS_NORMAL	Normal
FS_BOLD	Negrita
FS_ITALIC	Itálica
FS_BOLDITALIC	Itálica y negrita

***Tabla 3.** Constantes para definir el estilo de la fuente de los gráficos.*

Finalmente, el código de nuestro primer ejemplo sería el siguiente:


```
<?php

include 'jpgraph/jpgraph.php';
include 'jpgraph/jpgraph_line.php';

$grafico = new Graph(400, 300, "auto");

$grafico->SetScale("textlin");

$datos = array(10, 20, 5, 15, 20);

$linea = new LinePlot($datos);

$linea->SetLegend("Texto de la leyenda");

$linea->value->SetFormat("%0.2f");

$linea->value->Show();

$grafico->Add($linea);

$grafico->xaxis->title->Set("Título del Eje X");

$grafico->yaxis->title->Set("Título del Eje Y");

$grafico->title->Set("Título del gráfico");

$grafico->title->SetFont(FF_VERDANA, FS_BOLD, 10);

$grafico->Stroke();
?>
```

MÚLTIPLES BARRAS

Los gráficos de tipo barra permiten variantes como la inclusión de múltiples franjas, una sobre otra, o alineadas. Esto se puede lograr si utilizamos las clases disponibles para tal fin: **GroupBarPlot** y **AccBarPlot**. Podemos obtener más información sobre éstas en el manual oficial de **JpGraph**.

La salida estaría conformada por este gráfico:

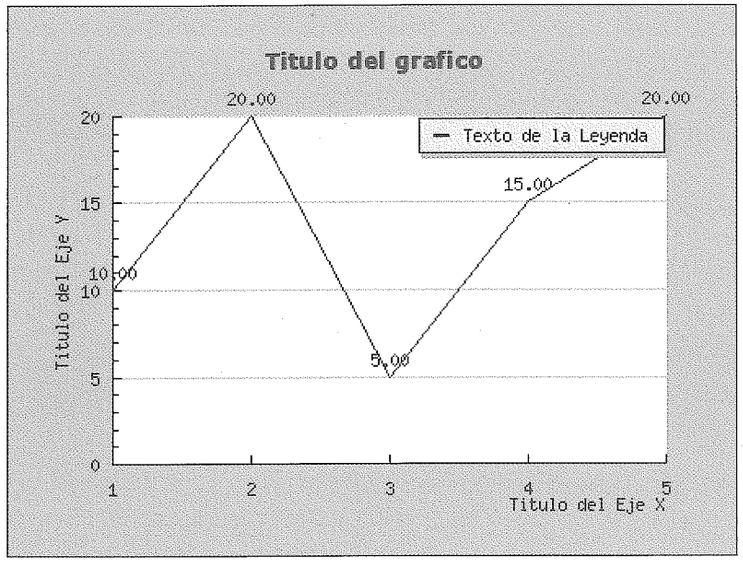

Figura 16. *Los gráficos lineales son sencillos para implementar y, a la vez, claros para visualizar.*

Como observamos, la imagen se vuelca directamente al navegador. Si quisiéramos incluir el gráfico dentro de una página HTML, que es la situación más habitual, podríamos recurrir a ubicar el código que genera el gráfico en un archivo (podríamos llamarlo **grafico.php**) y ubicar en nuestro código HTML una línea como la siguiente:

```
<img src="grafico.php" />
```

Incluso algo semejante a lo que sigue es válido:

```
<img src="grafico.php?a=1&b=2&c=3" />
```

Esto nos daría la posibilidad de dinamizar nuestros gráficos y obtener resultados utilizando fuentes externas de datos, como por ejemplo, bases de datos, documentos XML, servicios web, etcétera. Las posibilidades son ilimitadas.
Continuaremos trabajando con los tipos de gráficos disponibles en JpGraph y daremos una mirada rápida por los de barra.
En cuanto a los archivos que necesitamos para hacer uso de esta biblioteca deberemos incluir **jpgraph.php** y **jpgraph_ bar. php**:


```
include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_bar.php");
```

Luego de incluir los archivos de biblioteca en el módulo o módulos que necesitamos generar gráficos, realizamos un contenedor a través de la clase en los **Graph** y definimos la escala correspondiente a través del método **SetScale**:

```
$grafico = new Graph(600, 400, "auto");
```

```
$grafico->SetScale("textlin");
```

Para implementarlos contamos con la clase **BarPlot**, que recibe como argumento un array de datos, al igual que el ya visto **LinePlot**:

```
$datos = array(10, 20, 30, 29, 19, 9);
```

```
$barra = new BarPlot($datos);
```

Nuevamente podríamos definir valores para el título:

```
$grafico->title->Set("Título del gráfico");
```

Y los ejes horizontales (X) y verticales (Y) de la siguiente manera:

```
$grafico->xaxis->title->Set("Titulo del Eje X");
```

```
$grafico->yaxis->title->Set("Titulo del Eje Y");
```

Podríamos, también, especificar un texto descriptivo para la leyenda con la utilización del método **SetLegend**, veamos el ejemplo a continuación.

```
$barra->SetLegend("Texto de la leyenda");
```

Una opción importante es la que respecta a la definición del ancho de las barras. Para esto contamos con dos alternativas: asignarlo en pixeles, a través de **SetAbsWidth**, o en porcentajes con relación al ancho del gráfico, por medio de **SetWidth**. Presentamos ejemplos para cada una:

```
$barra->SetAbsWidth(40); //40 pixeles
```

```
$barra->SetWidth("0.8"); //80 porciento
```

Especificamos que queremos mostrar los valores de las columnas:

```
$barra->value->Show();
```

En cuanto al fondo de cada barra, tenemos dos posibilidades: utilizar un color en hexadecimal (**SetFillColor**) o utilizar gradientes, combinación de colores (**SetFillGradient**). Veamos ejemplos para cada opción.

```
$barra->SetFillColor("#990000");
```

```
$barra->SetFillGradient("#990000", "#D0BAA7", GRAD_VER);
```

El último argumento a **SetFillGradient** es una constante que indica la alineación de los colores pasados en primer y segundo lugar. Otras disponibles son:

- 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 último, añadimos la barra al contenedor:


```
$grafico->Add($barra);
```

Y enviamos el gráfico al navegador para poder visualizarlo:

```
$grafico->Stroke();
```

Si utilizamos los gráficos de barras, el código explicado hasta aquí en nuestro ejemplo, sería el código que podemos ver a continuación.

```
<?php

include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_bar.php");

$grafico = new Graph(600, 400, "auto");

$grafico->SetScale("textlin");

$datos = array(10, 20, 30, 29, 19, 9);

$barra = new BarPlot($datos);

$grafico->title->Set("Título del grafico");

$grafico->xaxis->title->Set("Título del Eje X");

$grafico->yaxis->title->Set("Título del Eje Y");

$barra->SetLegend("Texto de la leyenda");

$barra->SetWidth("0.8"); //80 porciento

$barra->value->Show();

$barra->SetFillGradient("#990000", "#D0BAA7", GRAD_VER);

$grafico->Add($barra);
```

```
$grafico->Stroke();

?>
```

La salida sería una imagen como la que se exhibe a continuación:

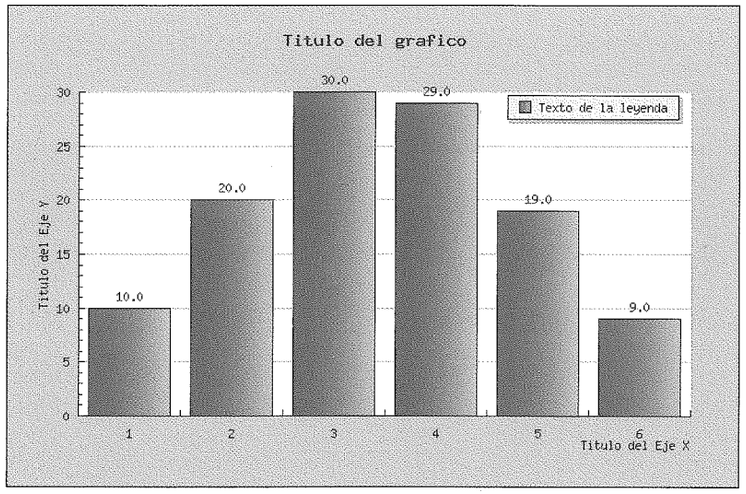

Figura 17. *La personalización de los esquemas generados es muy profunda y los gráficos de barra no son la excepción.*

Un tipo de imagen especial generada por JpGraph son los Leds, conjuntos de puntos que forman caracteres. Para hacer uso de esta característica, primero deberemos incluir el archivo **jpgraph_led.PHP** y crear una instancia de la clase **DigitalLED74**, la cual recibe como argumentos opcionales el radio (por defecto 2) y el margen (por defecto 0.6). Veamos un ejemplo:

```
<?php

include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_led.php ");

$led = new DigitalLED74();

$led->StrokeNumber(" Este es un ejemplo ", LEDC_YELLOW);

?>
```


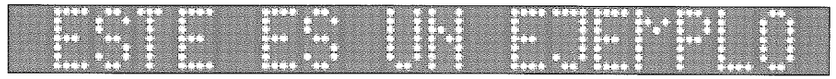

Figura 18. *JpGraph no sólo brinda gráficos de tipo estadístico, aunque ésta sea su característica más importante.*

Hay que notar que usamos **StrokeNumber** en lugar de **Stroke**. El primer argumento a este método es el texto a imprimir y el color a través de una constante. A continuación, el listado de las constantes:

- LEDC_RED
- LEDC_GREEN
- LEDC_BLUE
- LEDC_YELLOW
- LEDC_GRAY
- LEDC_CHOCOLATE
- LEDC_PERU
- LEDC_GOLDENROD
- LEDC_KHAKI
- LEDC_OLIVE
- LEDC_LIMEGREEN
- LEDC_FORESTGREEN
- LEDC_TEAL
- LEDC_STEELBLUE
- LEDC_NAVY
- LEDC_INVERTGRAY

Volvamos a los gráficos tradicionales: JpGraph nos ofrece la posibilidad de implementar el estilo de **tortas** y **anillos**. Como veremos en los próximos ejemplos, la forma de trabajo en ambos casos es similar. Para los gráficos de tipo torta, primero deberemos incluir dos archivos además de **jpgraph.php**: **jpgraph_pie.php** y **jpgraph_pie3d.php**.

```
include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_pie.php";
include "jpgraph/jpgraph_pie3d.php";
```

En lugar de la clase **Graph** se toma como base la clase **PieGraph**, que recibe los mismos argumentos, podemos verlo ejemplificado en las líneas siguientes.

```
$grafico = new PieGraph(400, 200, "auto");
```

Por su parte, la clase **PiePlot3D** recibe como argumento un array con los datos a graficar, que al generar el gráfico se convertirán en porcentajes:

```
$datos[] = 10;
$datos[] = 20;
$datos[] = 11;
$datos[] = 31;

$torta = new PiePlot3D($datos);
```

El método **SetCenter** define la posición con relación al contenedor (0.5 para ubicarla en el centro de éste):

```
$torta->SetCenter(0.35);
```

En cuanto a las leyendas, necesitaremos una por porción, por lo que tendremos que incluir un array con el mismo número de posiciones:

```
$leyendas[] = "Leyenda 1";
$leyendas[] = "Leyenda 2";
$leyendas[] = "Leyenda 3";
$leyendas[] = "Leyenda 4";

$torta->SetLegends($leyendas);
```

Finalmente añadimos el gráfico al contenedor:

```
$grafico->Add($torta);
```

* VERSIONES DISPONIBLES

JpGraph nos ofrece dos versiones de este producto: una versión gratuita y otra paga, que agrega ciertas funcionalidades a la versión básica, como por ejemplo, la manipulación de códigos de barra o velocímetros. De cualquier manera, la versión básica es muy completa y nos bastará para realizar desarrollos de alta calidad profesional.

Por último, enviamos la salida al navegador:

```
$grafico->Stroke();
```

El código completo de nuestro ejemplo es el siguiente:

```
<?php

include "jpgraph/jpgraph.php";
include "jpgraph/jpgraph_pie.php";
include "jpgraph/jpgraph_pie3d.php";

$grafico = new PieGraph(400, 200, "auto");

$datos[] = 10;
$datos[] = 20;
$datos[] = 11;
$datos[] = 31;

$torta = new PiePlot3D($datos);

$torta->SetCenter(0.35);

$leyendas[] = "Leyenda 1";
$leyendas[] = "Leyenda 2";
$leyendas[] = "Leyenda 3";
$leyendas[] = "Leyenda 4";

$torta->SetLegends($leyendas);
```

III LA OPCIÓN DE PEAR

El repositorio de clases oficial de PHP pone a disposición de los usuarios una alternativa para generar gráficos estadísticos de manera rápida y profesional. La herramienta se denomina **Image_Graph** (también conocida como **GraPHPite**) y podemos obtener más información en **http://graphpite.sourceforge.net/**.

```
$grafico->Add($torta);

$grafico->Stroke();

?>
```

La salida es una imagen como la que se ve a continuación:

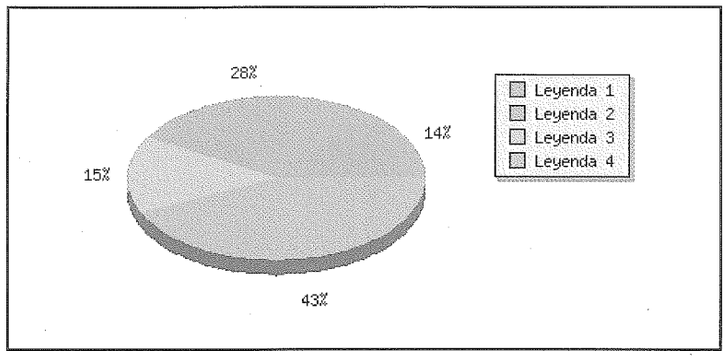

Figura 19. *La implementación de gráficos de torta no difiere demasiado de las demás opciones puestas a disposición por* ***JpGraph.***

Otros métodos de interés para esta clase de gráficos son los siguientes:

- **SetSize**: puede tomar valores entre **0** y **0.5** y especifica el tamaño de la torta con respecto de su contenedor:

```
$torta->SetSize(0.3);
```

- **SetLabelPos**: establece la distancia entre cada título de cada porción de la torta y el centro de ésta, como podemos verlo en el código siguiente.

```
$torta->SetLabelPos(0.5);
```

- **ExplodeAll**: nos permite separar las porciones de la torta al agregar un margen intermedio entre cada una, lo podemos visualizar en el siguiente ejemplo.

```
$torta->ExplodeAll();
```


Los gráficos de tipo anillo son similares, lo único que se modifica es que en lugar de la clase **PiePlot3D** utilizamos **PiePlotC**:

```
$anillo = new PiePlotC($datos);
```

Por útimo, explicaremos cómo implementar soluciones **CAPTCHA** (*Completely Automated Public Turing test to tell Computers and Humans Apart*, Prueba de Turing Pública y Automática para Diferenciar entre Máquinas y Humanos) a través de JpGraph. La idea, básicamente, es generar una imagen que contenga caracteres y un cuadro de texto para que el usuario los ingrese: si coinciden, se valida el ingreso y, en caso contrario, se repite el proceso. El archivo que debemos incluir en esta situación es **jpgraph_antispam.php**:

```
include "jpgraph/jpgraph_antispam.php";
```

Luego tenemos que inicializar una instancia de la clase **AntiSpam**, de la manera siguiente, podemos observarlo en el código que se encuentra a continuación.

```
$spam = new AntiSpam();
```

Hay dos formas para generar el contenido textual de la imagen CAPTCHA:

- de manera aleatoria, por medio del método **Rand**, que recibe como argumento la longitud de la cadena de caracteres:

```
$spam->Rand($longitud);
```

- con la utilización de **Set**, lo que nos permite definir el texto que queramos de manera manual, lo podemos ver ejemplificado a continuación.

```
$spam->Set($cadena);
```

En la práctica, el modo de trabajar con ambos es idéntico.
Al enviar un formulario tendremos que comparar lo ingresado por el usuario con el contenido de la imagen, por lo que deberemos guardar el valor de ésta (es decir, lo devuelto por **Rand** o **Set**) en una variable de sesión para no perder los datos:

```
session_start();
$_SESSION['cadenaCaptcha'] = $spam->Rand(6);
```

En caso de utilizar **Set** en lugar de **Rand**:

```
session_start();
$_SESSION['cadenaCaptcha'] = $spam->Set("AnBJa12P");
```

Por último, imprimimos la imagen generada con el método **Stroke**:

```
$spam->Stroke();
```

Veamos, a continuación, un ejemplo completo de uso:

```
<?php

session_start();

if ($_POST['enviar']) {
    if ($_SESSION['cadenaCaptcha'] == $_POST['texto']) {
        echo 'Ingreso correcto !';
        exit;
    } else {
        echo 'Error en el ingreso ! <br />';
    }
}

?>

<form method='post'>

<table border='0' cellspacing='0' cellpadding='4'>
<tr>
    <td>Ingrese el texto de la imagen</td>
    <td><input type="text" id="texto" name="texto"></td>
</tr>
```



```
<tr>
    <td>Imagen</td>
    <td><img src="captcha.php"></td>
</tr>
<tr>
    <td colspan='2'><input type="submit" id="enviar" name="enviar"
        value="Enviar"></td>
</tr>
</table>

</form>
```

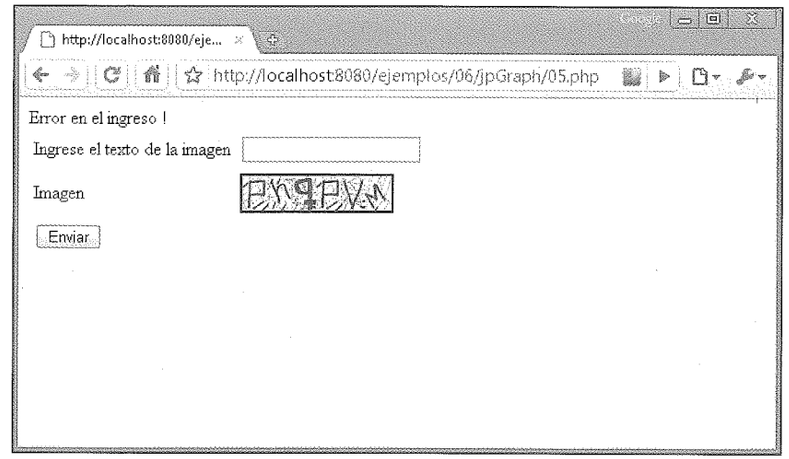

Figura 20. *La implementación **CAPTCHA** de **JpGraph** es rápida y, a la vez, eficaz con relación a las necesidades de las distintas clases de proyectos.*

El contenido de **captcha.php** es el siguiente:

```
<?php

session_start();

include "jpgraph/jpgraph_antispam.php";

$spam = new AntiSpam();

$_SESSION['cadenaCaptcha'] = $spam->Rand(6);
```

```
$spam->Stroke();

?>
```

Opciones de configuración

En el archivo de configuración **jpg-config.inc.php** encontramos múltiples opciones para ajustar las características de la librería. Veremos, algunas de ellas.

- La constante **DEFAULT_GFORMAT** nos permite definir el formato por defecto de nuestros gráficos.

```
DEFINE("DEFAULT_GFORMAT", "png");
```

Si este valor no está definido, JpGraph por omisión intentará primero utilizar **PNG**, **GIF**, o **JPG**, por defecto en ese orden.

Administración del caché de gráficos

Para administrar el uso del caché en JpGraph contamos con las siguientes constantes disponibles en el archivo **jpg-config.php**:

- **USE_CACHE** para habilitar la memoria caché.

```
DEFINE("USE_CACHE", true);
```

- **READ_CACHE** para utilizar los gráficos que están en caché.

```
DEFINE("READ_CACHE", true);
```

- **CACHE_DIR** para definir la ruta absoluta a un directorio temporal, que debe contar con permisos de escritura.

```
DEFINE("CACHE_DIR", "/imagenes/cache/");
```

Una vez que está habilitado, para hacer uso del caché deberemos incluir dos argumentos más al contenedor **Graph** (y a **PieGraph**):


```
$grafico = new Graph($w, $h, $imagenCache, $tiempo);
```

En donde **imagenCache** será el nombre que le daremos a la imagen guardada en caché y **tiempo** el número de minutos antes de actualizar la imagen (debemos poner el valor **0** para no actualizar nunca). Si se requiere el uso del caché y la imagen requerida no está disponible, JpGraph la genera nuevamente.

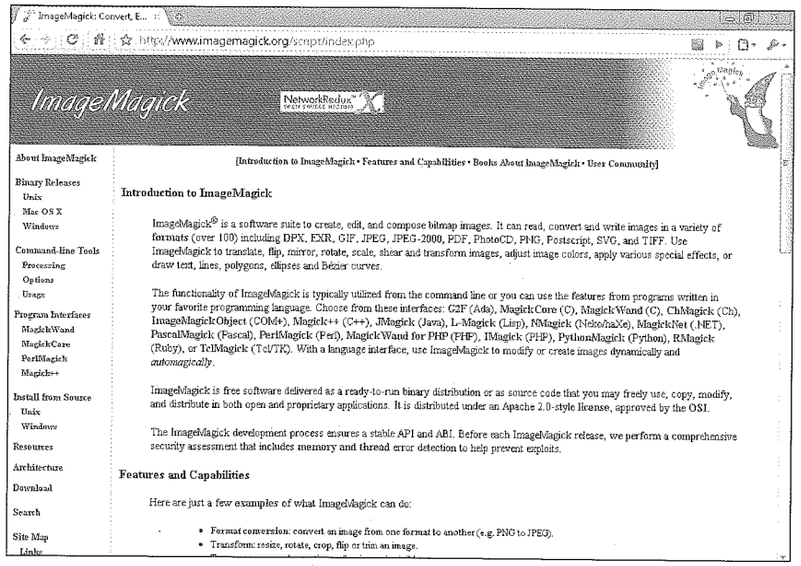

Figura 21. *Una alternativa al manejo de imágenes es **Imagick**, una extensión nativa de PHP para crear y modificar gráficos utilizando la API **ImageMagick**.*

RESUMEN

En este capítulo, observamos algunas de las características más notables en cuanto al tratamiento y la generación de imágenes por medio de PHP. Vimos las cualidades de GD, la biblioteca más popular de la actualidad, y terminamos con una reseña acerca de dos aplicaciones que hacen uso de ella y que, a su vez, nos permiten desatar todo el poder de PHP en lo que atañe a la manipulación de gráficos: phpThumb y JpGraph. También incluimos el uso de ellas para la generación de claves de seguridad antispam, como por ejemplo, el sistema CAPTCHA.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Cuál cree que es la función más importante de GD?

2 ¿Cuáles son los formatos gráficos más populares de la actualidad?

3 ¿Qué es phpThumb?

4 Nombre cinco casos en los cuales sería útil hacer uso de phpThumb.

5 ¿Para qué sirve Lightbox? ¿Requiere PHP?

6 ¿En qué casos utilizaría caché para imágenes?

7 ¿En qué casos no utilizaría caché para imágenes?

8 Nombre cinco casos en los cuales sería útil hacer uso de JpGraph.

9 ¿Cuál es la fuente de datos típica de los gráficos implementados en JpGraph?

10 Nombre cuatro tipos de gráficos disponibles.

EJERCICIOS PRÁCTICOS

1 Genere una galería de imágenes con phpThumb y guarde las miniaturas generadas en un directorio.

2 Implemente un gráfico a través de JpGraph para mostrar un ejemplo de los ingresos/egresos mensuales de un trabajador.

3 Utilice phpThumb para reducir el tamaño de sus archivos gráficos y limitar su peso.

4 Busque información acerca de la generación de mapas HTML a través de JpGraph.

5 Busque aplicaciones similares a phpThumb y compare su funcionamiento.

Capítulo 7

MySQL

A continuación, estudiaremos las características principales de uno de los servidores de bases de datos más populares de la actualidad, y veremos cómo obtener provecho de sus ventajas a través de PHP.

Características principales	**226**
Conceptos del Modelo Relacional	228
Acceso desde PHP	230
El lenguaje SQL	**231**
Tipos de datos en MySQL	231
Funciones de agregado	235
Operadores	235
Opciones para administrar MySQL	**236**
El monitor de MySQL	237
Referencia de funciones	243
Ejemplos prácticos	262
Resumen	**281**
Actividades	**282**

CARACTERÍSTICAS PRINCIPALES

MySQL es, desde hace mucho tiempo, el servidor de bases de datos más utilizado para desarrollar proyectos de cualquier tipo, desde los más simples hasta los más complejos. Esto se debe a una serie de factores, entre los que podemos citar los siguientes:

- Licenciamiento. MySQL es de libre distribución. Significa que podemos descargarlo de manera gratuita e instalarlo en nuestro sistema para desarrollos, tanto para uso particular, como para uso comercial. Pone a disposición de los usuarios dos tipos de licenciamiento: una licencia comercial y una licencia GPL, General Public License. La licencia comercial brinda, entre otras cosas, soporte técnico y garantía.

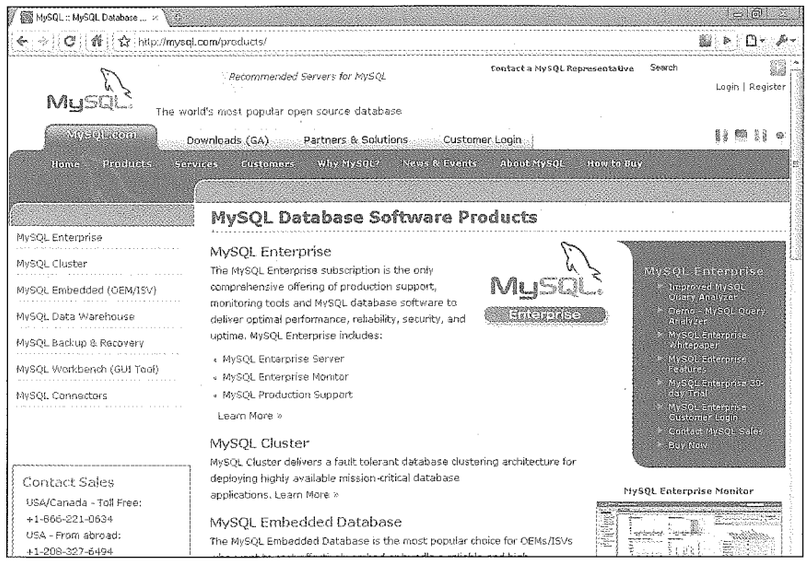

***Figura 1.** MySQL es uno de los motores de bases de datos más utilizados en la actualidad gracias a factores como facilidad de uso, rapidez, y acceso desde múltiples lenguajes de programación.*

- Facilidad de uso. Otra cualidad que podemos destacar es la sencillez con relación a la administración: al ser tan popular, este servidor de bases de datos dispone de un gran número de herramientas destinadas a controlar su funcionamiento de manera intuitiva. Algunas de estas aplicaciones son desarrolladas por la propia empresa responsable de MySQL (**MySQL Administrator**) o por terceros (**phpMyAdmin**). Con estas opciones, lograr acceder y sacar provecho de las cualidades del servidor se vuelve sencillo incluso para aquellos que no tienen experiencia previa en en cuanto a la administración de servidores de bases de datos, o para quienes la tienen con otros administradores de datos, como ser SQL Server, Oracle, etcétera.

- Multiplataforma. MySQL está disponible para distintas plataformas y sistemas operativos, por lo cual su disponibilidad no estará afectada por las características de nuestro servicio de alojamiento. Se puede usar para los siguientes sistemas operativos:

  - AS/400
  - BeOS
  - BSD (FreeBSD, NetBSD, OpenBSD, BSD/OS)
  - Linux
  - Mac OS X
  - Novell NetWare (6.0 o superior)
  - OS/2
  - RISC OS
  - SGI IRIX 6.5.x
  - Solaris
  - Windows (9x, Me, NT, 2000, XP, Vista)

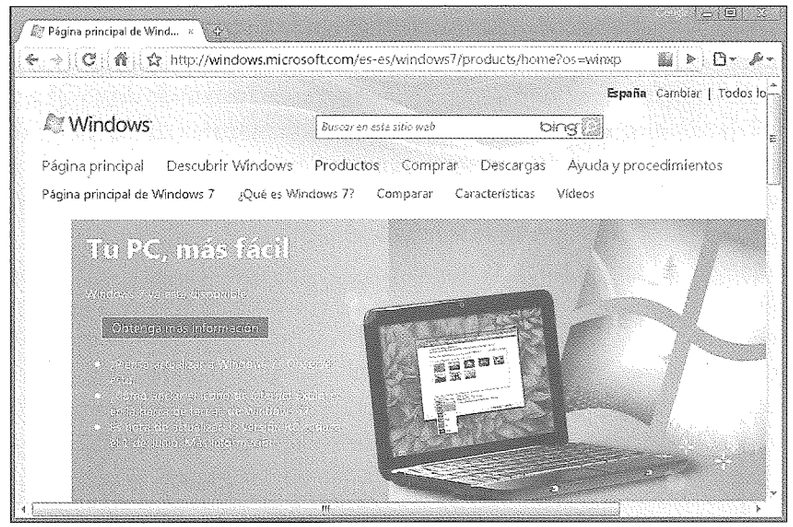

***Figura 2.** Una de las claves de MySQL es la posibilidad de ser utilizado desde distintos sistemas operativos, entre ellos Microsoft Windows.*

- Rapidez. Los niveles de velocidad alcanzados a través de MySQL no tienen nada que envidiar a las estadísticas publicadas por sus competidores. En el sitio oficial en particular y en Internet en general se publican, de manera periódica, pruebas sobre la performance lograda en diferentes situaciones específicas, hechas en distintas plataformas y arquitecturas.
- Seguridad. MySQL es utilizada en el nivel empresarial en grandes proyectos, y en estos y en otros casos es fundamental poder definir un esquema de seguridad acor-

de con las necesidades. A tal fin, provee herramientas (algunas de ellas incluidas en las distribuciones estándar y otras disponibles incluso a través de la empresa que desarrolla y mantiene este motor de bases de datos).

- Estabilidad. Si bien MySQL está preparada para salir de eventuales inconvenientes que atañen a distintos factores (errores internos y externos), éstos no son para nada frecuentes: una de las cualidades más salientes de este motor de bases de datos es su estabilidad alcanzada, algo que genera confianza en quienes lo utilizan para almacenar informaciones sensibles dentro de las organizaciones.

En el sitio web oficial del servidor podremos encontrar mucha información con respecto a con las características más notables y a las diferencias entre versiones: la dirección es **www.mysql.com**. MySQL puede obtenerse de forma completamente libre, y desde su sitio web oficial podemos descargar la versión que más se adecue al sistema sobre el cual trabajaremos en el desarrollo de nuestras aplicaciones.

Conceptos del Modelo Relacional

Al momento de definir la estructura de una base de datos, deberemos especificar tablas y, dentro de ellas, columnas. Esto no es exclusivo de MySQL, sino del Modelo Relacional que es común a casi todos los motores de bases de datos. El lenguaje que se utiliza tanto para crear como para manipular todos los aspectos conectados con una base de datos relacional se denomina SQL (*Structured Query Language, Lenguaje de Consulta Estructurado*).

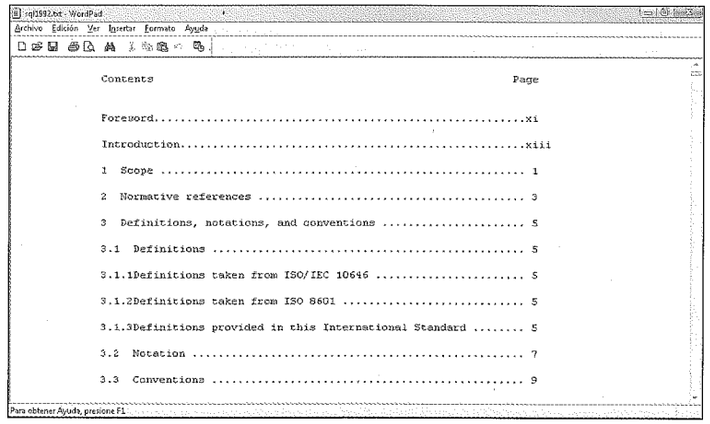

***Figura 3.** SQL es el lenguaje de consulta y manipulación de bases de datos relacionales más comúnmente utilizado en la actualidad.*

Podemos entender una base como una estructura organizada de datos relacionados entre sí. El lenguaje para obtener información y modificar se llama SQL.

Debemos decir que la idea fundamental detrás de la composición de las tablas de una base de datos y de las relaciones entre ellas es plasmar de manera pragmática distintas ideas que se desprenden de las características del sistema que queremos implementar. Para un catálogo de productos, podríamos tener una tabla para los productos (con campos: nombre, código, descripción, imagen asociada, precio, código de categoría, etcétera), y otra para categorías (código, nombre, descripción). Todo lo relativo a la normalización de bases de datos relacionales supera los alcances de este libro, aunque debemos remarcar que es un tema complejo, necesario y apasionante, sobre el cual hay mucha información disponible en Internet

MySQL, al igual que otras muchas aplicaciones para generar y administrar bases de datos, implementa el llamado modelo Relacional, en el que se instalan dos conceptos importantes que tenemos que conocer: el de tabla, el de fila, y el de campo.

Se recuperan los datos por medio de lenguajes de consulta (el más popular es SQL, pero existen otros) que mantienen la compatibilidad aun entre sistemas gestores de bases de datos de distintas compañías o sistemas operativos.

Una tabla es una colección de una o más columnas, también llamadas atributos o campos, y cero o más filas, también llamadas tuplas o registros. Por ejemplo, dentro de una tabla llamada Clientes podríamos tener una columna denominada Apellido y en la primera fila el valor Moreno, en la segunda fila Conte, etcétera. Los atributos pueden verse como características de las entidades (tablas).

Cada fila de cada tabla debe ser distinta, principalmente por motivos vinculados a la no inserción de datos redundantes ya que no tendría sentido incluir información duplicada en una misma tabla. Para evitar esto, generalmente se crean dentro de una tabla claves primarias. Una columna podría convertirse en clave primaria si tenemos la certeza de que sus valores nunca van a repetirse: por ejemplo, en la tabla Clientes la columna Apellido nunca podría ser una clave primaria puesto que existe la posibilidad de que haya dos personas con el mismo apellido. Si no contamos con una columna con estas características, lo que usualmente se hace es crear claves ficticias: por jemplo, podríamos crear además de Apellido, una columna llamada idCliente a la que daríamos intencionalmente valores diferentes para representar sin equívocos, a cada cliente.

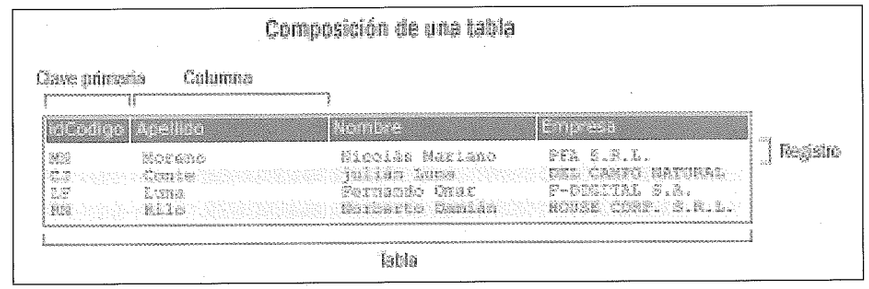

***Figura 4.** Diagrama de composición de las partes que conforman una tabla: columnas, registros, y clave primaria.*

En la sentencia SQL, el ejemplo del código para crear una tabla es el siguiente:

```
create table Cliente (
  idCliente primary key not null,
  Apellido varchar(100),
  Nombres  varchar(120),
  Empresa  varchar(60)
);
```

En breve, en este capítulo, veremos cómo ejecutar instrucciones SQL. Además, podemos combinar dos o más columnas para formar claves primarias. Las columnas que se definan como claves primarias no podrán admitir valores nulos.
Las tablas pueden vincularse entre sí, por, ejemplo, si contáramos en una misma base de datos con una tabla llamada Países y con otra llamada Provincias, podríamos vincularlas si dentro de Provincias hubiera un campo llamado codigoPaís, por caso. Este modelo admite relaciones uno a varios (un mismo país puede contar con más de una provincia), uno a uno, y varios a varios.

Acceso desde PHP

Para que una aplicación escrita en PHP pueda hacer uso y acceder a los datos almacenados en una base MySQL, deberemos asegurarnos de que el soporte esté activado, y para esto podemos remitirnos a la salida de la función **phpinfo** y buscar la sección correspondiente:

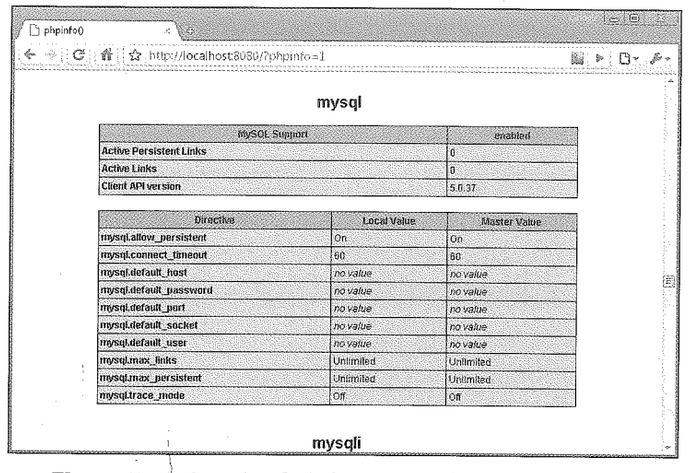

Figura 5. *La función phpinfo nos permite saber a ciencia cierta si el soporte para la extensión MySQL está disponible y activado.*

EL LENGUAJE SQL

Si bien SQL es un estándar mantenido por un ente oficial, cada servidor de bases de datos mantiene sus propias características. Aunque hay diferencias, son mínimas y, en general, si conocemos los basamentos del lenguaje, podremos comprender y asimilar rápidamente cómo trabajar con una base de datos en particular.
Abarcar cada punto del lenguaje SQL está fuera de los alcances de este libro, pero en los siguientes apartados observaremos cuáles son sus características principales y cómo realizar las tareas básicas necesarias para administrar una base de datos MySQL.

Tipos de datos en MySQL

Las columnas de cada tabla pertenecen a un tipo de dato específico. Si contamos con una tabla Empleados podríamos tener como campos **nombre**, **apellido**, **fecha de nacimiento**, **dirección**, **teléfono**, etcétera, y cada uno tendría asociado un tipo de dato particular. Veamos cuáles son los tipos de datos disponibles:

- Cadenas de caracteres
- Numéricos
- Fecha y hora

Cadenas de caracteres

Se utilizan para aquellas columnas que deban permitir almacenar datos en formato texto y que no se restrinjan a números o a fechas. Los subtipos de datos existentes dentro de las cadenas de caracteres son:

- **CHAR**: se utiliza generalmente para textos cortos de hasta 255 caracteres, por ejemplo, para definir nombres, apellidos, números de documento, etcétera. Si se ingresa una cadena de más caracteres que el tamaño prefijado al definir la columna, la cadena se truncará al llegar al límite. Por defecto, las comparaciones son insensibles a mayúsculas y minúsculas.
- **VARCHAR**: es similar a CHAR y su utilización está más ampliada. Aquí también, si se ingresa una cadena de más caracteres que el tamaño prefijado al definir la columna, la cadena se truncará al llegar al límite. Por defecto, las comparaciones son insensibles a mayúsculas y minúsculas.
- **BLOB** (*Binary Large OBject*, Objeto Binario de Gran Tamaño): se utiliza para almacenar grandes cantidades de texto. Permite comparar dentro de su contenido y distinguir entre mayúsculas y minúsculas. Generalmente se usa para objetos binarios, o sea, cualquier tipo de datos o información, desde un simple archivo de texto hasta imágenes, archivos de sonido o video. Se subdivide en cuatro tipos distintos, cada uno con una capacidad máxima de almacenamiento propia:

DIVISIONES DE BLOB	LÍMITE EN CARACTERES
TINYBLOB	255
BLOB	65.535
MEDIUMBLOB	16.777.215
LONGBLOB	4.294.967.295

***Tabla 1.** Diversas categorías limitantes de datos para el tipo de dato **BLOB**.*

- **TEXT**: este tipo de dato se utiliza para almacenar grandes cantidades de texto. Permite comparar dentro de su contenido sin distinguir mayúsculas y minúsculas. Es utilizado generalmente para almacenar texto plano, como descripciones o cualquier clase de texto largo que supere los 255 caracteres, equivalente a campos Memo en algunas bases de datos. Se subdivide en cuatro tipos distintos, cada uno con una capacidad máxima de almacenamiento propia:

DIVISIONES DE TEXT	LÍMITE EN CARACTERES
TINYTEXT	255
TEXT	65.535
MEDIUMTEXT	16.777.215
LONGTEXT	4.294.967.295

***Tabla 2.** Diversas categorías limitantes de datos, para el tipo de dato **TEXT**.*

- **ENUM**: es un tipo de string que puede seleccionar su valor, únicamente, de una lista de elementos definidos por el usuario, y como máximo es posible definir 65.535 elementos:

```
colorProducto enum('rojo','verde', 'azul')
```

Hay otras dos opciones por defecto: **0** que significa error (la opción ingresada está fuera de las disponibles) y **null** (no se ingresó ningún dato).

- **SET**: es similar al anterior, sólo que aquí podemos seleccionar ningún valor o más de un valor de la lista (hasta 64 simultáneamente), y los muestra separados por comas. Si tomamos el ejemplo anterior, podríamos ingresar en la columna **colorProducto** más de una opción, por ejemplo, rojo y azul.

Numéricos

Todos los tipos numéricos pueden definirse con dos parámetros opcionales: **UNSIGNED** (impide que los campos numéricos acepten signo negativo, es decir, sólo se aceptarán el cero y los valores positivos) y **ZEROFILL** (completa con ceros a la izquierda hasta la longitud máxima):


```
codigoProducto int(4) unsigned zerofill
```

Dentro de este tipo de dato, contamos con estas opciones para definir las características de las columnas de nuestras tablas:

- **DECIMAL (o NUMERIC o DEC)**: recibe dos argumentos opcionales: el número de posiciones enteras y el número de posiciones decimales. Por defecto, el primero es igual a 10 (con un rango de –9.999.999.999 hasta 99.999.999.999 para números con signo) y el segundo es igual a 0. El número es almacenado internamente como una cadena de caracteres (un carácter por cada dígito).
- **INTEGER (o INT)**: recibe como argumento el número de dígitos que el campo puede tomar. Si se omite o se sobrepasa la capacidad especificada, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo (es la acción por defecto) va desde –2.147.483.648 a 2.147.483.647, y sin signo (UNSIGNED) va desde 0 hasta 4.294.967.295.
- **TINYINT**: recibe como argumento el número de dígitos que el campo puede tomar. Si se omite o se sobrepasa la capacidad especificada, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo (es la acción por defecto) va desde -128 a 127, y sin signo (UNSIGNED) va desde 0 hasta 255.
- **BIT**: es un TINYINT de un dígito.
- **BOOL**: es un TINYINT de un dígito.
- **MEDIUMINT**: recibe como argumento el número de dígitos que el campo puede tomar. Si se omite o se sobrepasa la capacidad especificada, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo (es la acción por defecto) va desde –8.388.608 a 8.388.607, y sin signo (UNSIGNED) va desde 0 hasta 16.777.215.
- **BIGINT**: recibe como argumento el número de dígitos que el campo puede tomar. Si se omite o se sobrepasa la capacidad especificada, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo (es la acción por defecto) va desde –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807, y sin signo (UNSIGNED) va desde 0 hasta 18.446.744.073.709.551.615.

OPCIONES

MySQL provee múltiples formatos de tablas para cada necesidad. Si bien el tema está fuera de los alcances de este capítulo, recomendamos al lector buscar información acerca de esto, ya que, sin duda, sumará conocimientos válidos y de gran ayuda a la hora de desarrollar aplicaciones.

Todas las funciones matemáticas de MySQL trabajan internamente con valores de este tipo. Tipo de dato ideal para definir claves en tablas de millones de registros.

- **SMALLINT**: recibe como argumento el número de dígitos que el campo puede tomar. Si se omite o se sobrepasa la capacidad especificada, se toma la cantidad máxima soportada por este tipo de dato. Si se define con signo (es la acción por defecto) va desde –32.768 a 32.767, y sin signo (**UNSIGNED**) va desde 0 hasta 65.535.
- **FLOAT**: sirve para definir números con coma, con menos precisión que DOUBLE. Recibe como argumento el número de posiciones enteras y el número de posiciones decimales. El rango de posibles valores va desde -3.402823466E+38 a -1.175494351E-38, 0, y desde 1.175494351E-38 a 3.402823466E+38. Tipo de dato óptimo para almacenar valores del tipo moneda.
- **DOUBLE** (o DOUBLE PRECISION o REAL): nos permite definir números con coma con más precisión que FLOAT. Recibe como argumento el número de posiciones enteras y el número de posiciones decimales. El rango de posibles valores va desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0, y desde 2.2250738585072014E-308 a 1.7976931348623157E+308.

Fecha y hora

Los subtipos de datos existentes para este tipo de dato son:

- **DATETIME**: permite almacenar fecha y hora a la vez. El formato por defecto es 'YYYY-MM-DD HH:MM:SS'. El rango soportado va desde '1000-01-01 00:00:00' a '9999-12-31 23:59:59'. Si al momento de ingresar un valor de este tipo definimos un valor no válido, se almacenará la fecha nula ('0000-00-00 00:00:00').
- **DATE**: permite almacenar una fecha y el formato por defecto es 'YYYY-MM-DD'. El rango soportado va desde '1000-01-01' a '9999-12-31'. Si al momento de ingresar un valor de este tipo definimos un valor no válido, se almacenará la fecha nula ('0000-00-00').
- **TIMESTAMP**: es un tipo de dato que combina fecha y hora con posiciones numéricas, y permite almacenar valores entre el **1 de enero de 1970** hasta el año **2037** inclusive. Las distintas opciones son las siguientes:

DEFINICIÓN	FORMATO
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

***Tabla 3.** Opciones de fecha y hora dentro de **TIMESTAMP**.*

Al igual que con **DATE** o **DATETIME**, si ingresamos un valor no válido se almacena un valor nulo. Dependiendo del formato definido, todos los valores deberán ser ingresados ya que si esto no sucede se reemplazán por ceros para completar los que faltan.

- **TIME**: se utiliza para trabajar sólo con horarios. El formato por defecto es **HH:MM:SS** (también soporta **HHH:MM:SS**) y el rango va desde -838:59:59 a 838:59:59. Si al momento de ingresar un TIME definimos un valor inválido (por ejemplo, minuto superior a 60) se almacenará la fecha nula.
- **YEAR**: se usa para representar años y su formato por defecto es YYYY, aunque también puede definirse como YY. El rango va desde **1901** hasta **2155**.

Funciones de agregado

Se aplican en un grupo de registros y devuelven un único valor. Se usan dentro de la cláusula **SELECT** y las más utilizadas son las siguientes:

CONSTANTE	DESCRIPCIÓN
AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Devuelve el número de registros de la consulta SELECT.
SUM	Devuelve la suma de todos los valores de un campo determinado.
MAX	Devuelve el valor más alto de un campo especificado.
MIN	Devuelve el valor más bajo de un campo especificado.

***Tabla 4.** Funciones estadísticas o de agregado en SQL.*

Operadores

Al momento de llevar a cabo una instrucción SQL del tipo **SELECT**, **DELETE**, **UPDATE**, **ALTER** (selección, eliminación, actualización, modificación) podemos incluir en ellas los llamados operadores de comparación.

```
select * from productos where idProducto = '1'
```

ÚLTIMAS VERSIONES

Desde la versión 4.1 de MySQL hasta las últimas disponibles, se han incluido numerosas mejoras que tienen que ver, más que nada, con el soporte brindado para el estándar SQL ANSI. Contar con estas versiones nos permitirá obtener el máximo beneficio en nuestras aplicaciones.

Las comparaciones pueden devolver tres valores: **0** (falso), **1** (verdadero), y **NULL**. Los operadores son los que siguen:

OPERADOR	DESCRIPCIÓN
=	Igualdad
<> o !=	Desigualdad
<=	Menor o igual que
<	Menor que
>=	Mayor o igual que
>	Mayor que
<=>	Trata los valores nulos como ceros

***Tabla 5.** Operadores de comparación para realizar determinadas operaciones sobre los datos de una base de datos.*

Otras opciones relacionadas son:

- **IS NULL**: devuelve verdadero si el valor evaluado es nulo.
- **IS NOT NULL**: devuelve verdadero si el valor evaluado no es nulo.
- **IN**: devuelve verdadero si el valor está contenido en una lista.
- **NOT IN**: devuelve verdadero si el valor no está contenido en una lista.

Dentro de los operadores lógicos se encuentran:

OPERADOR	DESCRIPCIÓN
NOT o !	Negación
AND o &&	Unión
OR o ||	Opción
XOR	Exclusividad

***Tabla 6.** Tabla con los operadores lógicos para especificación de filtros.*

OPCIONES PARA ADMINISTRAR MYSQL

Si bien es cierto que, como veremos en breve, PHP nos provee una serie de funciones para manipular todo lo relacionado con el acceso a datos en MySQL en lo que respecta a su contenido (base de datos), hay ocasiones en que nos será más cómodo acudir a aplicaciones externas para poder sacar provecho de ciertos beneficios incluidos en ellas, como ser el diseño de una base de datos y sus tablas.
Hay múltiples aplicaciones para acceder a la funcionalidad de nuestro servidor de bases de datos MySQL. A continuación, observaremos dos de ellas que son muy

distintas pero, a la vez, muy populares y utilizadas, cada una en su ámbito y ocasión particulares: el monitor de MySQL y phpMyAdmin. A la primera, se accede a través del prompt del sistema operativo y, a la segunda, desde un navegador web.

El monitor de MySQL

El llamado monitor de MySQL nos permite, por medio de la línea de comandos del sistema operativo, poder ejecutar instrucciones de manera sencilla y directa. Esta es una aplicación cliente que viene incluida en la distribución del servidor, y nos será útil para dar los primeros pasos con la base de datos, conocer a fondo su sintaxis y familiarizarnos con los mensajes de error del sistema.

Para ponerlo en marcha (el servidor deberá estar activo: ver el primer capítulo para más información), lo único que tenemos que hacer es abrir una **terminal Linux** o una **ventana DOS** en Windows. Luego deberemos ubicarnos en el directorio de ejecutables de MySQL (en nuestro caso podría ser **C:\wamp\mysql\bin**).

```
C:\wamp\mysql\bin> mysql -u nombreUsuario -pContraseña
```

Deberemos incluir la p (de password) inmediatamente antes de la contraseña, si no hemos definido un usuario, lo más probable es que nuestros datos de acceso sean a través del usuario root sin contraseña, por lo cual podríamos ingresar simplemente al incluir el nombre de la aplicación:

```
C:\wamp\mysql\bin> mysql
```

Otra opción para ingresar al monitor es la siguiente:

```
C:\wamp\mysql\bin> mysql -h nombreServidor
```

ADMINISTRADOR PROPIO

La empresa encargada de desarrollar MySQL mantiene su propio administrador, MySQL Administrator. Se trata de una herramienta basada en software de escritorio que se instala en equipos locales (no tiene interfaz web) y permite manipular todas las opciones relacionadas con los distintos servidores a los cuales estamos conectados.

nombreServidor es el servidor al que nos queremos conectary puede ser o bien su nombre o su dirección IP. Si se omite esta opción se intentará conectar por defecto la propia máquina desde donde estamos trabajando.

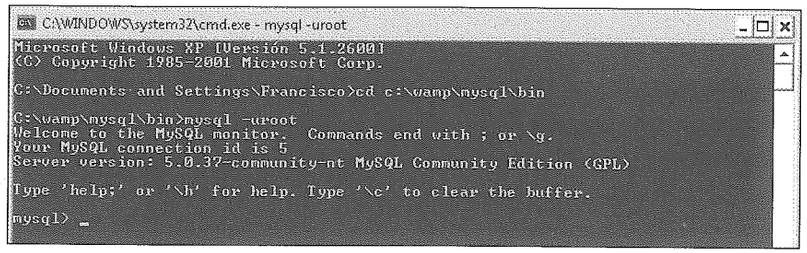

***Figura 6.** El monitor de MySQL nos permite ejecutar instrucciones SQL de manera directa y simple.*

Si todo funcionó bien, veremos un mensaje de bienvenida y el programa estará preparado y a la espera de nuestras instrucciones SQL.

Algunos consejos para situaciones frecuentes:

- Para ejecutar una instrucción, deberemos escribirla (finalizarla con ;) y presionar la tecla enter.
- Para cancelar una directiva contamos con la opción **\c**.
- Para cerrar el programa y volver tanto a DOS como al **prompt** de Linux, existen las directivas **quit** y **exit**.

```
MYSQL> EXIT
```

- Obtener un listado con las opciones disponibles (se guardará en el mismo directorio del programa, C:\wamp\mysql\bin):

```
MySQL -? > listado.txt
```

Si queremos conectarnos a una base ya existente deberemos utilizar el comando **use**:

```
MYSQL> USE nombreBase;
MYSQL> SELECT * FROM nombreTabla WHERE campo = 109;
```

Incluso podemos crear nuevas bases, si es que contamos con los permisos suficientes:


```
MYSQL> CREATE DATABASE nombreBase;
```

En definitiva, esta aplicación nos permite controlar, de manera sencilla, nuestros servidores de bases de datos. Si bien su interfaz puede no resultar del todo amigable, es de utilidad para acceder directamente a ejecutar instrucciones SQL y, en determinados casos, para acceder a una BD desde una PC en la que no tenemos instalado un administrador gráfico de MySQL, por ejemplo. Su uso esta extendido entre los usuarios de Linux, que se inclinan a mantener un control mas profundo y directo sobre MySQL.

phpMyAdmin

Esta aplicación es de las más utilizadas para administrar servidores MySQL, gracias a sus características (podemos controlar desde aquí casi cualquier situación vinculada al servidor) y, sobre todo, a su disponibilidad: phpMyAdmin es ni más ni menos que un sitio web corriente, por lo cual es multiplataforma y podemos acceder desde cualquier navegador. Dadas sus caracteristicas, esta aplicación se encuentra disponible en un gran número de paneles de control de servicios de alojamiento web. Podemos descargar una copia desde su sitio web oficial (en la dirección **www.phpmyadmin.net**) y copiar los archivos a nuestro **DocumentRoot**, o bien utilizar la disponible a través de WAMP, conjunto de aplicaciones para desarrollo de soluciones web (vista en el primer capítulo).

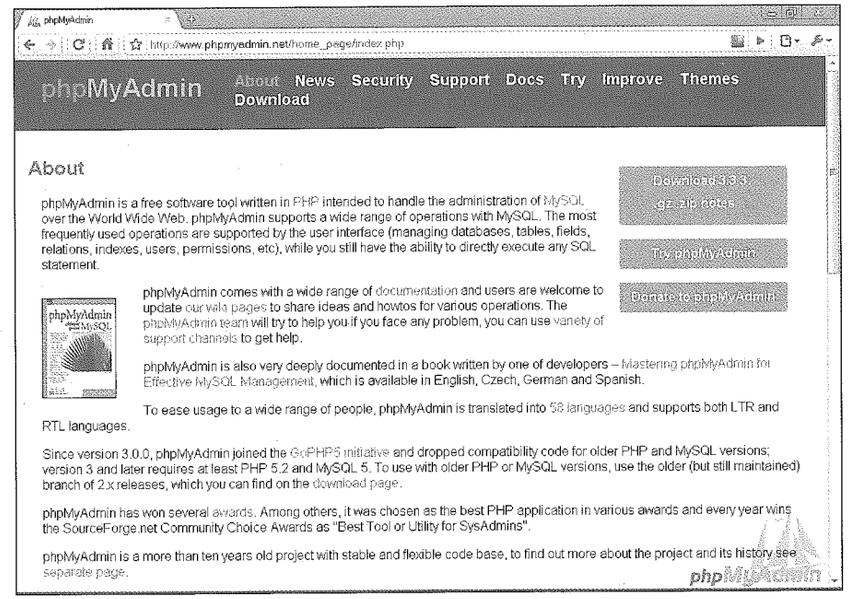

***Figura 7.** phpMyAdmin es una de las aplicaciones generalmente más utilizadas para crear y administrar bases de datos MySQL.*

En las versiones más recientes de **PHPMyAdmin** el proceso de instalación es sencillo: deberemos descargar la aplicación, descomprimirla en un directorio de nuestro servidor (accesible a través de un navegador web) y renombrar el archivo **config.sample.inc.php** a **config.inc.php**. Una vez hecho esto, tendremos que editar este archivo y asignar un valor cualquiera a la variable **blowfish_secret**:

```
$cfg['blowfish_secret'] = 'francisco123';
```

Luego podremos acceder a la aplicación:

```
http://localhost/directorioPhpMyAdmin/
```

Finalmente, ingresar el nombre de usuario (por defecto, **root**) y la contraseña (por defecto, vacía) correspondiente.

***Figura 8.** Desde WAMP tenemos la posibilidad de acceder directamente a phpMyAdmin, que viene incluido en la aplicación.*

Desde su página principal contamos con la pantalla dividida en dos sectores: a la izquierda, un listado con las bases de datos disponibles y a la derecha, una serie de opciones, entre ellas la de **Crear nueva base de datos**; para hacer esto deberemos ingresarle un nombre y presionar el botón **Crear**.

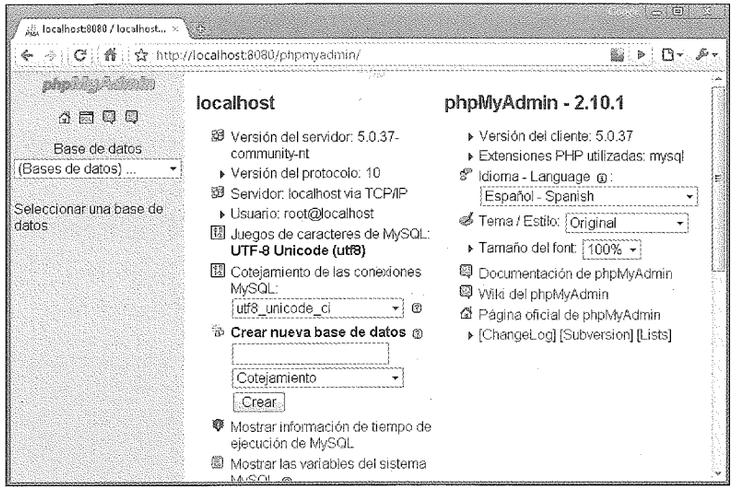

***Figura 9.** La interfaz de phpMyAdmin está dividida en dos sectores y compuesta por múltiples opciones para llevar a cabo distintas tareas.*

Ya sea al seleccionar una existente o al crear una nueva, accederemos a la pantalla de detalle de la base de datos en cuestión. Aquí veremos a la izquierda el listado de tablas de la base (puede no haber ninguna) y a la derecha, un menú de opciones:

- **Estructura** (muestra las tablas de la base).
- **SQL** (nos permite ingresar y ejecutar instrucciones SQL).
- **Buscar** (busca términos en las tablas).
- **Generar una consulta** (nos permite construir y ejecutar instrucciones SQL).
- **Exportar** (exporta el contenido de la base o de una tabla, por ejemplo, a un archivo)
- **Importar** (importa desde un archivo ya creado).
- **Operaciones** (distintas opciones como cambiar el nombre de la base de datos o copiarla a otro servidor).
- **Privilegios** (editar los privilegios de los usuarios y crear nuevos).
- **Eliminar** (elimina la base de datos actual).

Al ingresar en las opciones de una tabla en particular (por ejemplo, desde el menú de la izquierda), veremos nuevas opciones específicas para este tipo de estructuras:

- **Examinar** (muestra un listado de los registros de la tabla).
- **Estructura** (muestra los campos de la tabla).
- **SQL** (nos permite ingresar y ejecutar instrucciones SQL).
- **Buscar** (busca términos en la tabla) .
- **Insertar** (nos permite ingresar nuevos registros en la tabla).
- **Exportar** (exporta el contenido de la tabla, por ejemplo, a un archivo).

- **Importar** (importa desde un archivo ya creado).
- **Operaciones** (distintas opciones como cambiar el nombre de la tabla o copiarla a otra base disponible).
- **Vaciar** (elimina todos los registros de la tabla).
- **Eliminar** (elimina la tabla actual).

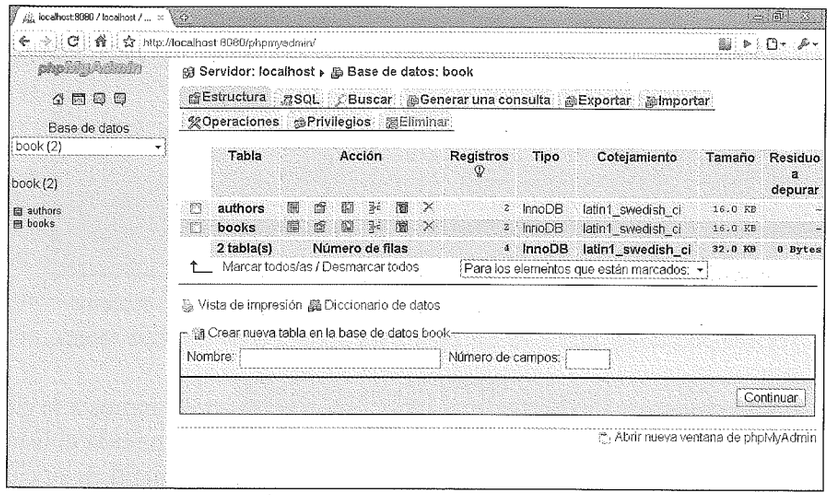

Figura 10. *Al momento de editar las opciones de una tabla, MySQL pone a nuestra disposición todas las alternativas en la ventana correspondiente.*

Algo que puede sernos de suma utilidad es que en la parte superior del sector derecho de la pantalla disponemos de una barra que nos permite saber dónde estamos ubicados en cada momento. Los componentes son:

```
Servidor > Base de datos > Tabla
```

Así, si no hemos seleccionado ninguna tabla, el último punto no aparecerá, y si no hemos seleccionado ninguna base, no aparecerán ni el segundo ni el tercero.

EVOLUCIÓN

Una de las mejoras constantes que MySQL va ofreciendo a lo largo del lanzamiento de sus nuevas versiones es el soporte para el estándar del lenguaje de consulta SQL. Por este motivo, es recomendable descargar las últimas versiones del motor de bases de datos.

A medida que nos vayamos familiarizando con las opciones de phpMyAdmin nos daremos cuenta de que hay múltiples maneras de realizar las mismas acciones, y esto le da flexibilidad a la aplicación.

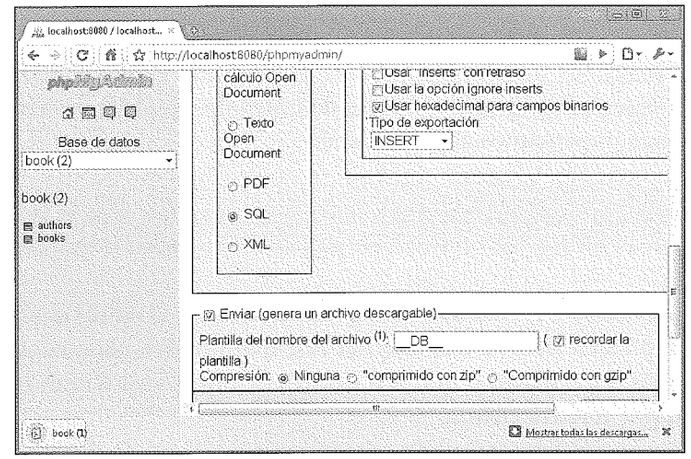

Figura 11. *A través de phpMyAdmin podemos exportar el contenido de una base de datos a un archivo, y luego importarlo desde el mismo servidor u otro distinto.*

Referencia de funciones

Lo primero que debemos hacer antes de comenzar a recuperar y manipular la información contenida en una base de datos es establecer una conexión con el servidor MySQL. Para hacerlo, PHP nos ofrece dos funciones que son similares aunque difieren en algunos aspectos: **mysql_connect** y **mysql_pconnect**. En el primer caso, la función **mysql_connect** requiere tres argumentos: nombre o dirección del servidor MySQL, nombre de usuario válido, y contraseña.
Veamos a continuación un ejemplo de conexión a una base de datos:

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);

if ($conexion) {
    echo "La conexion se establecio con exito.";
} else {
    echo "Error al intentar conectar.";
}
?>
```

El primer argumento debe contener el nombre del servidor o bien su dirección **IP**. Cuando hablamos de servidor nos referimos a la máquina en donde se encuentra instalado el servidor de bases de datos MySQL. Si estamos trabajando en forma local o si el servidor está instalado en la misma máquina en la cual están nuestras páginas, podemos utilizar como nombre de servidor **localhost** o bien la dirección IP **127.0.0.1**.

Opcionalmente, se puede agregar el puerto de conexión (desde la versión 3 de PHP) de la manera que presentamos a continuación:

```
$conexion = mysql_connect("localhost:80", $usuario, $password);
```

Esta función devuelve 1 (verdadero) si logró conectarse y 0 (falso) si algo falló.
En la práctica pueden ocurrir dos alternativas: que nuestro proveedor de hosting nos facilite todos estos datos, o que nos dé acceso a un panel de control para que los generemos por nuestra cuenta.
El nombre de usuario debe ser válido para acceder a la base de datos en particular, y la contraseña será la correspondiente al nombre de usuario ingresado.
Todos los argumentos **mysql_connect** son opcionales y, en caso de no incluir alguno de ellos, se asumen los valores por defecto que se encuentran en el archivo **php.ini**. Si observamos con atención este archivo podremos encontrar líneas como las siguientes:

```
mysql.default_port =
mysql.default_host =
mysql.default_user =
mysql.default_password =
```

Cada una significa respectivamente:

- puerto por defecto
- nombre del servidor por defecto
- nombre de usuario por defecto
- clave de usuario por defecto

Si tenemos acceso a este archivo podemos completar cada sección con nuestros valores y conectarnos a través de **mysql_connect** sin la necesidad de incluir los argumentos correspondientes. Por cuestiones de seguridad, esto no suele hacerse de hecho (en algunas configuraciones PHP directamente lo prohíbe).
Además, hay que tener en cuenta que los valores pasados a esta función tienen validez y preferencia aun cuando hayamos completado los valores en el archivo **php.ini**.

Los argumentos pasados a **mysql_connect** son opcionales desde el último hasta el primero, lo que significa que podemos omitirlos en ese orden:

- podemos no incluir ningún argumento
- podemos incluir sólo el nombre del servidor
- podemos incluir sólo el nombre del servidor y el nombre de usuario
- podemos incluir sólo el nombre del servidor, el nombre de usuario y la contraseña

En caso de no haber incluido algún argumento en **mysql_connect** y si además no contamos con un valor definido en el archivo **php.ini**, PHP acude a una tercera instancia y utiliza por su cuenta ciertos valores por defecto, que son los siguientes:

- nombre del servidor: localhost
- nombre de usuario: el del propietario del proceso
- contraseña: vacía

En cualquier caso, si logramos establecer una comunicación con el servidor MySQL, la función nos devuelve un identificador de conexión. Sin esto no podremos interactuar con la base de datos.

```
$identificador = mysql_connect($servidor, $usuario, $password);
```

La función **mysql_pconnect** (**P** de persistente) es idéntica, con la diferencia de que las conexiones abiertas no se cierran cuando termina la ejecución del archivo PHP. Si al llamar a esta función, ya existía una conexión abierta idéntica, se usa y, si no existía, se la crea. En general y para no sobrecargar al servidor con conexiones que ya no estén activas, suele utilizarse la función **mysql_connect**.
Las conexiones abiertas que utilizan la función **mysql_connect** se cierran automáticamente cuando finaliza el script. Si por algún motivo quisiéramos hacerlo de manera explícita, podríamos utilizar la función **mysql_close**, que recibe como argumento opcional un identificador de conexión:

```
<?php

// abrimos conexion ...
// interactuamos con la base ...

mysql_close();
?>
```

Esta función devuelve verdadero si tuvo éxito y falso si algo falló, por ejemplo, si el identificador no es válido o si no se había establecido ninguna conexión. La función **mysql_close** no cerrará la conexión si ésta fue abierta con la función **mysql_pconnect**. Aunque no es demasiado frecuente hacerlo, podemos crear bases de datos desde un script PHP. Para esto contamos con la función **mysql_create_db**, que recibe como argumentos el nombre de la nueva base de datos (no debe haber ninguna base con el mismo nombre del servidor) y un identificador de conexión. Esta función devuelve 1 (verdadero) si la instrucción pudo ejecutarse con éxito y 0 (falso) si algo falló.

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);
$base = mysql_create_db($nombreBase, $conexion);

if ($base) {
    echo "La base $nombreBase se creo con exito.";
} else {
    echo "Error al intentar crear la base $nombreBase.";
}

?>
```

En el caso de que queramos eliminar una determinada base de datos, contamos con la función **mysql_drop_db**, que recibe como argumentos el nombre de la base de datos y un identificador de conexión. Esta función devuelve 1 (verdadero) si la instrucción pudo ejecutarse con éxito y 0 (falso) si algo falló.

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);
$base = mysql_drop_db($nombreBase, $conexion);

if ($base) {
    echo "La base $nombreBase se elimino con exito.";
} else {
    echo "Error al intentar eliminar la base $nombreBase.";
}

?>
```


En el caso de que la base de datos a la que intentamos conectar ya esté creada, lo único que deberemos hacer será seleccionarla, esto es, indicar de manera precisa sobre cuál ejecutaremos las instrucciones SQL. Para esto tenemos la función **mysql_select_db**, que recibe como argumentos el nombre de la base de datos y un identificador de conexión.

```
mysql_select_db($nombreBase, $identificador);
```

Para entender cuál es la verdadera razón de esta función, debemos tener presentes dos cuestiones:

- En un mismo servidor MySQL pueden coexistir múltiples bases de datos.
- Dentro de un script, como veremos, podemos acceder a más de una base de datos.

Como observamos, el segundo argumento es un identificador de conexión. Este argumento es opcional y si se decide no incluirlo, la función asume la última conexión abierta con el servidor MySQL. Si no la encuentra, intenta crear una al invocar la función **mysql_connect**, pero sin pasarle argumentos. Esto vale tanto para **mysql_select_db** como para las demás funciones que requieren como argumento un identificador de conexión.

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);
mysql_select_db($nombreBase);

?>
```

Esta función devuelve **1** (verdadero) si logró conectarse y **0** (falso) si algo falló. Una vez seleccionada la base, el identificador de conexión quedará ligado a ella, por lo cual no deberemos volver a seleccionarla.
En este punto, ya estamos en condiciones de poder enviar instrucciones SQL al servidor para realizar distintas operaciones como recuperar registros de una tabla, modificarlos, eliminarlos, etcétera, por citar sólo algunos de los más comunes. Para esto contamos con dos funciones: **mysql_db_query** y **mysql_query**.
En el primer caso, la función **mysql_db_query** recibe tres argumentos: el nombre de la base de datos sobre la cual queremos realizar la consulta, la cadena que contiene la instrucción SQL a realizar (no deben terminar con punto y coma) y, por último, un identificador de conexión (opcional).

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

mysql_db_query($nombreBase, $sql, $conexion);

?>
```

Veamos algunos ejemplos de instrucciones SQL:

```
$sql = "select * from nombreTabla order by campo1";

$sql = "insert into nombreTabla (campo1, campo2) values ('9', '99')";

$sql = "update nombreTabla set campo1 = '10' where campo2 = '99'";

$sql = "delete from nombreTabla where campo2 = '99'";
```

En la extensión MySQL no contamos con una función específica para seleccionar, actualizar, o eliminar registros: para hacerlo debemos utilizar la función **mysql_db_ query** (o **mysql_query**, que veremos a continuación) y escribir la instrucción SQL que corresponda. Esta función devuelve **1** (verdadero) si la instrucción pudo ejecutarse y **0** (falso) si algo falló.
En cuanto a la función **mysql_query**, recibe como argumentos la cadena que contiene la instrucción SQL a realizar (no deben terminar con punto y coma), y un identificador de conexión (opcional).
Como podemos llegar a suponer, la diferencia entre las dos funciones reside en que con **mysql_db_query** podemos definir explícitamente sobre cuál base de datos ejecutaremos la instrucción (debemos recordar que desde un mismo script podemos conectarnos a más de una base de datos al mismo tiempo). En general, y por cuestiones prácticas, se suele utilizar la función **mysql_query**.

```
<?php

$conexion = mysql_connect($servidor, $usuario, $password);
```



```
mysql_select_db($nombreBase);

$nuevaTabla =  "create table nombreTabla (";
$nuevaTabla .= "  campo1 int(2) primary key not null,";
$nuevaTabla .= "  campo2 int(3)";
$nuevaTabla .= ") ";

$res = mysql_query($nuevaTabla);

if ($res) {
  echo "Tabla $nuevaTabla creada.";
} else {
  echo "Error al crear tabla $nuevaTabla.";
}

?>
```

Al igual que **mysql_db_query**, esta función devuelve 1 (verdadero) si la instrucción pudo ejecutarse con éxito y 0 (falso) si algo falló.
Con las funciones vistas anteriormente podemos enviar instrucciones al servidor. Las respuestas pueden variar según el caso: por ejemplo, al insetar un nuevo registro en una tabla, al modificar algún valor en una determinada columna o eliminando una tabla completa, la respuesta será verdadero si la instrucción pudo ejecutarse con éxito y falso si falló. Ahora bien, hay casos en los que necesitamos recuperar información de una base como los registros de una tabla. En estas situaciones contamos con una serie de funciones para obtener y recorrer el resultado devuelto por una consulta de selección.

Disponemos de tres clases de consultas y todas pueden ser ejecutadas desde PHP:

- Las simples, es decir, aquellas que se aplican sobre una única tabla.

```
select * from nombreTabla;
```

- Las multitabla, es decir, las que implican varias tablas a la vez, o sea que recuperan datos de varias fuentes.

```
select * from nombreTabla1, nombreTabla2 where nombreTabla1.id =
   nombreTabla2.idTabla1;
```

Podemos incluir más de dos tablas y, en general, todas deberían estar relacionadas a través de una de sus columnas.

- Las subconsultas, que posibilitan incluir más de una consulta de selección en la misma consulta.

```
select * from nombreTabla1 where id = (select idTabla2 from nombreTabla2);
```

No todas las versiones del servidor de bases de datos MySQL soportan subconsultas: éstas se implementaron a partir de la versión 4.1.
Más allá del tipo específico, veamos cómo recorrer las filas devueltas de una consulta. La extensión MySQL de PHP pone a nuestra disposición la función **mysql_fetch_array**, que recibe como argumentos un identificador de consulta y, opcionalmente, una constante para definir el tipo de índice:

- Un identificador de consulta es lo devuelto por alguna de las funciones que nos permiten enviar instrucciones SQL al servidor, es decir, tanto **mysql_db_query** como **mysql_query**.

- La constante para definir el tipo de índice tiene que ver con el hecho de que **mysql_fetch_array** nos permite recuperar filas y acceder a los valores de cada campo como si se tratara de una matriz. Las opciones son MYSQL_ASSOC, MYSQL_NUM, y MYSQL_BOTH.
  Este argumento es opcional y, si se opta por no incluirlo, se asume por defecto MYSQL_ASSOC. Deben escribirse con letras mayúsculas.

Antes de observar en detalle las diferencias entre cada opción, debemos saber que en cada llamada a **mysql_fetch_array** el puntero interno avanza una fila. Por ejemplo, en la primera llamada recuperamos la primera fila de resultados, en la segunda la segunda, y así sucesivamente. Cuando ya no hay más filas, la función devuelve falso:

```
<?php

$res = mysql_query("select * from nombreTabla");

while ($fila = mysql_fetch_array($res)) {
  // ...
  // ...
  // ...
```



```
}

?>
```

MYSQL_ASSOC toma al nombre de la columna como índice. Supongamos que la tabla **nombreTabla** está creada en la base de datos, tomemos el siguiente ejemplo:

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select campo1, campo2 from nombreTabla order by campo2
   desc");

while ($fila = mysql_fetch_array($res)) {
echo "<br /> ".$fila[campo1]." - ".$fila[campo2];
}

?>
```

Los nombres de campos son sensibles a mayúsculas y minúsculas: escribirlos correctamente dependerá de cómo los definamos al momento de crear las tablas.
La constante MYSQL_NUM nos permite utilizar como índice el número del campo (columna) según el orden dado en la consulta SQL. El índice 0 corresponde al primer campo. Veamos un ejemplo:

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
```

```
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select campo2, campo1 from nombreTabla");

while ($fila = mysql_fetch_array($res, MYSQL_NUM)) {
  echo "<br /> ".$fila[1]." - ".$fila[0];
}

?>
```

Si tomamos lo anterior, vemos que en cada iteración se muestra primero el valor de la columna **campo1** y si luego el valor de **campo2**. Si se seleccionan todos los campos (o sea, si no se especifican luego de la palabra **select**), se tendrá en cuenta el orden dado en la creación de la tabla.
Por último, contamos con **MYSQL_BOTH**, que nos permite utilizar como índice el número o el nombre del campo (columna). Es semejante a utilizar **MYSQL_ASSOC** y **MYSQL_NUM** al mismo tiempo. Observemos este ejemplo:

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select campo1, campo2 from nombreTabla");

while ($fila = mysql_fetch_array($res, MYSQL_BOTH)) {
  echo "<br /> ".$fila['campo1']." - ".$fila[1];
}
?>
```


Nuevamente, aquí debemos recordar que los nombres de campos son sensibles a mayúsculas y minúsculas: la forma correcta para escribirlos dependerá de cómo los hayamos definido al momento de crear las tablas.
Si realizamos una consulta sobre más de una tabla y por lo menos dos de los campos recuperados tienen el mismo nombre, sólo tendrá validez la última columna. Más adelante veremos más acerca de esta situación.
Como alternativa a **mysql_fetch_array** disponemos de **mysql_fetch_row** que, en realidad, es bastante similar ya que es semejante a utilizar **mysql_fetch_array** con la opción **MYSQL_NUM**. Veamos un ejemplo:

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select campo2, campo1 from nombreTabla");

while ($fila = mysql_fetch_row($res)) {
  echo "<br /> ".$fila[1]." - ".$fila[0];
}
```

El código anterior guarda similitudes con el ejemplo visto en **mysql_fetch_array**, ya que devuelve el mismo conjunto de resultados. Aquí también cada llamada a **mysql_fetch_row** devuelve la próxima fila del resultado (si es que hay una próxima) o falso si ya no quedan más filas.
Con la función **mysql_fetch_assoc** sucede algo parecido, puesto que es como utilizar **mysql_fetch_array** con la opción **MYSQL_ASSOC**. Para ejemplificar:

```
<?php

$servidor = "localhost";
```

```
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select campo1, campo2 from nombreTabla");

while ($fila = mysql_fetch_assoc($res)) {
  echo "<br /> ".$fila['campo1']." - ".$fila[1];
}

?>
```

En caso de que dos o más columnas tengan un mismo nombre (por ejemplo, una tabla **clientes** puede tener un campo **nombre** y una tabla **paises** también) lo que podemos hacer es diferenciarlos en la consulta:

```
$res = mysql_query("select clientes.nombre as nombreCliente, paises.nombre
   as nombrePais from clientes, paises where clientes.idPais =
   paises.idPais");
```

En el ejemplo suponemos que contamos con las tablas **clientes** y **paises**:

```
create table clientes (
  idCliente int(5) primary key not null,
  nombre varchar(255),
  apellido varchar(255),
  idPais int(2) not null,
  email varchar(255)
);

create table paises (
  idPais int(2) primary key not null,
  nombre varchar(255)
);
```


Ya en un script PHP, podremos referenciar las columnas según su alias:

```
while ($fila = mysql_fetch_assoc($res)) {
  echo "<br /> ".$fila['nombreCliente'];
}
```

Otra opción, en lugar los alias, es utilizar índices numéricos, como observamos en **mysql_fetch_array** y en **mysql_fetch_row**.
Si seguimos con las funciones para recorrer resultados tenemos **mysql_fetch_object**, que recibe como argumento un identificador de consulta. Esta función trata cada fila como si fuera un objeto y cada campo como si fuera una propiedad de éste. Observemos un ejemplo con respecto a esto:

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);

mysql_select_db($nombreBase);

$res = mysql_query("select * from nombreTabla");

while ($fila = mysql_fetch_object($res)) {
  echo "<br /> ".$fila->campo1;
}

?>
```

Cada llamada a **mysql_fetch_object** devuelve la próxima fila del resultado (si es que hay una próxima) o falso si ya no quedan más filas. Los nombres de campos son sensibles a mayúsculas y minúsculas: la forma correcta para escribirlos dependerá de cómo los hayamos definido al momento de crear las tablas. Si realizamos una consulta sobre más de una tabla y por lo menos dos de los campos recuperados tienen el mismo nombre, sólo tendrá validez la última columna. Una opción para resolver esto es utilizar alias en las consultas, como ya vimos.

La función **mysql_fetch_field** nos permite recuperar información sobre la estructura interna del campo, al trabajar con éste como si de un objeto se tratara. Veamos un ejemplo de aplicación para esta función:

```
<?php

$res = mysql_query("select * from nombreTabla");

while ($fila = mysql_fetch_field($res)) {
  echo "<br /> ".$fila->nombrePropiedad;
}

?>
```

Las propiedades (en el ejemplo anterior sería **nombrePropiedad**) disponibles son las que siguen:

- **name** (nombre de la columna).
- **table** (nombre de la tabla a la que pertenece la columna).
- **max_length** (longitud máxima de la columna).
- **not_null** (1 si la columna no puede contener un valor nulo, si no, 0).
- **primary_key** (1 si la columna es clave primaria, si no, 0).
- **unique_key** (1 si la columna es clave única, si no 0).
- **multiple_key** (1 si la columna es clave no única, si no, 0).
- **numeric** (1 si la columna es numérica, si no 0).
- **blob** (1 si la columna es un **blob**, si no, 0).
- **type** (el tipo de dato de la columna).
- **unsigned** (1 si la columna es sin signo, si no, 0).
- **zerofill** (1 si la columna se completa con ceros, si no, 0).

Cada vez que se llama a **mysql_fetch_field**, se recupera el siguiente campo disponible. Los nombres de la propiedades son sensibles a mayúsculas y a minúsculas, y deben ser escritos en la manera detallada.
Las funciones vistas siempre nos permiten avanzar al siguiente registro si es que hay alguno disponible. En caso de que queramos retroceder o posicionarnos en alguna fila en particular, podemos recurrir a la función **mysql_data_seek**, que recibe como argumentos un identificador de consulta y un número de fila.
En estos casos debemos tener en cuenta que a la primera fila de un conjunto de resultados le corresponde el valor 0. Luego de invocar **mysql_data_seek**, la llamada posterior a una de las funciones para recuperar una fila (**mysql_fetch_array**, **mysql_fetch_object**,

mysql_fetch_assoc, o **mysql_fetch_row**) nos devolverá la especificada en el segundo argumento. Esta función devuelve verdadero si tiene éxito y falso si ocurrió algún error, por ejemplo, si el número de fila especificado no existe.
Está dentro de las posibilidades que las consultas de selección que realicemos no devuelvan ningún registro. Para saber de manera precisa cuál es el número de filas recuperadas, contamos con la función **mysql_num_rows**, que recibe como único argumento un identificador de consulta.

```
<?php

// conexion ...

$res = mysql_query("select * from nombreTabla");
if ($total = mysql_num_rows($res)) {
  echo "Se recuperaron $total filas ...";

  while ($row = mysql_fetch_array($res)) {
    //...
  }
} else {
  echo "No se recuperaron filas ...";
}

?>
```

El resultado devuelto por esta función es similar al que podemos recuperar cuando realizamos una consulta como la siguiente:

```
$res = mysql_query("select count(*) as total from nombreTabla");
```

La gran diferencia está en que **mysql_num_rows** no precisa enviar una instrucción SQL extra al servidor y que, además, su sintaxis es más cómoda y nos permite escribir menos líneas de código.
Debemos tener en cuenta que esta función sólo sirve para sentencias SQL de selección, las de tipo **SELECT ... FROM**. Existe un función, en cierto sentido similar, denominada **mysql_affected_rows**, que nos permite recuperar el número de filas afectadas por una sentencia de tipo **INSERT**, **UPDATE** o **DELETE**. Recibe como único argumento un identificador de conexión, no de consulta. Tomemos un ejemplo de utilización en las siguientes líneas de código.

```
<?php

$servidor = "localhost";
$usuario = "root";
$password = "";
$nombreBase = "nombreBase";

$conexion = mysql_connect($servidor, $usuario, $password);
mysql_select_db($nombreBase);

$res = mysql_query("insert into clientes values (11, 'John', 'Smith', 1,
    'jsmith@jsmith.com')");
echo mysql_affected_rows($conexion);
//número de registros ingresados

$res = mysql_query("update clientes set idPais = 1 where idCliente > 10");
echo mysql_affected_rows($conexion);
//número de registros actualizados

$res = mysql_query("delete from clientes where idPais = 2");
echo mysql_affected_rows($conexion);
//número de registros eliminados

?>
```

Hay una situación especial que surge en los casos en los que eliminamos todos los registros de un tabla, es decir, cuando no incluimos la cláusula **WHERE**, la podemos visualizar ejemplificada en la línea de código a continuación:

```
$res = mysql_query("delete from clientes");
```

FORMA DE TRABAJO

Algunas de las características de la extensión MySQL son: la posibilidad de realizar consultas al servidor de bases de datos, recuperar el resultado (el conjunto de filas) y mantener la información en memoria sin tener que recurrir nuevamente al servidor.

Aquí deberíamos recuperar, por medio de **mysql_affected_rows**, el número de registros eliminados en la consulta ejecutada anteriormente (todos los que había originalmente), pero sin embargo la función devuelve como resultado cero.
Por su parte, la función **mysql_num_fields** nos permite conocer el número de campos (columnas) devueltos por una instrucción **SELECT ... FROM**. Recibe como único argumento un identificador de consulta.

```
<?php

// conexion ...

$res = mysql_query("select a, b, c from nombreTabla");
echo mysql_num_fields($res); //3

?>
```

Para instrucción de tipo **SELECT * FROM** esta función devuelve el número total de columnas de la/s tabla/s involucradas.
Una de las funcionalidades más utilizadas por quienes desarrollan aplicaciones y utilizan MySQL es la de definir columnas autoincrementables en las tablas. Tomemos como base el siguiente ejemplo:

```
create table pedidos (
idPedido int(5) primary key not null auto_increment,
fechaPedido datetime,
idCliente int(5) not null,
estadoPedido smallint(2)
```

Al momento de definir la columna **idPedido** como clave primaria, agregamos la clausula **auto_increment**. Esto implica que cuando ingresemos nuevos registros podremos omitir el valor para esta columna, a la cual se le asignará de manera automática un valor no repetido:

```
insert into pedidos (fechaPedido, idCliente, estadoPedido) values ('2008
    07-26 12:29:44', 1, 1);
insert into pedidos (fechaPedido, idCliente, estadoPedido) values ('2008
    07-26 13:32:00', 2, 1);
```

```
insert into pedidos (fechaPedido, idCliente, estadoPedido) values ('2008
   07-26 14:55:37', 5, 1);
```

Internamente, MySQL mantiene el valor máximo de autoincremento, y para la siguiente inserción toma ese valor y lo aumenta en 1. Este valor máximo se mantiene aun cuando eliminemos el registro correspondiente.
En este sentido, la función **mysql_insert_id** nos permite conocer el valor correspondiente al último campo de autoincremento insertado. Esta función recibe opcionalmente como argumento un identificador de conexión. Observemos el ejemplo:

```
<?php

// conexion ...

$res = mysql_query("insert into pedidos (fechaPedido, idCliente,
   estadoPedido) values ('2008-07-26 12:29:44', 1, 1)");
$idPedido = mysql_insert_id();
echo "Valor asignado a idPedido: $idPedido";

$res = mysql_query("delete from pedidos where idPedido = $idPedido");

$res = mysql_query("insert into pedidos (fechaPedido, idCliente,
   estadoPedido) values ('2008-07-26 13:32:00', 2, 1)");
echo "Valor asignado a idPedido: ".mysql_insert_id();

?>
```

Las funciones **mysql_list_dbs** y **mysql_tablename** nos permiten conocer las bases de datos disponibles en el servidor al cual nos hemos conectado y las tablas de cada una de las bases. Ambas pueden trabajar en conjunto: **mysql_list_dbs** recibe como argumento opcional un identificador de conexión y **mysql_tablename** recibe un apuntador a una base (devuelto por **mysql_list_dbs**) y un índice para recorrer la lista de tablas. Ejemplifiquemos para comprender el trabajo de estas funciones:

```
<?php

// conexion ...
```



```
$res = mysql_list_dbs($conexion);

while ($indice < mysql_fetch_row($res))  {
  echo mysql_tablename($res, $indice);
  $indice++;
}

?>
```

Por su parte, la función **mysql_list_tables** nos permite obtener una listado de tablas de una base de datos (primer argumento). Opcionalmente, recibe como segundo argumento un identificador de conexión. Esta función también puede interactuar con **mysql_tablename**, que en este caso cumple un rol diferente:

```
<?php

// conexion ...

$res = mysql_list_tables($nombreBase, $conexion);

while ($indice < mysql_fetch_row($res))  {
  echo mysql_tablename($res, $indice);
  $indice++;
}

?>
```

En cuanto al manejo y control de errores relacionados con MySQL, contamos con dos funciones básicas: **mysql_error**, que devuelve una descripción del ultimo error cometido y **mysql_errno**, que devuelve un código correspondiente al ultimo error cometido). Ambas reciben como argumento opcional un identificador de conexión.

```
<?php

// conexion ...
echo mysql_error();
echo mysql_errno();
?>
```

Por último, veamos una función muy importante que nos permite interactuar de manera segura con los datos almacenados en nuestras bases de datos. Se trata de **mysql_real_escape_string**, que recibe como argumento una cadena de caracteres y le aplica un procesamiento que evita en gran medida la posibilidad de que código dañino se ejecute a través de las instrucciones SQL. Esto tiene sentido si pensamos que muchas instrucciones incluyen datos ingresados por los usuarios de nuestras aplicaciones, a través de formularios, por citar sólo un caso. Este método de ataque es conocido como **SQL Injection** y, por eso, desde MySQL siempre aconsejan aplicar esta función a todos los datos a ser incluidos en las instrucciones que utilizaremos en el desarrollo de nuestras aplicaciones. Veamos un ejemplo:

```
<?php

// conexion ...

// valores definidos por el usuario:
$nombre = mysql_real_escape_string($_POST['nombre']);
$apellido = mysql_real_escape_string($_POST['apellido']);
$email = mysql_real_escape_string($_POST['email']);

mysql_query("insert into usuarios (nombre, apellido, email) values
    ('$nombre', '$apellido', '$email')");

?>
```

En este caso, las variables son accesibles por medio del método **POST**, pero la función debería utilizarse en todos los casos en los que intervengan datos ingresados por los usuarios. Asimismo, deberían escaparse las variables en toda clase de instrucciones: **SELECT ... FROM**, **INSERT**, **UPDATE**, **DELETE**, etcetera.

Ejemplos prácticos

A continuación, y para ilustrar algunas de las funciones vistas hasta aquí, desarrollaremos dos ejemplos prácticos que nos permitirán empezar a descubrir cómo trabajar e incluir MySQL en nuestros proyectos reales y cotidianos, directamente desde nuestro lenguaje de programación PHP.

Ejemplo #1 – Registro de usuarios

En el primer caso, se trata de un sistema de registro de usuarios. La estructura de la base de datos que utilizaremos es la siguiente:


```
CREATE DATABASE `ejemplo1`;

USE `ejemplo1`;

CREATE TABLE `usuarios` (
  `idUsuario` int(4) NOT NULL auto_increment,
  `nombreUsuario` varchar(255) default NULL,
  `apellidoUsuario` varchar(255) default NULL,
  `emailUsuario` varchar(255) default NULL,
  `passwordUsuario` varchar(255) default NULL,
  `ultimoIngresoUsuario` datetime default NULL,
  PRIMARY KEY  (`idUsuario`)
);
```

Para instalarla en nuestro servidor podríamos acceder a la aplicación PHPMyAdmin, sección SQL, e incluir el código anterior.

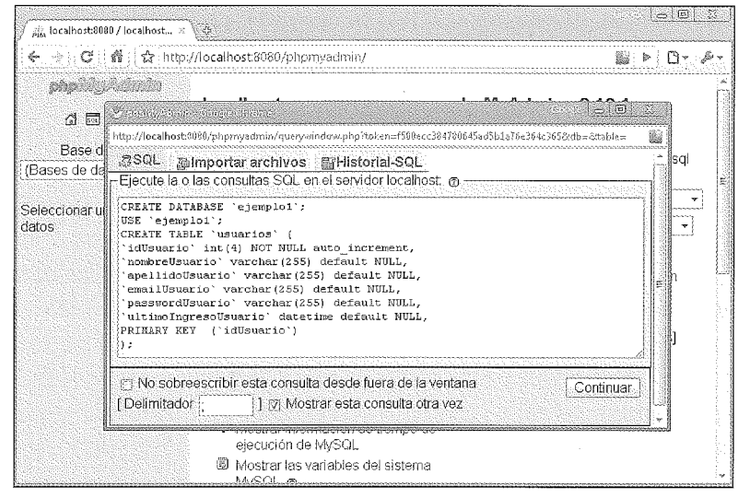

***Figura 12.** PHPMyAdmin nos posibilita acceder a la funcionalidad de un servidor de bases de datos de manera flexible, al permitirnos llevar a cabo las tareas más habituales.*

Ubicaremos lo relacionado con el establecimiento de la conexión en un archivo al que llamaremos **db.php**, cuyo contenido es el siguiente:

```
<?php
```

```
$hostname = "localhost";
$database = "ejemplo1";
$username = "root";
$password = "";

mysql_connect($hostname, $username, $password) or die('Error al conectar
    con el servidor MySQL');
mysql_select_db($database) or die('Error al conectar con la base de datos
    '.$database);

?>
```

Este archivo será incluido al comienzo de nuestras páginas, por lo menos aquellas que interactúen con la base de datos:

```
<?php

include 'db.php';

// ...

?>
```

En nuestro archivo principal, llamado **index.php**, contamos con las cuatro opciones que señalamos a continuación:

- Registrarme.
- Ingresar al sistema.
- Modificar mis datos.
- Terminar sesión.

RAPIDEZ

Una de las cuestiones que han influido notoriamente en el éxito logrado por MySQL es la rapidez que alcanza para resolver instrucciones y devolver registros de consultas. Las grandes empresas necesitan brindar a sus usuarios una imagen robusta y rápida, todo de manera simultánea.

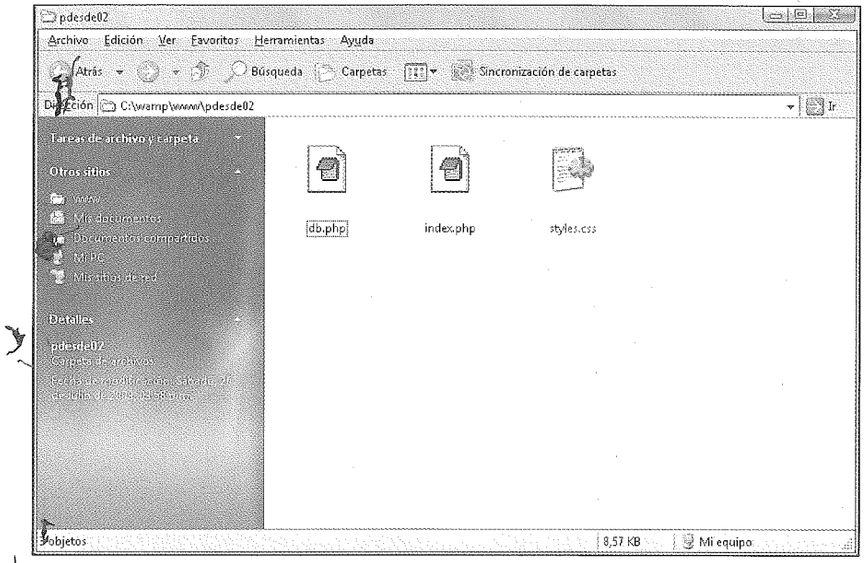

Figura 13. *Si existe la posibilidad de tener que reutilizar un código (por caso necesario, para establecer una conexión) es recomendable incluirlo desde un archivo externo.*

Depende de si el usuario está o no registrado (esto dependerá del valor de la variable de sesión **usuarioRegistrado**, que contendrá el valor del campo **idUsuario**) para disponer de una opción o de otra:

```
if ($_SESSION['usuarioRegistrado']) {
  $menu[1] = 'Modificar mis datos
     ['.$_SESSION['nombreUsuarioRegistrado'].']';
  $menu[2] = 'Terminar sesion';
} else {
  $menu[3] = 'Ingresar al sistema';
  $menu[4] = 'Registrarme';
}
```

El menú de opciones es generado a partir del contenido del array **menu**:

```
if (is_array($menu)) {
  foreach ($menu as $c => $v) {
    $menuHtml[] = "<a href='?accion=$c'>$v</a>";
  }
```

```
  echo implode(' | ', $menuHtml);
}
```

Por supuesto, antes de interactuar con sesiones deberemos iniciarlas con la función correspondiente al inicio del archivo:

```
session_start();
```

Veamos cuál es el proceso para agregar nuevos usuarios a la base de datos. Ya que el formulario es el mismo tanto para ingresar como para modificar datos de usuarios existentes, utilizaremos el mismo código:

```
if ($_GET['accion'] == '4' || $_GET['accion'] == '1') {
  // formulario
}
```

El código específico para generar el formulario es el siguiente:

```
echo <<<FORMULARIO
  <form method="post">
  <table cellspacing="0" align="center">
  <tr>
  <td>$obligatorio[frmNombreUsuario] Nombre</td>
  <td><input type="text" id="frmNombreUsuario" name="frmNombreUsuario"
     value="$_POST[frmNombreUsuario]"></td>
  </tr>
  <tr>
  <td>$obligatorio[frmApellidoUsuario] Apellido</td>
  <td><input type="text" id="frmApellidoUsuario" name="frmApellidoUsuario"
     value="$_POST[frmApellidoUsuario]"></td>
  </tr>
  <tr>
  <td>$obligatorio[frmEmailUsuario] E-mail (Username)</td>
  <td><input type="text" id="frmEmailUsuario" name="frmEmailUsuario"
     value="$_POST[frmEmailUsuario]"></td>
  </tr>
  <tr>
```



```
  <td>$obligatorio[frmPasswordUsuario] Password</td>
  <td><input type="password" id="frmPasswordUsuario"
     name="frmPasswordUsuario" value="$_POST[frmPasswordUsuario]"></td>
  </tr>
  <tr>
  <td>$obligatorio[frmRePasswordUsuario] Confirme Password</td>
  <td><input type="password" id="frmRePasswordUsuario"
     name="frmRePasswordUsuario" value="$_POST[frmRePasswordUsuario]"></td>
  </tr>
  <tr>
  <td colspan="2" align="right"><input type="submit" id="frmEnviarRegistro"
     name="frmEnviarRegistro" value="Enviar Formulario"
     class="submit"></td>
  </tr>
  </table>
  </form>
FORMULARIO;
```

Las variables de tipo **$obligatorio[campoFormulario]** contendrán una advertencia en caso de que no hayan sido completados por el usuario.
Ahora bien, el valor que tomen los campos del formulario dependerá de la situación: si estamos editando, los valores se toman de la base de datos (sobrescribimos el array **POST**) y si el formulario ha sido enviado, tomamos los datos ingresados por el usuario, como podemos ver en el ejemplo a continuación.

```
if (!count($_POST) && $_GET['accion'] == '1' &&
   $_SESSION['usuarioRegistrado']) {
  $res = mysql_query("select * from usuarios where idUsuario =
     '$_SESSION[usuarioRegistrado]'");
  if (mysql_num_rows($res)) {
    $row = mysql_fetch_array($res);

    $_POST['frmNombreUsuario'] = $row['nombreUsuario'];
    $_POST['frmApellidoUsuario'] = $row['apellidoUsuario'];
    $_POST['frmEmailUsuario'] = $row['emailUsuario'];
    $_POST['frmPasswordUsuario'] = $_POST['frmRePasswordUsuario'] =
       $row['passwordUsuario'];
  }
}
```

```
  foreach ($_POST as $c => $v)     {
    $_POST[$c] = htmlentities(stripslashes($v), ENT_QUOTES);
  }
```

Una vez enviado el formulario, validamos que los campos no estén vacíos:

```
if ($_POST['frmEnviarRegistro']) {

  $campos[] = 'frmNombreUsuario';
  $campos[] = 'frmApellidoUsuario';
  $campos[] = 'frmEmailUsuario';
  $campos[] = 'frmPasswordUsuario';
  $campos[] = 'frmRePasswordUsuario';

  foreach ($campos as $v)                {
    if (trim($_POST[$v])==='')           {
      $obligatorio[$v] = '* ';
    }
  }

  if (count($obligatorio)) {
    $mensajeError = 'Por favor complete los campos obligatorios !';
  } else                              {
    foreach ($campos as $v)           {
      $$v = mysql_real_escape_string($_POST[$v]);
    }

    //...
  }
}
```

MULTIPLATAFORMA

Sin duda, la clave de la masividad lograda por MySQL tiene que ver, más allá del funcionamiento, con la posibilidad de ser instalada en múltiples sistemas operativos (Windows, Linux, MacOs, etcétera) de manera sencilla, algo determinante para que los desarrolladores dispongan de sus servicios sin complicaciones.

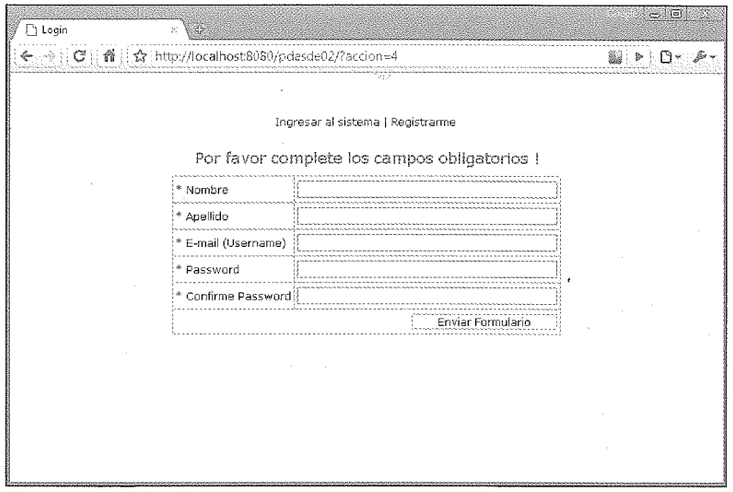

Figura 14. *Las variables de sesión nos permiten modificar el comportamiento de una aplicación para personalizar el acceso de los usuarios.*

Las otras validaciones básicas son:

- Que la dirección de correo no esté repetida, o sea, que no pertenezca a otro usuario. También se podría aplicar a otros datos, como ser el número de documento.

```
  if ($_SESSION['usuarioRegistrado']) { $sql = "and idUsuario <>
     $_SESSION[usuarioRegistrado]"; }

  $res = mysql_query("select * from usuarios where emailUsuario =
     '$frmEmailUsuario' $sql");
  if (mysql_num_rows($res)) {
    $mensajeError = 'Por favor complete ingrese otra direccion de
       E-Mail !';
    $obligatorio['frmEmailUsuario'] = '* ';
  }
```

- Que las contraseñas ingresadas coincidan.

```
  } elseif ($frmPasswordUsuario != $frmRePasswordUsuario) {
    $mensajeError = 'Las contraseñas no coinciden !';
    $obligatorio['frmPasswordUsuario'] = '* ';
    $obligatorio['frmRePasswordUsuario'] = '* ';
```

- Que la contraseña supere los cinco caracteres.

```
} elseif (strlen($frmPasswordUsuario) < 6) {
  $mensajeError = 'La contraseña debe ser tener por lo menos 6
     caracteres !';
  $obligatorio['frmPasswordUsuario'] = '* ';
  $obligatorio['frmRePasswordUsuario'] = '* ';
```

Si todas las validaciones se han superado, insertamos o actualizamos los datos del usuario de la siguiente manera:

```
if (!$_SESSION['usuarioRegistrado']) {
  $ultimoIngresoUsuario = date("Y-m-d H:i:s");
  mysql_query("insert into usuarios (nombreUsuario, apellidoUsuario,
      emailUsuario, passwordUsuario, ultimoIngresoUsuario) values
      ('$frmNombreUsuario', '$frmApellidoUsuario', '$frmEmailUsuario',
      '$frmPasswordUsuario', '$ultimoIngresoUsuario')");

  $_SESSION['usuarioRegistrado'] = mysql_insert_id();
} else {
  mysql_query("update usuarios set nombreUsuario =
      '$frmNombreUsuario', apellidoUsuario = '$frmApellidoUsuario',
      emailUsuario = '$frmEmailUsuario', passwordUsuario =
      '$frmPasswordUsuario' where idUsuario =
      $_SESSION[usuarioRegistrado]");
}

$_SESSION['nombreUsuarioRegistrado'] = "$frmNombreUsuario
   $frmApellidoUsuario";
```

SERVICIOS WEB

Un servicio es una aplicación que recibe una petición desde otra y devuelve una repuesta. El formato generalmente es XML, pero podría ser cualquiera. La aplicación que resuelve la petición puede, a su vez, obtener recursos desde otros orígenes, como por ejemplo, bases de datos.


```
header("location: ?");
exit;
```

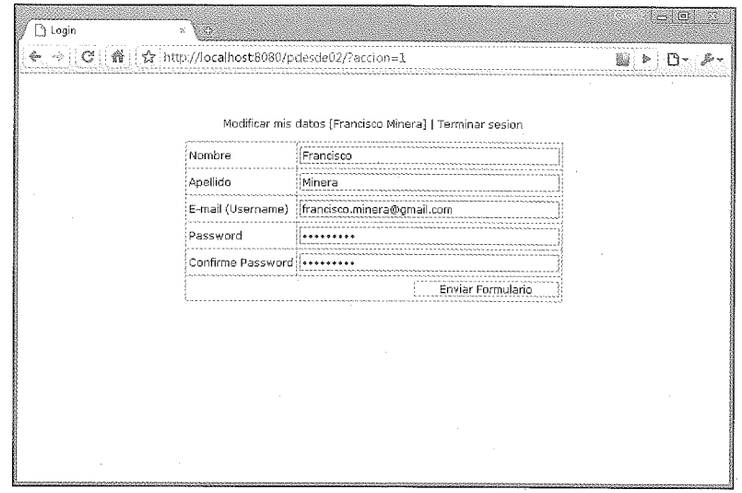

Figura 15. *La reutilización de código es una de las alternativas que se pueden lograr a través de lenguajes dinámicos como PHP.*

El proceso de ingreso al sistema de usuarios ya registrados es similar:

```
} elseif ($_GET['accion'] == '3') {

  if (count($_POST)) {
    foreach ($_POST as $c => $v)  {
      $_POST[$c] = htmlentities(stripslashes($v), ENT_QUOTES);
    }
  }

  echo <<<FORMULARIO
  <form method="post">
  <table cellspacing="0" align="center">
  <tr>
  <td>$obligatorio[frmEmailUsuario] E-mail (Username)</td>
  <td><input type="text" id="frmEmailUsuario" name="frmEmailUsuario"
     value="$_POST[frmEmailUsuario]"></td>
  </tr>
  <tr>
```

```
    <td>$obligatorio[frmPasswordUsuario] Password</td>
    <td><input type="password" id="frmPasswordUsuario"
       name="frmPasswordUsuario" value="$_POST[frmPasswordUsuario]"></td>
    </tr>
    <tr>
    <td colspan="2" align="right"><input type="submit" id="frmEnviarLogin"
       name="frmEnviarLogin" value="Enviar Formulario" class="submit"></td>
    </tr>
    </table>
    </form>
FORMULARIO;
}
```

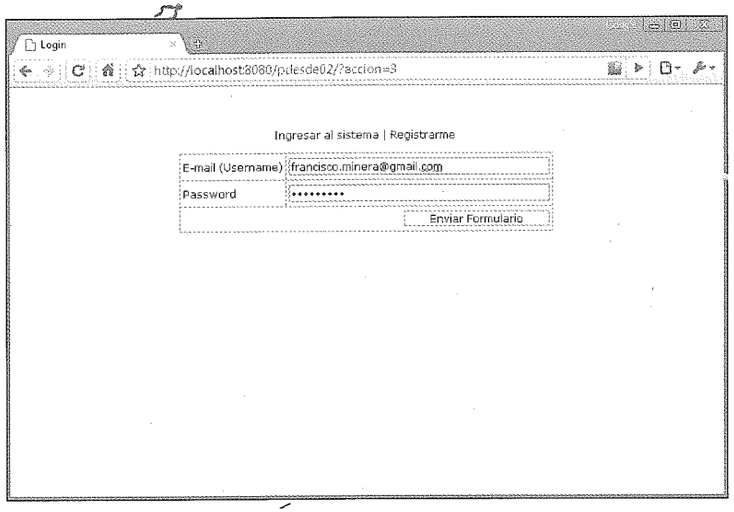

Figura 16. *El acceso restringido a contenidos de un sitio es algo habitual y fundamental para toda clase de aplicaciones.*

Una vez enviado, comparamos lo ingresado con los registros de la base de datos. Si hay coincidencia, permitimos el ingreso del usuario:

```
if ($_POST['frmEnviarLogin']) {
    // validaciones

    foreach ($campos as $v)        {
      $$v = mysql_real_escape_string($_POST[$v]);
    }
```



```
    $res = mysql_query("select * from usuarios where emailUsuario =
        '$frmEmailUsuario' and passwordUsuario = '$frmPasswordUsuario'");
    if (mysql_num_rows($res)) {
      $row = mysql_fetch_array($res);

      $_SESSION['usuarioRegistrado'] = $row['idUsuario'];
      $_SESSION['nombreUsuarioRegistrado'] = "$row[nombreUsuario]
          $row[apellidoUsuario]";

      $ultimoIngresoUsuario = date("Y-m-d H:i:s");
      mysql_query("update usuarios set ultimoIngresoUsuario =
          '$ultimoIngresoUsuario' where idUsuario =
          $_SESSION[usuarioRegistrado]");

      header("location: ?");
      exit;

    } else {
      $mensajeError = 'Datos incorrectos !';
      $obligatorio['frmEmailUsuario'] = '* ';
      $obligatorio['frmPasswordUsuario'] = '* ';
    }
}
```

Si todo resultó bien, mostramos un mensaje de bienvenida:

```
if ($_SESSION['usuarioRegistrado']) {

    $res = mysql_query("select * from usuarios where idUsuario =
        '$_SESSION[usuarioRegistrado]'");
    if (mysql_num_rows($res)) {
      $row = mysql_fetch_array($res);

      echo "<br /><center>Bienvenido $_SESSION[nombreUsuarioRegistrado],
          su ultimo ingreso fue el dia ".date("d/m/Y \a \l\a\s H:i",
          strtotime($row['ultimoIngresoUsuario'])."hs.</center>";
    }
}
```

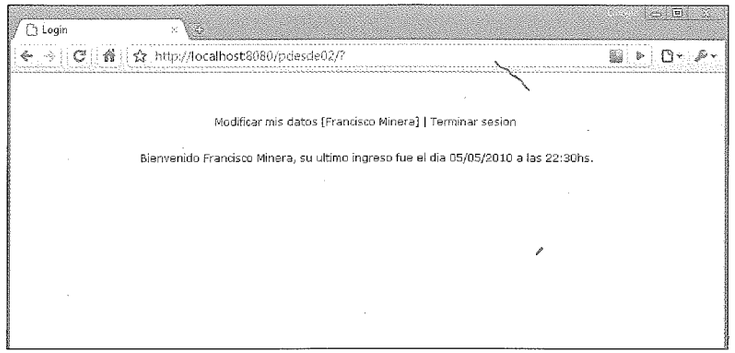

Figura 17. *El reconocimiento de usuarios es algo que sirve tanto para personalizar los contenidos como para mantener estadísticas de acceso.*

Finalmente, para terminar la sesión actual, ejecutamos el siguiente código:

```
if ($_GET['accion'] == '2') {
  session_unset();
  session_destroy();
  header ("location: ?");
  exit;
}
```

Con esto eliminamos todas las variables de sesión y volvemos al inicio.

Ejemplo #2 – Listado de productos

Para este ejemplo utilizaremos la siguiente base de datos:

```
CREATE DATABASE `ejemplo2`;

USE `ejemplo2`;

CREATE TABLE `productos` (
  `idProducto` int(5) NOT NULL auto_increment,
  `nombreProducto` varchar(255) default NULL,
  `descripcionProducto` text,
  `imagenProducto` varchar(255) default NULL,
  `precioProducto` double default NULL,
  `idCategoria` int(3) NOT NULL,
  PRIMARY KEY  (`idProducto`)
```



```
);

CREATE TABLE `productos_categorias` (
  `idCategoria` int(3) NOT NULL,
  `nombreCategoria` varchar(255) default NULL,
  PRIMARY KEY  (`idCategoria`)
);

INSERT INTO `productos_categorias` (`idCategoria`, `nombreCategoria`) VALUES
(1, 'Hogar y Muebles'),
(2, 'Celulares y Telefonía'),
(3, 'Libros, Revistas y Comics'),
(4, 'Computación'),
(5, 'Música, Películas y Series');
```

Como en el caso anterior, volvemos a contar con el archivo **db.php**:

```
include 'db.php';
```

La estructura del archivo principal (**index.php**) es la siguiente:

```
if ($_GET[mode]) {
  //mostramos el formulario para agregar / editar / eliminar productos
} else {
  //mostramos el listado de productos
}
```

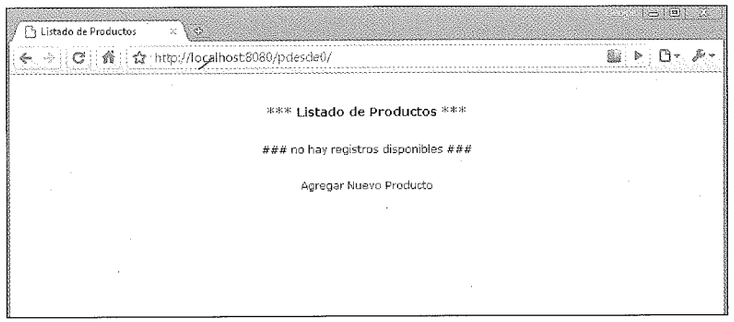

Figura 18. *A través de las funciones provistas por la extensión MySQL podemos generar aplicaciones funcionales e intuitivas.*

Veamos ahora el código para generar el listado:

```
$res = mysql_query("select * from productos, productos_categorias where
    productos.idCategoria = productos_categorias.idCategoria order by
    nombreProducto");
if (mysql_num_rows($res)) {

  echo "<table align='center' cellspacing='0'>";
  echo "<tr>";
  echo "<td>Imagen</td>";
  echo "<td width='400'>Nombre</td>";
  echo "<td>Categoria</td>";
  echo "<td class='options'>Editar</td>";
  echo "<td class='options'>Eliminar</td>";
  echo "</tr>";

  while ($row = mysql_fetch_array($res)) {
    echo "<tr>";
    echo "<td><img src='phpThumb/phpThumb.php?src=../
        imagenes/$row[imagenProducto]&w=90&h=90&far=C'/></td>";
    echo "<td>".$row['nombreProducto']."</td>";
    echo "<td>".$row['nombreCategoria']."</td>";
    echo "<td class='options'><a href='index.php?idProducto=
        $row[idProducto]&mode=e'>Editar</a></td>";
    echo "<td class='options'><a href='index.php?idProducto=
        $row[idProducto]&mode=d'>Eliminar</a></td>";
    echo "</tr>";
  }

  echo "</table>";

} else {
  echo "<center>### no hay registros disponibles ###</center>";
}

echo "<br /><br /><center><a href='?mode=a'>Agregar Nuevo
    Producto</a></center>";
```

Vemos que, para generar la miniatura (**thumbnail**) de la imagen, utilizamos la aplicación **phpThumb**, que explicamos en el capítulo dedicado al tratamiento de imágenes.

Todas las imágenes de los productos se almacenan en la carpeta llamada **imágenes**. En la tabla **productos** el campo **imagenProducto** contendrá el nombre del archivo.

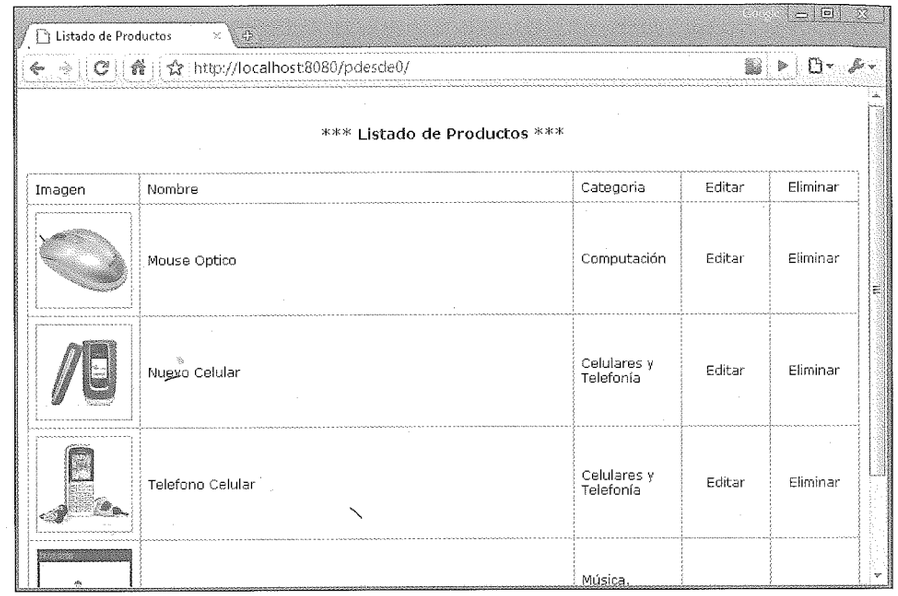

Figura 19. *La visualización de registros de una tabla puede lograrse por medio de unas pocas líneas de código.*

Si accedimos a alguna opción, esto es, si la variable **$_GET[mode]** tiene asignado algunos de los valores **a** (alta), **d** (baja), **e** (modificación), lo primero que hacemos es asignar las variables necesarias para luego asignarlas a los campos del formulario:

```
if (count($_POST)) {
  $array = $_POST;
  $imagenProductoGuardado = $array[imagenProductoGuardado];
} else {
  if ($_GET[idProducto]) {
    $res = mysql_query("select * from productos where idProducto =
        '$_GET[idProducto]'");
    if (mysql_num_rows($res)) {
      $array = mysql_fetch_array($res);
    }

    $imagenProductoGuardado = $array[imagenProducto];
  }
```

```
}

if (is_array($array)) {
  foreach ($array as $c => $v) {
    $array[$c] = htmlentities(stripslashes($v), ENT_QUOTES);
  }
}

$nombreProducto = $array[nombreProducto];
$descripcionProducto = $array[descripcionProducto];
$precioProducto = $array[precioProducto];
$idCategoria = $array[idCategoria];
```

En cuanto al formulario en sí, primero definimos los títulos para las opciones y el nombre del botón submit (de acuerdo con esto tomaremos una acción u otra al momento de enviar):

```
if ($_GET[mode] == 'a') $labelSubmit = 'Agregar';
elseif ($_GET[mode] == 'e') $labelSubmit = 'Guardar Cambios';
elseif ($_GET[mode] == 'd') $labelSubmit = 'Eliminar';

$cancel = 'Cancelar';

$nameSubmit = $_GET[mode].'_productos';

echo "<form method='post' action='?$_SERVER[QUERY_STRING]'
   enctype='multipart/form-data'>";
echo "<input type='hidden' name='imagenProductoGuardado'
   value='$imagenProductoGuardado'>";

// ...

echo "<tr>";
echo "<td></td>";
echo "<td><input class='submit' type='submit' name='$nameSubmit'
   value='$labelSubmit'> <input class='submit' type='button'
   value='$cancel' onclick='window.location=\"index.php\"'></td>";
echo "</tr>";
echo "</table>";
```



```
echo "</form>";
```

Incluimos un campo para almacenar el nombre de la imagen del producto: la idea es que si el usuario carga una imagen y envía el formulario, no tenga que volver a hacerlo en caso de que algún campo no haya sido completado.
Veamos ahora el código del formulario en el que se imprimen los controles para que el usuario complete las características del producto y que nos será de utilidad tanto para agregar como para editar ítems en nuestra base de datos:

```
echo "<table align='center' cellspacing='0'>";
echo "<tr>";
echo "<td>$error[nombreProducto]Nombre</td>";
echo "<td><input type='text' name='nombreProducto'
   value='$nombreProducto'></td>";
echo "</tr>";

if ($imagenProductoGuardado)    {
  $imagenProductoGuardado = "<br><img src='phpThumb/phpThumb.php?src=..
     /imagenes/$imagenProductoGuardado&w=200'/><br>";
}

echo "<tr>";
echo "<td>$error[imagenProductoGuardado]Imagen </td>";
echo "<td><input type='file' name='imagenProducto'>
   $imagenProductoGuardado </td>";
echo "</tr>";

echo "<tr>";
echo "<td>$error[descripcionProducto]Descripcion</td>";
echo "<td><textarea name='descripcionProducto'
   >$descripcionProducto</textarea></td>";
echo "</tr>";

echo "<tr>";
echo "<td>$error[precioProducto]Precio</td>";
echo "<td><input type='text' name='precioProducto' value=
   $precioProducto'></td>";
echo "</tr>";
$res = mysql_query("select * from productos_categorias order by
```

```
      nombreCategoria");
  if (mysql_num_rows($res)) {
    $select  = "<select id='idCategoria' name='idCategoria'>";
    $select .= "<option value=''></option>";
    while ($row = mysql_fetch_array($res))  {
      $selected = $idCategoria == $row[idCategoria] ? 'selected' : '';
      $select .= "<option $selected value='$row[idCategoria]'>
          $row[nombreCategoria] </option>";
    }
    $select .= "</select>";
  }

  echo "<tr>";
  echo "<td>$error[idCategoria]Categoria</td>";
  echo "<td>$select</td>";
  echo "</tr>";
```

Una vez enviado el formulario, depende del valor del botón submit el que tomemos una acción u otra:

```
if ($_POST["a_productos"] || $_POST["e_productos"] ||
   $_POST["d_productos"]) {
  if ($_POST["e_productos"]) {
    $idProducto = $_GET['idProducto'];
  } elseif ($_POST["d_productos"]) {
    $res = mysql_query("delete from productos where idProducto =
        '$_GET[idProducto]'");
    header ("location: index.php?msg=3");
  }
  // ... carga de la imagen
  // ... validaciones
  foreach ($_POST as $v) {
    $$v = mysql_real_escape_string($_POST[$v]);
  }
  if ($_POST["a_productos"]) {
    $res = mysql_query("insert into productos (nombreProducto,
        descripcionProducto, imagenProducto, precioProducto, idCategoria)
        values ('$nombreProducto', '$descripcionProducto',
        '$imagenProductoGuardado', '$precioProducto', '$idCategoria')");
```



```
    header ("location: index.php?msg=1");
  } elseif ($_POST["e_productos"]) {
    $res = mysql_query("update productos set nombreProducto =
        '$nombreProducto', descripcionProducto = '$descripcionProducto',
        imagenProducto = '$imagenProductoGuardado', precioProducto =
        '$precioProducto', idCategoria = '$idCategoria' where idProducto =
        $idProducto");

    header ("location: index.php?msg=2");
  }
}
```

Por supuesto que los ejemplos aquí planteados sirven sólo de muestra y necesitan ser trabajados tanto para personalizar su funcionalidad como para perfeccionar su comportamiento ante determinadas acciones: la idea es mostrar, en cierta medida, qué se puede lograr a partir de la utilización de herramientas tan versátiles como el lenguaje de programación PHP y la base de datos MySQL.

RESUMEN

En este capítulo, observamos las características principales de MySQL y nos enfocamos tanto en el aspecto teórico como en el práctico. Además, vimos cómo llevar a cabo las tareas más comunes que se dan al tener que mantener una base de datos, como ser, la creación de tablas, los distintos tipos de datos que se pueden almacenar en los campos y el acceso a una base de datos desde la línea de comandos. Tambien vimos otras características de MySQL que resultarán útiles al comenzar a trabajar con este motor en el desarrollo de aplicaciones web.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Qué es un servidor de bases de datos?

2 ¿Todas las aplicaciones web trabajan con bases de datos?

3 ¿Hay otros servidores de bases de datos además de MySQL?

4 ¿Alcanza únicamente con tener habilitada la extensión correspondiente para acceder a una base de datos? ¿Cuáles son los demás requisitos?

5 ¿Por qué cree que MySQL es uno de los servidores de bases de datos más utilizados en la actualidad?

6 Nombre tres características importantes de MySQL.

7 ¿Qué significa SQL?

8 ¿Qué es phpMyAdmin? ¿Para qué sirve?

9 Nombre algunos de los tipos de datos disponibles en MySQL.

10 ¿Para qué sirven los operadores?

EJERCICIOS PRÁCTICOS

1 Cree una base de datos llamada stock.

2 Cree una tabla llamada productos, que contenga los campos idProducto (entero, clave primaria), nombreProducto (texto, 255 caracteres), precioProducto (decimal, 3 enteros y 2 decimales) y descripcionProducto (texto largo).

3 Ingrese registros en la tabla productos.

4 Exporte la estructura y los datos de la tabla a un archivo.

5 Visualice los registros de la tabla.

Capítulo 8

XML

XML es un lenguaje de marcado, que posee una serie de características sobresalientes que serán estudiadas en este capítulo. Además, conoceremos cuáles son la alternativas disponibles a partir de las últimas versiones de PHP para su tratamiento e inclusión en aplicaciones web.

Introducción	**284**
XML, HTML y XHTML	285
Reglas de marcado	286
XML en PHP	**300**
SimpleXML	300
DOM	315
Interacción entre SimpleXML y DOM	331
Más extensiones	331
Resumen	**331**
Actividades	**332**