Manual Imprescindible de

# PHP5

Luis Miguel Cabezas Granado

Prólogo de Zeev Suraski y Andi Gutmans

ANAYA
MULTIMEDIA

MANUAL IMPRESCINDIBLE

RESPONSABLE EDITORIAL:
Eugenio Tuya Feijoó

ILUSTRACIÓN DE CUBIERTA:
Cecilia Poza Melero

REALIZACIÓN DE CUBIERTA:
Gracia Fernández-Pacheco

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2004
Juan Ignacio Luca de Tena, 15. 28027 Madrid
Depósito legal: M-44.920-2004
ISBN: 84-415-1785-1
Printed in Spain
Imprime: Artes Gráficas Guemo, S.L.
Febrero, 32. Madrid 28022

A mi mujer María Fernanda por creer
en nuestro Proyecto de vida.

# Agradecimientos

Finalizar este libro ha sido una odisea donde han intervenido muchas personas y factores, mudanza incluida.

Para empezar me gustaría agradecer a Marta Camarero y a Eugenio Tuya su fe en mí como escritor. Me dieron ánimos desde el principio y un poquito de prisa.

A Zeev Suraski y Andi Gutmans, creadores de PHP 5, por crear el lenguaje y tratarme como a uno de los suyos en los *¿seminar* de www.zend.com. Además, tengo que agradecerles la rapidez con la que escribieron el prólogo del libro.

A Marco Tabini, director de la revista canadiense PHParchitect, por enviarme el manual de Certificación PHP en papel y no en PDF.

A Tim Converse y Joyce Park por escribir *PHP5 and MySQL Bible,* cuyos conocimientos me bebí en apenas 3 días.

A Harry Fuecks por escribir el mejor libro de PHP de todos los tiempos *The PHP Anthology,* que me hizo reorientar todos los esfuerzos a la innovación.

A la Asociación Regional de Universidades Populares de Extremadura (AUPEX) por probar todos mis programas escritos en PHP (libros de visita, foros, nccproject, newton, etcétera). Espero dar algún día con la tecla.

A mis compañeros de trabajo Pako, Palomo, Javi y Pedro por las largas horas de discusión en torno a Delphi, Java, C# y, por supuesto, PHP.

A Justo Cabezas, porque todo lo que me regaló fueron libros de informática, y encima escritos por él. Sigo tus pasos.

A los grupos de usuario de gnuLinux de Extremadura www.sinuh.org y www.gulex.org, por ser una fuente inagotable de conocimientos.

Por último:

A mi padre Ramón, que ya no está, a mi madre Felisa y a mi hermano Felipe, porque nunca entendieron nada de informática, pero me animaron a seguir adelante con todo lo que me propuse.

Al resto de la familia, porque cada uno de ellos ha tenido algo que ver en mí vida y forman parte de lo que soy.

# índice


Cómo usar este libro ..... 19

Destinatarios de este libro ..... 20
Organización del libro ..... 20
Convenios que emplea este libro ..... 25
Los ejemplos en la Web de Anaya ..... 26

Prólogo ..... 27

Introducción ..... 29

Historia de PHP ..... 30
Nuevas Características de PHP 5 ..... 31
Fácil de usar ..... 31
Embebido en HTML ..... 32
Multiplataforma ..... 33
Licencia Open Source ..... 34
Multitud de Extensiones ..... 34
Velocidad e incorporación de objetos ..... 34
Popularidad ..... 35
Gran Comunidad de apoyo ..... 35
Objetivos del libro ..... *36*

1. Introducción a PHP 5 ..... 39

Introducción ..... 40

HTML estático 40
Tecnologías del lado del cliente 42
Tecnologías del lado del servidor 43
Etiquetas de PHP 45
Nuestro primer programa en PHP 5 46
Repaso de HTML 47
Cabecera y cuerpo de una página Web 47
Cabecera 48
Cuerpo del documento 48
Párrafos y saltos de líneas 49
Estilo de texto 49
Enlaces de texto 51
Listas 51
Imágenes 52
Tablas 53
Resumen 54

2 Variables, constantes y tipos de datos 55

Variables en PHP 5 56
Tipos de Variables 56
Asignación de variables 57
Tipos simples 57
Enteros (integer) 58
Números de coma flotante (double) 59
Cadena de caracteres (string) 59
Boolean 61
NULL 62
Variables de variables 62
Constantes 63
defined() 63
Constantes predefinidas 64
Funciones relacionadas con variables 65
isset() 65
unsetQ 66
gettype() 66
settype() 67
empty() 67
is_integer(), is_double(), is_string() 67
intvalQ, doublevalQ, strvalQ 68
Resumen 68

*3.* Operadores.....69

Introducción.....70
Operador de asignación.....70
Operador Unario.....71
Operadores Aritméticos.....71
Operadores de comparación.....71
Operadores Lógicos.....73
Operador Ternario.....74
Operadores bit a bit.....74
Operadores de asignación combinados.....75
Operador de ejecución.....*76*
Operador de supresión de errores.....*77*
Precedencia de Operadores.....78
Resumen.....80

4 Estructuras de control.....81

Introducción.....82
Estructuras de elección.....82
if-else.....82
elseif.....83
switch.....84
Bucles.....86
while.....86
do-while.....89
for.....89
break y continué.....92
Finalizar la ejecución de un programa.....93
Sintaxis alternativa.....93
Resumen.....94

*5.* Funciones.....95

Introducción.....96
Valores de las funciones.....96
Función de ejemplo. Obtención de la fecha actual.....97
Documentación sobre funciones.....98
Funciones de usuario.....99
Definición de funciones.....99
Parámetros insuficientes.....101
Parámetros en exceso.....102

Ámbito de las variables ... 102
Variables estáticas ... 104
Include() y require() ... 105
Recursividad ... 105
Funciones con número de argumentos variables ... 106
Argumentos por defecto ... 106
Argumentos mediante un array ... 107
Múltiples argumentos con func_num_args() ... 109
Llamadas por valor ... 110
Llamadas por referencia ... 111
Referencia a variables ... 112
Funciones variables ... 113
Resumen ... 114

6. Cadenas de caracteres y expresiones regulares ... 115

Introducción ... 116
Propiedades de las cadenas ... 116
índices de string ... 116
Operadores ... 118
Sintaxis para múltiples líneas ... 118
Funciones de string ... 119
Tamaño de la cadena ... 120
Posición de los caracteres ... 120
Comparación ... 121
Búsqueda de caracteres ... 122
Selección de subcadenas ... 122
Funciones de limpieza de cadenas ... 124
Sustitución de cadenas ... 126
Funciones de mayúscula y minúscula ... 126
Expresiones regulares ... 127
Comprobar expresiones regulares ... 129
Reemplazar patrones ... 131
Resumen ... 131

7. Conjuntos de datos del tipo array. ... 133

Introducción ... 134
Creación de arrays ... 134
Asignación directa ... 134
array() ... 135
Funciones que devuelven arrays ... 136

Arrays multidimensionales....136
Propiedades de arrays....137
count()....137
in_array()....137
Borrar ocurrencias....138
Interactuar con arrays....138
Funciones para avanzar en un array....140
Funciones para retroceder en un array....142
Intercambio de valores....143
Inversión del contenido....144
Mezcla de los valores....145
Pilas....145
Ordenación de los valores....146
Resumen....148

& Paso de información entre formularios....149

Introducción....150
Argumentos GET....150
Formularios con GET....151
Paso de información con GET....155
Argumentos POST....157
Variables súper-globales....158
Resumen....159

9. Programación orientada a objetos....161

Introducción....162
Definición de clases....163
Instancia de clase....164
Función constructor....165
Herencia....165
Métodos o funciones de objeto....166
Herencia encadenada....168
Valores y alcance de variables....168
Miembros públicos, privados y protegidos....170
Métodos privados....170
Métodos protegidos....171
Métodos públicos....171
Interfaces....172
Clases abstractas....172
Clases con métodos estáticos....173

Llamadas a funciones padre ... 174
Sobrecarga de métodos ... 176
Señalización ... 176
Funciones de manejo de clases ... 177
Resumen ... 179

10. Ficheros y almacenamiento de datos ... 181

Introducción ... 182
Funciones de lectura y escritura de ficheros ... 182
Abrir el fichero ... 182
Lectura de ficheros ... 183
Escritura de ficheros ... 185
Sistema de ficheros y directorios ... 188
Copiar, borrar y renombrar ... 188
Funciones de comprobación ... 189
Directorios ... 190
Ficheros de configuración ... 191
Manejo de ficheros en el servidor ... 193
Subida de ficheros ... 193
Descarga de ficheros ... 195
Resumen ... 197

11. Bases de datos con SQL y SQLite ... 199

Introducción ... 200
SQL ... 200
SELECT ... 202
Uniones ... 203
INSERT ... 205
UPDATE ... 206
DELETE ... 207
Definición de tablas ... 207
SQLite ... 208
Creación de bases de datos ... 209
Últimos cambios en una tabla ... 211
Selección de datos ... 212
SQLite orientado a objetos ... 213
Selección de registros ... 214
Funciones de Array para recuperar datos ... 215
Número de filas ... 216
Moverse entre registros ... 216
Resumen ... 218

12. PHP 5 y MySQL ..... 219

Introducción ..... 220
Administración de usuarios ..... 220
Conexión a MySQL ..... 220
Seleccionar datos ..... 222
Manipulación de datos ..... 224
Insertar una fila ..... 224
Actualizar una fila ..... 225
Borrar una fila ..... 225
Errores con las comillas ..... 226
Contando filas ..... 228
Contar filas con PHP ..... 228
Contar filas con MySQL ..... 229
Contar filas afectadas ..... 230
Último número insertado ..... 230
Búsquedas dentro de una tabla ..... 231
Definición de bases de datos ..... 231
Creación de bases de datos ..... 231
Creación de tablas ..... 231
Resumen ..... 233

13. Sesiones y Cookies ..... 235

Introducción ..... 236
Sesiones en PHP 5 ..... 237
Instanciando sesiones ..... 237
Variables de sesión ..... 239
Problemas con los navegadores ..... 240
Funciones para el manejo de sesiones ..... 241
Cookies ..... 243
setcookie() ..... 243
Borrar una cookie ..... 244
Cabeceras HTTP ..... 245
Resumen ..... 246

14 Lectura y escritura de archivos XML ..... 247

Introducción ..... 248
SAX, DOMySimpleXML ..... 250
SAX ..... 250
DOM ..... 254

Usar DOM para leer archivos ... 255
Todo es un objeto ... 255
Atributos ... 256
Búsquedas múltiples ... 257
Escribir archivos XML con DOM ... 257
Modificar archivos XML ... 259
SimpleXML ... 259
Resumen ... 261

15. Aplicaciones prácticas de XML ... 263

Introducción ... 264
Compartir información con RSS ... 264
Distintos formatos ... 264
Leer un archivo RSS ... 268
Escribir archivos RSS ... 269
Servicios Web XML-RPC ... 272
Clase IXR ... 273
Cliente XML-RPC ... 274
Servidor XML-RPC ... 275
Usos de XML-RPC ... 277
Resumen ... 277

16. Generación de gráficos con PHP 5 ... 279

Introducción ... 280
Gráficos HTML ... 280
Gráficos de barras ... 284
Librería GD ... 285
Tipos MIME ... 285
Mostrar una imagen en pantalla ... 286
Crear imágenes en miniatura ... 288
Generar una marca de agua ... 290
Gráficos estadísticos profesionales con JpGraph ... 292
Gráficos de barras ... 292
Gráficos en 3D ... 295
Resumen ... 297

17. Gestión de errores en PHP 5 ... 299

Introducción ... 300
Errores y Excepciones ... 300

La clase Exception 301
Bloque Try / Catch 303
Heredar de la clase Exception 304
Limitaciones de PHP 5 305
Control de errores sin excepciones 306
Errores nativos de PHP 306
Controladores de error 307
Errores de usuario con trigger_error() 308
Depuración de errores 308
Resumen 309

18. Conexiones desde PHP 5 311

Introducción 312
FTP 312
Mostrar los archivos remotos 314
Descargar y Enviar ficheros 315
Otras funciones de FTP 316
Correo electrónico 317
Enviar correo desde PHP 318
PHPMailer 319
Añadir un fichero adjunto 320
Resumen 322

19. Creación de archivos PDF 325

Introducción 326
Librería FPDF 326
Nuestro primer documento 327
Funciones de texto 328
Método Write() 328
Método Cell() 329
Desplazamiento de los cursores 330
Salto de página automático 330
Sobrescribir los métodos 331
Cabecera 331
Imagen de cabecera 332
Pie de página 333
Tablas 334
Enlaces 337
Resumen 338

20. Plantillas con Smarty ... 339

Introducción ... 340
Instalación de Smarty ... 341
Utilización básica de Smarty ... 341
Cuidado con los estilos CSS ... 345
Llamada a varias plantillas ... 346
Variables ... 346
Modificadores ... 347
Funciones ... *350*
foreach ... 350
if, elseif, else ... 351
php incluido en plantillas ... 351
assign ... 352
counter ... 352
cycle ... 352
Opciones avanzadas de Smarty ... 353
Plugins ... 353
Filtros ... 354
Resumen ... 355

Apéndice A. *Instalación de PHP 5 y* MySQL ... 357

Antes de comenzar ... 358
Instalación en MacOSX ... 358
Apache ... 358
PHP 5 ... 360
MySQL 4 ... 360
Comprobación final ... 361
Instalación en Windows ... 362
Instalación en gnuLinux ... 363
Recomendación final ... 364

Apéndice B. Configuración de php.ini ... 367

Introducción ... 368
short_open_tag ... 368
disable_functions ... 368
max_execution_time ... 368
error_reporting ... 368
register_globals ... 368
magic_quotes_runtime ... 369

include_path 369
Resumen 369
Apéndice C Bibliografía 371
Bibliografía 372
Libros de PHP 5 372
Revistas profesionales 372
Páginas Web 373
Glosario 375
índice alfabético 381

# Cómo usar este libro

## Destinatarios de este libro

Este libro está dirigido a usuarios noveles que no sepan programar en ningún lenguaje de desarrollo y a usuarios que ya conocen PHP, pero desean tener una base sólida sobre la nueva versión.

PHP 5 está ligado a Internet, por lo tanto, todas las aplicaciones prácticas del lenguaje están orientadas a realizar contenidos dinámicos para páginas. Por eso el lector debe conocer los aspectos básicos de diseño en HTML.

El libro puede dividirse en dos partes. Una primera en la que se detallan los conceptos principales del lenguaje como las variables, estructuras de control, manejo de cadenas de caracteres, creación de objetos y conectividad con bases de datos, que dará al lector novel una visión general de PHP 5 y le permitirá comenzar a desarrollar sus primeras aplicaciones.

La segundaparte, algo más compleja, hará las delicias del usuario novel y del medio. Esta cuenta con técnicas avanzadas para la lectura / escritura de archivos XML, creación de imágenes en tiempo de ejecución, conexiones de FTP y correo electrónico o diseño de plantillas con Smarty.

El Software Libre está presente en el ámbito de la programación. Por eso, podemos utilizar algunos programas libres como base para construir aplicaciones más complejas. El diseño de plantillas o la creación de llamadas a procedimientos remotos no sería posible sin las librerías Smarty o IXR para XML-RPC desarrolladas por la comunidad de PHP para su uso libre.

Por tanto, el libro va dirigido a usuarios noveles que no saben nada de PHP. Pero también a usuarios medios o avanzados que desean conocer las nuevas características de PHP 5 y manejar técnicas avanzadas de comunicación y diseño.

## Organización del libro

Este libro está dividido en 20 capítulos con los siguientes contenidos:

- Capítulo 1: Introducción a PHP 5: La introducción a PHP hace un recorrido por las nociones básicas de HTML. Además, aprenderá cómo crear su primera página en PHP y cómo mezclar el código HTML con PHP.
- Capítulo 2: Variables, constantes y tipos de datos: Los valores que manejamos en los programas deben ser almacenados en zonas de me-

moria reservadas por PHP. Estas zonas reciben el nombre de variables y pueden ser tratadas de diferentes forma dependiendo de los valores que guarden (números enteros, cadenas de caracteres, valores booleanos). Este capítulo contiene todo lo necesario para comprender cómo maneja PHP 5 las variables y las constantes, estableciendo una diferencia con otros lenguajes fuertemente tipados como Java o C.

- **Capítulo 3:** Operadores: Otro aspecto de los lenguajes de programación tiene que ver con los símbolos que se utilizan para realizar operaciones aritméticas, lógicas o de asignación. Cada uno de los símbolos recibe el nombre de operador. Este capítulo muestra por grupos todos los operadores con los que se puede encontrar en PHP 5. Así, podrá ver operadores especializados en sumas, restas, multiplicaciones o divisiones (aritméticos), operadores de igualdad, desigualdad o asignación (de comparación), incluso operadores de manejo a nivel de bits (binarios).
- **Capítulo 4:** Estructuras de control: El flujo de los programas viene determinado siempre por las estructuras de control. Estas indican en cada momento el rumbo que debe llevar el código y lo que debe mostrar nuestra página basándose en el valor contenido en las variables. Las estructuras de control y los operadores están íntimamente relacionados y, su unión, permite desviar *a* evolución *del programa en uno u otro* sentido. Las estructuras de control que verá en este capítulo le darán la potencia suficiente para crear sus primeros programas funcionales.
- **Capítulo 5:** Funciones: Si su código se hace muy extenso a medida que avanza en la comprensión del libro, en este capítulo aprenderá a remediarlo. Las funciones permitirán al lector agrupar fragmentos de código repetitivo y aislarlo en ficheros independientes. Este capítulo no sólo cubre los aspectos teóricos para la creación de funciones, sino que, además, es un compendio de buenas prácticas a tener en cuenta para que sus proyectos se desarrollen ordenadamente. Los aspectos básicos a tratar son la creación de funciones propias y el paso de parámetros entre funciones de varias formas diferentes.
- **Capítulo 6:** Cadenas de caracteres y expresiones regulares: La mayor parte de la información que puede encontrar en las páginas es texto. Conocer todas las técnicas para concatenar, medir, cortar y buscar textos es básico para crear un proyecto basado en contenido dinámico. Un periódico digital, un *Log* o, incluso, su propio CMS ya no tendrán secretos después de leer este capítulo. Además, la gran potencia de búsqueda de patrones viene de la mano de las expresiones regulares. Éstas

permiten encontrar palabras, frases y patrones dentro de un texto determinado para manipularlo a nuestro antojo.

- **Capítulo 7:** Conjuntos de datos del tipo array: Este capítulo trata a fondo las colecciones de datos. PHP 5 maneja los conjuntos de datos como *arrays* asociativos. Es posible acceder a los valores de *un array* de forma simple, utilizando un índice como en C, y de forma asociativa, asociando una palabra a un valor determinado dentro del *array.* Los *arrays* tienen un incalculable valor en los posteriores capítulos usándose como valor de retorno de muchas funciones de conectividad de bases de datos, lectura de archivos XML o creación de plantillas.
- **Capítulo 8:** Paso de información entre formularios: Si en el capítulo 1 pudo ver los conceptos básicos de la creación de páginas, en este podrá conocer las técnicas para enviar información a través de formularios. La mayoría de las páginas utilizan formularios para interactuar con los usuarios. Aprenderá a pasar variables *y arrays* entre dos páginas y cómo recoger desde PHP 5 los valores que se envían. Además podrá saber la diferencia entre los métodos GET y POST y las variables súper-globales.
- **Capítulo 9:** Programación orientada a objetos: El gran avance qué ha sufrido PHP 5 desde su versión anterior, ha sido la total incorporación de la sintaxis y técnica orientada a objetos. Esto permite desarrollar programas reutilizables en varios proyectos. Una de las ventajas de la programación orientada a objetos es que nos permite utilizar software de otras personas, simplemente conociendo los métodos que implementa; no es necesario conocer cómo funciona el objeto para utilizarlo. En este capítulo el usuario novel y medio aprenderá las nuevas características de la programación orientada a objetos de PHP 5. A partir del capítulo 9, casi todos los ejemplos estarán basados en esta metodología de trabajo.
- **Capítulo 10:** Ficheros y almacenamiento de datos: La forma más natural de almacenar datos persistentes es guardarlos en un fichero de datos. Es una tarea muy sencilla y no dependemos de la instalación de algún programa externo como una base de datos. El capítulo 10 mués-, tra cómo almacenar variables y textos dentro de un archivo, para después recuperarlo en nuestra página. Además aprenderá una serie de técnicas para recuperar ficheros enviados a través de un formulario y para poner en descarga archivos del servidor.
- **Capítulo** 11: Bases de datos con SQL y SQLite: Sin duda, la mejor forma de almacenar datos masivos es utilizar una base de datos. PHP 5 incorpora una API de gestión de bases de datos, SQLite, que permite almacenar millones de registros de diferentes tipos en un archivo binario. Lo

mejor de utilizar SQLite es que no necesita instalación de una base de datos, ni su administración, todo está contenido dentro de la API. Aunque este libro no está dedicado a la programación con bases de datos, se da una referencia de los usos *más* comunes del lenguaje de consultas estructurado (SQL) para que se pueda iniciar en este complejo mundo.

- **Capítulo 12:** PHP 5 y MySQL: MySQL ha sido siempre la panacea del desarrollo en el ámbito profesional. Es una base de datos robusta, que permite administración de usuarios y seguridad a nivel de tablas y celdas. Es algo compleja de administrar, pero muy sencilla de manejar desde PHP 5. Su utilización ha sido tal que en el mundo de desarrollo se habla de las páginas LAMP (Linux + Apache + MySQL + PHP). En este capítulo aprenderá a manejar las funciones más características de MySQL y a utilizarlas, por medio de objetos, en sus aplicaciones. La próxima aparición de MySQL 5 hace prever que la combinación con PHP será perfecta (MySQL 5 + PHP 5 = 10).

- **Capítulo** 13: Sesiones y Cookies: En este capítulo se detallan las técnicas necesarias para grabar información en el equipo de los usuarios. Estas pequeñas variables *(Cookies)* permiten crear un pequeño control de asistencia de sus usuarios o guardar configuraciones especiales. En cuanto a las sesiones, son muy necesarias a la hora de crear espacios de venta de productos. Las sesiones identifican a un usuario en concreto dentro de una página y permiten asociar variables concretas a ese usuario, aunque la página esté siendo visitada por muchas personas a la vez.

- **Capítulo 14:** Lectura y escritura de archivos XML: Frente a una inmensa cantidad de tipos de archivos propietarios como los documentos DOC o XLS, nos encontramos a una serie de archivos basados en el lenguaje XML. Este tipo de archivos, como SXW *(Openoffice)* o SVG (gráficos vectoriales), están creados para ser entendidos por las personas y por los ordenadores. Por lo tanto, crear programas que interpreten estos archivos no es nada complejo. El capítulo 14 muestra tres caminos distintos para leer archivos XML. Dos de ellos que vienen funcionando desde versiones anteriores, SAX y DOM. Y uno nuevo, implementado en PHP 5 para facilitar esta tarea al programados Además se incluyen las pautas necesarias para que cree sus propios archivos XML con la metodología DOM.

- **Capítulo 15:** Aplicaciones prácticas de XML: Si en el capítulo anterior sentábamos las bases para la lectura de información desde archivos XML, en este capítulo crearemos objetos que implementen aplicaciones reales *y* actuales para compartir información. Es muy habitual, hoy

en día, que las páginas lleven asociados un panel de noticias que se generan en portales diferentes. Esta información se puede mostrar gracias a técnicas como RSS.

Además de crear sus propios objetos de generación de contenido RSS y su lectura, haremos hincapié en lo que actualmente se llaman servicios. Los servicios son pequeños programas almacenados en algún servidor que informan acerca de algo concreto. Aquí aprenderá a crear sus servicios con las técnicas ya adquiridas en el capítulo 14.

- Capítulo 16: Generación de gráficos con PHP 5: La generación de gráficos desde PHP 5 es posible gracias a la utilización de librerías externas como GD. Esto le permite manipular archivos gráficos que sus usuarios puedan subir a través de un formulario. Con esta librería puede generar gráficos en miniatura para, después de hacer clic en él, añadir una marca de agua a todas las imágenes añadiendo el logo de su empresa o asociación.

  Otra gran librería es JpGraph, que permite crear todo tipo de gráficos estadísticos. Esto le servirá para conocer el número de visitas por día, mes o año, comparativas anuales de ventas de productos y todo un abanico de posibilidades.
- **Capítulo 17:** Gestión de errores en PHP 5: Una vez terminada la aplicación tendrá que tener en cuenta las partes del código donde puede tener problemas. Algunos usuarios pueden introducir caracteres no permitidos o números donde se esperaban letras. La gestión de errores permite mostrar al desarrollador y al usuario dónde aparece el error y el tipo de fallo que se ha producido. Una buena gestión de errores hará que su programa sea más fiable.
- **Capítulo** 18: Conexiones desde PHP 5: La conectividad de PHP con otras aplicaciones o servicios ha sido siempre una preocupación de sus desarrolladores. Así, desde anteriores versiones puede encontrar conectividad con servicios LDAP, POP3, SMTP o FTP.

  Este capítulo trata de abarcar dos servicios muy utilizados por la comunidad de usuarios. El FTP, en primer lugar, para enviar y recuperar información de otros servidores desde un programa escrito en PHP. Y, en segundo lugar, el manejo de las funciones necesarias para enviar información por correo electrónico.
- **Capítulo 19:** Creación de archivos PDF: El formato de texto más difundido en Internet es el PDF. Tanto se ha extendido este formato que la mayoría de las páginas profesionales contienen archivos PDF con diferentes contenidos. Desde páginas de contenido tecnológico hasta las de cocina

exportan sus contenidos en PDF para que sus usuarios puedan disfrutar desde su ordenador del manual del video o la receta de la semana.

En este capítulo verá paso a paso cómo generar un archivo PDF completo, desde la cabecera hasta el pie de página, pasando por todo tipo de textos e imágenes.

- Capítulo 20: Plantillas con Smarty: La última aportación de este libro tiene que ver con la tendencia de separar el código PHP de la presentación en HTML. Smarty provee unos sencillos mecanismos desde PHP que permiten realizar sencillamente esta separación. Además de las características básicas de las plantillas, la potencia de Smarty reside en la compilación del diseño cada vez que cambian los datos, hecho que otorga a Smarty el puesto número 1 en velocidad en el *ranking* de sistemas de plantillas. En este capítulo aprenderá a crear plantillas con Smarty, que permitirán cambiar el aspecto gráfico de su aplicación sin tocar el código escrito en PHP 5.
- **Apéndice A:** Instalación de Apache y PHP 5 en distintos Sistemas Operativos: La instalación de PHP 5 es muy sencilla en gnuLinux, MacOSX y Windows. El Apéndice muestra los pasos necesarios para instalar un servidor Apache con el parse de PHP 5 listo para ser utilizado con los ejemplos. En cada caso,, la instalación se hace de una forma distinta. Hemos buscado para cada Sistema Operativo la opción más sencilla, que es utilizando algún paquete compacto que se instale y deje listo el sistema. Puesto que es un libro para principiantes, hemos huido de crear PHP 5 desde el código fuente compilando el programa.
- **Apéndice** B: Configuración básica de php.ini: El archivo de configuración `php.ini` contiene aspectos de funcionamiento de PHP 5, que pueden cambiarse sin necesidad de compilar el programa. Este Apéndice contiene algunos parámetros de configuración que pueden ser útiles.
- **Apéndice C:** Bibliografía: Este libro está basado en el conocimiento adquirido tras la lectura de numerosos libros sobre la temática, páginas Web y revistas profesionales del sector. Este Apéndice cubre todas las posibilidades, dando al lector un camino a seguir después de terminar el presente libro.

## Convenios que emplea este libro

El nombre de los comandos, funciones, métodos u objetos aparecen resaltados en el libro de la siguiente forma: `objeto`.

Algunas palabras técnicas que no tienen una traducción fácil al castellano o simplemente está fuertemente asentada en el idioma técnico aparecen de la siguiente forma: *array, true ofalse.*

En el libro aparecen resaltados una serie de temas, circunstancias o acontecimientos extraordinarios de la siguiente forma: *array, true ofalse.*

**Nota:**

*Anotaciones sobre el texto.*

**Advertencia:**

*Información importante a tener en cuenta a la hora de desarrollar un programa. Explican de alguna manera posibles equivocaciones o ayudas para no caer en errores frecuentes.*

**Truco:**

*Consejo o información importante que puede facilitar el trabajo.*

## Los ejemplos en la Web de Anaya

La mayoría de los ejemplos del libro se pueden obtener en la Web de Anaya Multimedia, en la dirección siguiente:

http://www.anayamultimedia.es

Seleccionando los menús: Atención a! cliente>Complementos>Manuales Imprescindibles>Manual Imprescindible de **PHP** 5.

Además de los ejemplos puede encontrar las aplicaciones externas, o enlaces a las mismas, que utilizamos para desarrollar algunos ejemplos como: Smarty, FPDF, JpGraph o la librería IXR.

Todos los ejemplos han sido probados en el servidor Apache para MacOSX con el paquete de PHP 5.0.1 y en un servidor Apache sobre gnuLinEx. Todas las capturas de pantalla se han realizado con el navegador Grulla (Mozilla Firefox) de gnuLinEx 2004.


# Prólogo

Una de las preguntas más interesantes que nos hacen es qué hace que PHP tenga tanto éxito. Distintas personas nos han dado respuesta a esta pregunta. Algunos sostienen que es su excelente conexión con bases de datos, otros que se debe al código abierto; por el contrario otros mantienen que es su funcionamiento. Sin embargo, nuestra opinión difiere. PHP permite que cualquier usuario obtenga rápidamente resultados de forma sencilla, incluso aquellos que no posean experiencia en programación. Al mismo tiempo, a diferencia de las herramientas RAD, PHP permite el desarrollo de proyectos tan complejos como se desee. En nuestra opinión, esta combinación, ausente en la mayoría de las plataformas Web actuales, es lo que hace que PHP sea especial.

Lograr el equilibrio entre estas dos líneas (la potencia y la sencillez) no ha sido fácil, pero sí imprescindible para el éxito de PHP, como también lo son las nuevas características que permiten al desarrollador aprovechar la última tecnología.

PHP 5 es el producto de este importante equilibrio. Aunque dotado de muchas funciones nuevas, esta versión es tan accesible como lo fueron PHP 3 y PHP 4, con lo que permite un rápido aprendizaje para aquellos que se inicien. De hecho, algunas de las novedades basadas en la interfaz de Zend Engine II, como SimpleXML, SQLite y SOAP, hacen más sencillo el uso de PHP, al tiempo que incrementa el potencial de aquellos usuarios noveles, pues les permite trabajar con XML, SQL y Web Services sin esfuerzo.

La nueva generación PHP también supone una buena noticia para desabolladores experimentados. En primer lugar, por simplificar todas aquellas funciones que ya se han indicado (al contrario de lo que normalmente se piensa, ni siquiera a los desabolladores avanzados les gusta trabajar a menos que sea imprescindible) y, en segundo lugar, por poner a su disposición funciones mejoradas. Las nuevas opciones orientadas a objetos de Zend Engine II son un regalo para el desarrollo de aplicaciones de media y gran escala; como la nueva extensión mysqli y el nuevo módulo de integración .NET, que permite la creación de aplicaciones híbridas de PHP que emplean la estructura .NET.

Este libro es una gran introducción a PHP en general y de PHP 5 en particular. Si no tiene experiencia previa con PHP, e incluso si no tiene ninguna experiencia en programación, en breve podrá trabajar con él.

Zeev Suraski, Tel Aviv

Andi Gutmans, Cupertino


# Introducción

# Historia de PHP

Rasmus Lerdorf, miembro del equipo de desarrollo de Apache, creó PHP *(Personal Home Page)* en 1994. Su única intención fue la de crear un pequeño sistema de control para verificar el número de personas que leían su curriculum vitae en la Web.

En los meses siguientes a su creación, PHP se desarrolló en torno a un grupo de programadores que comprobaban el código y sus revisiones. Para dar más potencia al sistema, Rasmus creó funciones en lenguaje C para permitir conexión a bases de datos. Este fue el comienzo de la potencia real del leguaje.

En 1995, apareció un conjunto de herramientas sobre PHP. Esta biblioteca se llamó "Herramientas para páginas personales" y contenían un analizador de código muy sencillo, un libro de visitas, un contador y algunas macros que facilitaban el trabajo de los diseñadores.

A mediados de 1995, apareció una revisión pública llamada PHP/FI 2.0. Esta nueva versión contaba con un analizador sintáctico reescrito desde 0, además de unas herramientas escritas para el tratamiento de datos desde un formulario (de ahí el nombre *áeFI, Form ínterpreter)* y conectividad con *mSQL* (Gestor de bases *de* datos).

Hacia 1997, PHP/FI se estaba usando en más de 50.000 páginas en todo el mundo. En este período de tiempo, Zeev Suraski y Andi Gutmans decidieron crear una nueva versión de PHP/FI para solventar unos problemas con una aplicación de comercio electrónico que estaban desarrollando. PHP 3.0 nació con suculentas innovaciones como la conectividad con varios gestores de bases de datos, protocolos y una API ampliada. La versión oficial de PHP 3.0 vio la luz en junio de 1998, donde se contemplaba ya la programación orientada a objetos.

En 1999 se realizó la primera revisión del motor Zend *(Zend Engine),* que aportaba modularidad, claridad y herramientas de optimización para páginas de gran escala. Zend viene de la unión de Zeev y Andi.

PHP 4.0 vio la luz en mayo de 2000, dividida en 3 partes: El motor Zend, la API de servidor y los módulos de funcionas. El motor *Zend* es el responsable de analizar el código PHP, definir la sintaxis y del lenguaje de programación. La *API* permite la comunicación con el servidor. Con esta API es posible utilizar PHP desde varios servidores. Los módulos contienen funciones para el manejo de cadenas, archivos XML o tratamiento de imágenes.

La orientación a objetos no está muy lograda en PHP 4.0. Los objetos tienen un tratamiento muy pobre e ilógico. La definición de las variables miembro (propiedades) y los métodos son siempre públicos, por lo que la encapsulación es nula. Todos los objetos se pasan por valor por defecto cuando deberían pasarse por referencia.

Todas estas propuestas realizadas por el equipo de desarrollo de PHP han desembocado en la creación del motor Zend 2.0. y su consecuencia PHP 5.

PHP 5 incorpora una verdadera orientación a objetos. Añadiendo las palabras reservadas *public, protected y prívate* a la definición de las propiedades y métodos de los objetos, se permite una verdadera encapsulación. Además del considerable avance con respecto a los objetos, PHP 5 incorpora un control de errores muy mejorado, al estilo de los lenguajes de programación más avanzados.

**Nota:**

*Después de una encuesta entre desarrolladores y usuarios de PHP se decidió cambiar el significado de PHP a* PHP Hypertext Preprocessor. *Este cambio sigue los pasos de Richard Stallman al nombrar a su sistema operativo GNU* (GNU is not Unix), *dotando a la palabra PHP de recursividad. Es un juego de palabras muy utilizado en el ambiente* hacker.

# Nuevas Características de PHP 5

Existen muchas razones para elegir PHP 5:

## Fácil de usar

PHP 5 es un lenguaje muy fácil de aprender con respecto a otros lenguajes utilizados para el mismo propósito, como JAVA o ASP. Debido a esto no es necesario hacer un estudio muy concienzudo de sus funciones para realizar programas sencillos que nos resuelvan la mayoría de los problemas diarios.

La mayoría de las funciones más usuales están disponibles por defecto, como la conexión a bases de datos o la utilización de servidores IMAP. Existe una gran cantidad de páginas con documentación y programas hechos por desarrolladores que se pueden leer y modificar libremente.

## Embebido en HTML

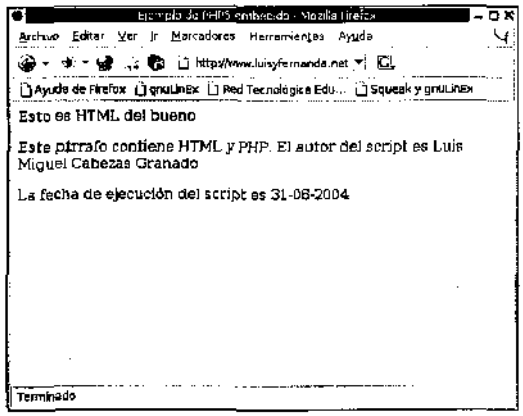
Las páginas escritas en PHP son simples páginas en HTML que contienen, además de las etiquetas normales, el programa que queremos ejecutar. Por ejemplo:

```
<HTML>
<BODY>
<HEAD>
<TITLE>Ejemplo de PHP5 embebido</TITLE>
</HEAD>
<BODY?
<P>Esto es HTML del bueno</P>
<?php
//Aquí comienza el código PHP
//Lo siguiente es una simple asignación de variables
//y su salida por pantalla
$nombre="Luis Miguel";
$apellidos="Cabezas Granado";
$fecha_hoy=date('d-ra-Y1);
?>
<P>Este párrafo contiene HTML y PHP.
El autor del script es <?php echo ("$nombre $apellidos">; ?></
P>
<P>La fecha de ejecución del script es <?php
echo("$fecha_hoy");?></P>
</BODY>
</HTML>
```

Cuando un cliente solicita esta página, el servidor preprocesa los datos y ejecuta las instrucciones de PHP. En este caso, las variables (las palabras que tienen el signo $ delante) se llenan con los valores que hay a la derecha del signo igual. Una vez resuelto todo el proceso, el servidor le envía al cliente una página sólo con etiquetas de HTML. La figura 1.1 muestra este proceso.

Si inspeccionamos ahora el código que tenemos en el navegador nos daremos cuenta de que las etiquetas de PHP han desaparecido.

```
<HEAD?
<TITLE>Ejemplo de PHP5 embebido-:/TITLE>
</HEAD>
<BODY>
<P>Esto es HTML del bueno</P>
<P>Este párrafo contiene HTML y PHP.
El autor del script es Luis Miguel Cabezas Granado</P>
<P>La fecha de ejecución del script es 05-07-2004<P>
</BODY>
</HTML>
```




**higura i.i.** Kesuitaao ae ejecutar un scnpi sencillo.

La consecuencia más inmediata es que no es necesario compilar el programa en código binario antes de poder testar si funciona o no. PHP es un lenguaje interpretado como otros muchos en el mercado (ASP, Python o JSP).

## Multiplataforma

PHP 5 se ejecuta en multitud de plataformas, Sistemas Operativos y Servidores existentes. Es compatible con los tres servidores líderes del mercado: Apache,, Microsoft Internet Information Server y Netscape Enterprise Server.

**Tabla 1.1.** Sistemas Operativos y Servidores para PHP 5.

| **"Basados en UNIX** | **Windows** |
|---|---|
| Sistema Operativo | AIX, A/UX, BSDI, Digital UNIX/Tru64, FreeBSD, HP-UX, ÍRIX, MacOS X, gnuLinux, gnuLinEx, NetBSD, OpenBSD, SCO UnixWare, Solaris, SunOS, Ultrix, Xenix y muchos más Windows 98/Me, Windows NT/ 2000/XP/2003 |
| Servidores | Apache, fhttpd, Netscape US, PWS, Netscape, Apache, Omni |

Puesto que PHP se ejecuta en todos los Sistemas Operativos indicados en la tabla 1.1 y en la mayoría de las plataformas hardware existentes (Intel,

AMD, PowerPc, SPARC, etcétera), nos será muy sencillo conseguir un laboratorio de pruebas para nuestros script.

## Licencia Open Source

La licencia de Código Abierto implica que el código fuente de PHP 5 es libre de ser descargado e inspeccionado por nosotros. La consecuencia principal es que el coste del producto en la mayoría de los casos es de 0 Euros. Tener el código fuente de PHP 5 sirve, entre otras cosas, para poder hacer nuestro servidor a medida, es decir, podemos compilar el programa con las opciones que realmente utilicemos (base de datos, LDAP).

Si acompañamos Apache, el servidor más popular, a la instalación de PHP 5 y añadimos alguna base de datos *Open Source* como PostgreSQL, tendremos un sistema completo de *script* de servidor, cuyo coste es nulo, frente a otras opciones en las que es necesario el uso de licencias.

## Multitud de Extensiones

PHP 5 se desarrolla para dar la mayor versatilidad y flexibilidad a los usuarios que lo utilizan. Es por esto por lo que existen muchas extensiones del lenguaje que permiten utilizar nuevas bases de datos, protocolos, enlaces a librerías, etcétera.

El acceso a bases de datos tiene una gran potencia, implementando soporte nativo para 15 Sistemas Gestores de Bases de Datos muy populares. En cuanto a los protocolos, podemos contar con extensiones que controlan el acceso a LDAP, IMAP o POP3. También se ha cuidado el soporte para crear imágenes en tiempo de ejecución, gracias a la librería GD.

Dada la importancia del desarrollo del lenguaje XML en los últimos años, PHP 5 incorpora tres métodos de acceso a este tipo de archivos, SAX, DOM y simpleXML. Además, se incorpora a PHP 5 la gestión de errores mediante el manejo de excepciones.

En esta versión se han añadido facilidades para utilizar los repositorios de código de PEAR.

## Velocidad e incorporación de objetos

El nuevo motor Zend 2.0 acelera los procesos de ejecución del código. Además, incorpora un nuevo modelo de objetos que permite crear cía-

ses y métodos privados, protegidos y públicos, clases abstractas e interfaces.

## Popularidad

El uso de PHP se ha disparado desde el año 1999 como puede verse en la figura 1.2.

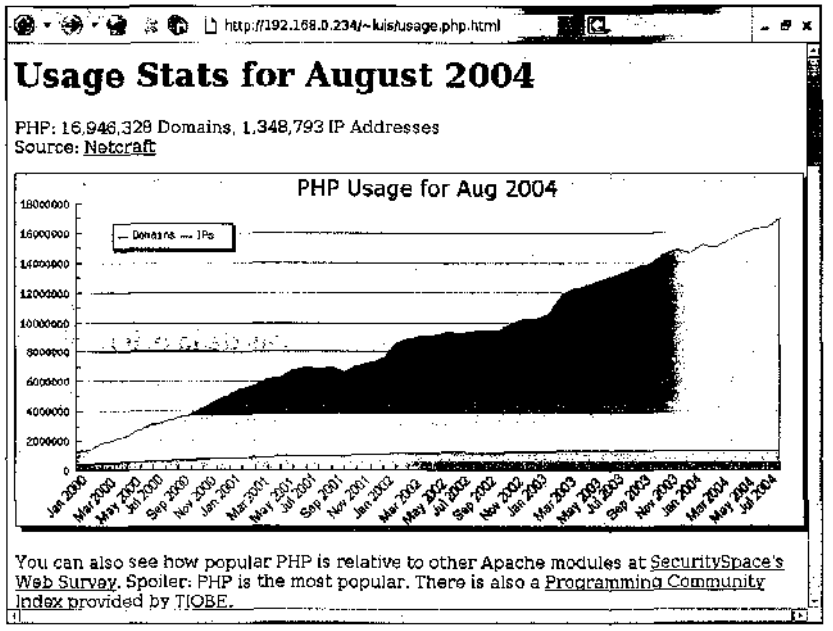



Figura 1.2. Uso de PHP desde el año 2000.

## Gran Comunidad de apoyo

PHP 5 se ha escrito bajo el auspicio del Código Abierto. Por lo tanto, existe una comunidad que apoya su desarrollo en colaboración. La ventaja principal es que existen multitud de páginas, listas de correo y foros de debate cuyo tema de conversación es el manejo de este lenguaje de programación.

Esta comunidad sirve de apoyo para todos los que necesitamos conocer desde los aspectos más básicos, hasta las implementaciones más complicadas. Tan pronto como hagamos uso de esta ayuda, nos sentiremos obligados a prestar la nuestra a usuarios principiantes y así, la Comunidad se

irá ampliando. Si nuestros conocimientos llegan a superar algún día los objetivos de este libro, podemos pensar en contribuir enviando fallos en el lenguaje, respondiendo a mensajes de las listas de correo, participando en foros de debate o escribiendo extensiones en lenguaje C.

## Objetivos del libro

A la hora de seleccionar este libro, o cualquier otro sobre PHP 5, creo que es esencial que conozca los objetivos que marcan sus capítulos. Este libro en concreto está pensado para:

- Tener una visión general de los lenguajes de *script* para desarrollo.
- Conocer las ventajas de PHP 5 frente a otros lenguajes similares. Tener una visión general de las nuevas características que ofrece PHP 5.
- Aprender los conceptos básicos para empezar a desarrollar aplicaciones sencillas que muestren contenido dinámico.
- Identificar claramente todas las características de la orientación a objetos y aprender a desarrollar aplicaciones divididas en partes con módulos reutilizables en distintos proyectos.
- Sacar partido de la API SQLite y del gestor de bases de datos MySQL para crear aplicaciones más profesionales que nutran de contenido las páginas.
- Crear formularios que envíen información y ficheros entre varias páginas de distinta forma.
- Aprender a generar archivos para Internet de distintos formatos. Los archivos XML o PDF se han vuelto indispensables en esta era y muy útiles para exportar información a los usuarios.
- Conocer las funciones básicas para la creación de páginas que controlen y manipulen imágenes. Además, conocerá herramientas para generar en el momento gráficos estadísticos de diferentes formas.
- Controlar todos los errores que puedan producir los programas, evitando mostrar en el navegador mensajes de error genéricos, para mostrar errores controlados por nosotros.
- Generar plantillas con librerías para separar el código escrito en PHP de la presentación creada con HTML o algún programa de diseño.

El libro crea una base muy fuerte para comenzar a realizar aplicaciones profesionales. Es evidente que la creación de archivos XML, PDF, genera-

ción de gráficos o plantillas Smarty necesitan un libro entero para cubrir todas las características que ofrecen. Este libro da una visión general a los lectores y permite conocer las herramientas que actualmente se utilizan para que cada uno pueda investigar por su cuenta las que más le interese.


# Capítulo 1

# Introducción a PHP5

**En este capítulo aprenderá a:**

- Diferenciar entre scripts de cliente y servidor.
- Utilizar diferentes tipos de etiquetas de inicio de PHP.
- Escribir su primer programa en PHP.
- Conocer las etiquetas HTML.

## Introducción

El mundo de los desarrolladores de páginas ha cambiado de forma vertiginosa en los últimos años. En poco tiempo han surgido tecnologías y revisiones de esas tecnologías, que hacen que los programadores tengamos que reciclarnos continuamente.

Hace unos años, el desarrollo de grandes páginas era complicado de mantener. El lenguaje existente era únicamente HTML, y solo permitía crear las páginas y subirlas a un servidor. El problema de esto es que modificar algún dato de una de las páginas implicaba un trabajo extra, que incluía descargarla, modificarla, maquetarla de nuevo y subirla al servidor.

Actualmente existe un conjunto de lenguajes que permiten desarrollar páginas Web dinámicas, es decir, que el contenido puede variar muy rápidamente sin ningún esfuerzo por parte de los desarrolladores. Éstos se denominan lenguajes de *script* de servidor, porque la ejecución del programa se realiza en el servidor Web donde se encuentra alojada la página. Existe un largo compendio de lenguajes que se pueden utilizar para la creación de páginas Web dinámicas: ASP de Microsoft, Java y JSP de Sun, Perl, Python, PHP.

## HTML estático

El tipo más básico de página Web es completamente estático, basado en texto plano y completamente escrito en HTML. La página siguiente es un ejemplo de Web estática:

```
<HTML>
<HEAD>
<TITLEb.Libros sobre PHP 5 y gnuLinux</TITLE>
</HEAD>
<BODY>
<TABLE BORDER-l>
<TR>
<TD>Título de libros americanos sobre PHP 5</TD>
</TR>
</TABLE>
<ul>
<lixb:> Advanced PHP Programming</bx/brx/li>
<lixb>PHP5 and MySQL bible</bx/brx/li>
<lixb>Learning PHP 5</b></brx/li>
<lixb>Upgrading to PHP 5</bx/brx/li>
<lixb>Beginning PHP 5 and MySQL</bx/brx/li>
```

```
<lixb>PHP 5 Power Programming</fc>></br></li>
</ul>
<TABLE border=l>
<TR>
<TD>Título de libros americanos sobre gnuLinux</TD>
</TR>
</TABLE>
<ul>
<lixb>Running Linux</bx/brx/li>
<lixb>Linux in a Nutshell</bx/brx/li>
<lixb>How Linux Works</bx/brx/li>
<lixb>Linux for Non-Geeks</bx/brx/li>
</ul>
</BODY>
</HTHL>
```

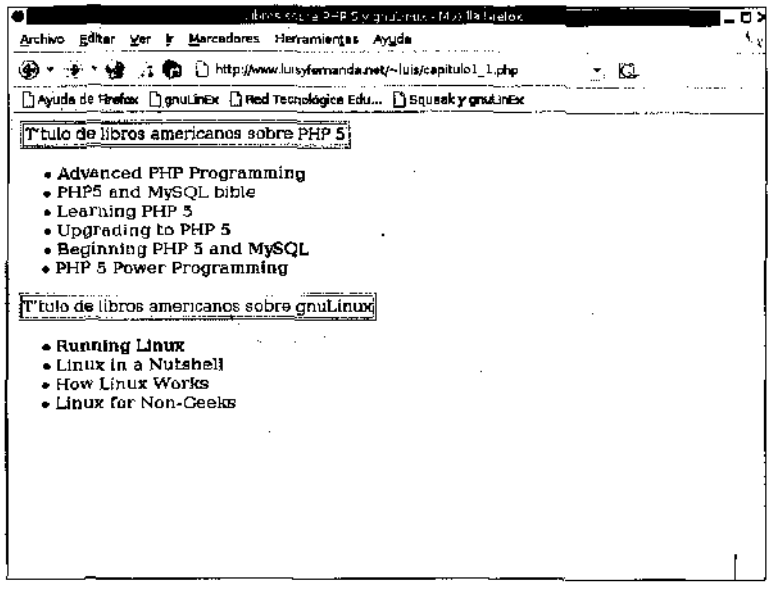



**Figura 1.1.** Ejemplo de página Web estática.

La figura 1.1 muestra el ejemplo de una Web estática. El funcionamiento de Internet es algo complejo para explicarlo en este capítulo, pero vamos a dar vinas nociones básicas sobre lo que pasa tras la petición de una página Web por un cliente. Los pasos que sigue nuestro navegador de Internet son:

- Nuestro ordenador solicita al servidor una página Web a través de Internet.
- El servidor envía los datos solicitados en formato de texto.

- El navegador recibe estos datos, interpreta la página Web enviada y la muestra en la pantalla de acuerdo con la resolución del monitor, las preferencias del usuario y algún otro factor.

En la figura 1.2 podemos ver todo el proceso.

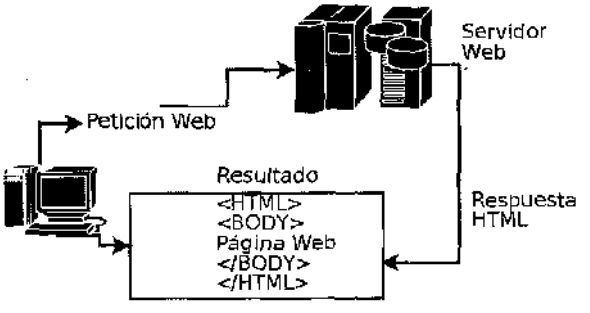



**Figura 1.2.** Esquema de petición de una Web estática.

Las páginas Web estáticas no permiten apenas una interactividad con el usuario final. Por eso, han aparecido numerosas tecnologías que hacen más usable las páginas y eliminan algunas restricciones o limitaciones. Los lenguajes Javascript, Vbscript, CSS o los *ap-plet* de Java ofrecen una mayor interactividad, pero son dependientes del navegador que utilicemos o de programas externos.

## Tecnologías del lado del cliente

Muchas de las mejoras añadidas al lenguaje HTML pertenecen a tecnologías del lado del cliente, es decir, que se descargan junto con la página Web solicitada al servidor y se ejecutan en nuestro ordenador local.
La tabla 1.1 todas las tecnologías aparecidas hasta ahora del lado del cliente.
Estas tecnologías, aunque son más llamativas y permiten desarrollar la imaginación en pro del diseño, dependen totalmente de las características del navegador. Por ejemplo, si tiene un ordenador ibook de Apple con un Sistema Operativo gnuLinux Debían (como el autor), le será complicado obtener una máquina virtual de Java y un intérprete de Flash. Las tecnologías de cliente no pueden rescatar datos de servidores, porque su ejecución es únicamente en local. Ésto los imposibilita para recuperar información de bases de datos o servicios.

**Tabla 1.1.** Tecnologías del lado del cliente.

| [Tecnología | Descripción | Efecto de ejemplo |
|---|---|---|
| CSS, HTML Dinámico | Sirve para dar formato a las páginas: color, tamaño, capas, efectos, | Enlaces con movimiento. |
| Javascript, Vbscript | Manejo de eventos del navegador. | Imágenes que cambian cuando se aproxima el ratón. |
| Applets de Java | Aplicaciones pequeñas | Puzzles, conectividad con bases de datos. |
| Animaciones Flash | Animaciones gráficas | Películas *¡interactivas* |

# Tecnologías del lado del servidor

Los lenguajes del lado del servidor son invisibles para los clientes. Las páginas que utilicen *scripts* de este tipo contienen el código entre etiquefas *parecidas a las de* HTML, pero éstas desaparecen *cuando el cliente recibe* la página.

Los pasos que debe seguir nuestro navegador de Internet son:

- Nuestro ordenador solicita al servidor una página Web a través de Internet.
- El servidor comprueba si la página solicitada contiene *scrvpt* del lado del servidor (PHP, ASP, JSP, etcétera).
- Ejecuta los posibles *scripís y* añade el resultado final a la página Web resultante.
- El navegador recibe estos datos, interpreta la página Web enviada y la muestra en la pantalla de acuerdo con la resolución del monitor, las preferencias del usuario y algún otro factor.

La figura 1.3 muestra el proceso.

Los lenguajes del lado del servidor necesitan un motor (un programa) que interprete el código.

Este programa puede formar parte o no del servidor Web. En nuestro caso utilizaremos el motor Zend Engine 2.0 para parsear (interpretar) los programas escritos en PHP 5.

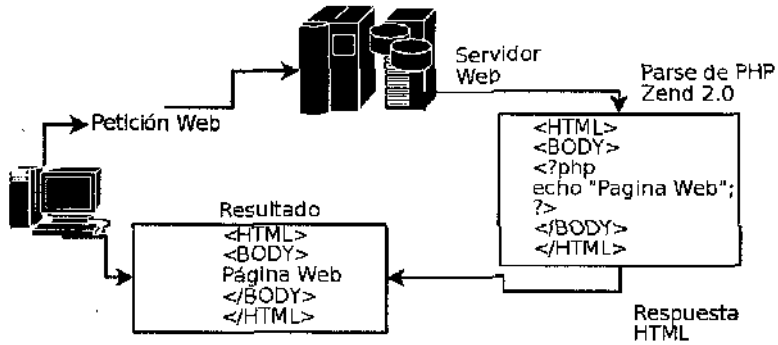



**Figura 1.3.** Esquema de petición de una Web dinámica.

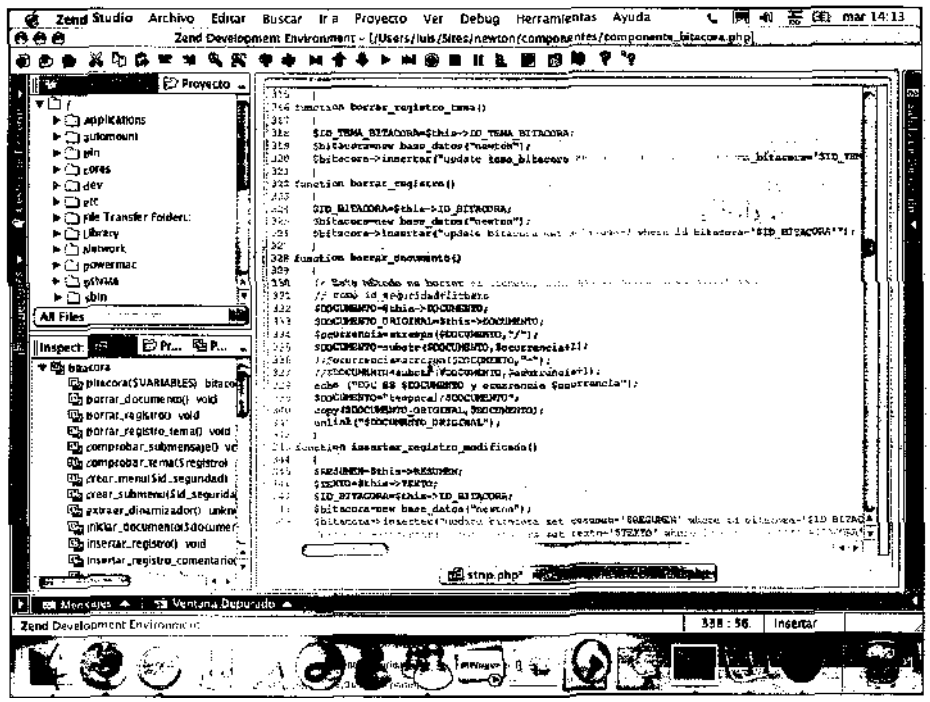



**Figura 1.4.** Zend Studio 3.5.1 para MacOSX.

Suponiendo que ya tiene un servidor para probar los programas o ha leído el Apéndice A para instalar su propio servidor Web con PHP 5, vamos a crear nuestro primer programa del lado del servidor.

Para escribir los programas puede utilizar cualquier editor de texto plano.

En gnuLinux tiene algunos editores libres que funcionan muy bien. Sus nombres son Bluefish o Anjuta.

En entornos propietarios como Windows o MacO^X existen editores profesionales que, además de diseñar páginas Web, permiten añadir todo tipo de código PHP, eso sí, los programas no son gratuitos y hay que pagar una elevada licencia por su uso.

Para los tres sistemas operativos tenemos una impresionante herramienta de edición y depuración creada por los programa^ores de PHP 5.
La herramienta Zend Studio tiene dos versiones, una de pago bajo licencia con múltiples opciones de depuración y otra gratuita, con opciones de desarrollo tan interesantes como el auto completado de funciones y variables.

## Etiquetas de PHP

Lo primero que debe saber es que todo prograrr[1]^ escrito en PHP debe empezar por unas etiquetas determinadas.

```
<?php     ?>
```

Además áe esta íoxma ó.e ímc'iax *xm* programa, pu^dt ^*urf*^*cam* -¿'ï -hiti ¿ivo php . i n i para que sus programas puedan empezar entre:

```
<?        ?>
```

> Nota:
>
> *El archivo* php.ini *contiene la configuración de PHP en ese momento. Cambiando algunos parámetros, podrá cambiar la forma de funcionar del parser.*

El parámetro s h o r t - o p e n - t a g puede igualarse a *off o* a *on.* Si contiene el valor *on,* PHP permitirá el uso de la etiqueta cofta.
Otra forma posible de empezar un programa escrito en PHP es utilizando las etiquetas de ASP.

Mucha gente que desarrolla con programas de Microsoft, como FrontPage, para generar páginas Web utiliza esta forma de empezar el código.

```
<%      %>
```

## Nuestro primer programa en PHP 5

Después de estas pequeñas nociones, ya está preparado para crear su primer programa. Éste le va a permitir tener una visión de la configuración global de PHP 5.

Puede escribir el programa siguiente en cualquier editor de texto como vimos anteriormente.

```
<HTML>
<HEAD>
<TITLE>Mi primer programa en PHP 5</TITLE>
</HEAD>
<BODY>
<?php
echo "Este es el típico Hola Mundo!!!<brxbr>";
phpinfo();
?>
</BODY>
</HTML>
```

Como puede ver, el programa contiene la estructura normal de una página Web. La etiqueta <HTML> define el comienzo y el final de la página. Esta puede dividirse en dos partes. La primera está separada por la etiqueta <HEAD> y contiene información sobre el autor, título y meta datos que pueden servir para añadir código Javascript o CSS. La segunda parte es el cuerpo de la página Web, es decir, contiene todos los datos que verá el usuario en el navegador. Está separada entre etiquetas <BODY>.

Dentro del BODY puede ver que empleamos las etiquetas de inicio y fin de programa PHP. Dentro de estas etiquetas hay dos funciones que hacen cosas distintas. Las funciones son programas ya creados y funcionales que pueden utilizarse en el transcurso de nuestro programa. Existen en PHP 5 una infinidad de funciones útiles para el manejo de cadenas de caracteres, funciones aritméticas o funciones para mostrar texto en pantalla.

La función `echo {)` se encarga de mostrar los datos en pantalla; en este caso, se encargará de mostrar la frase "Este es el típico Hola Mundo!!!". Además, imprime dos etiquetas <br >, que no se muestran, porque tienen sentido dentro de HTML. La etiqueta <br> añade un salto de línea a la página Web.

La función `phpinfo ()` envía al navegador la configuración de PHP que estamos utilizando en el servidor. Aquí aparece la configuración que se ha utilizado para compilar el parser, las librerías auxiliares que contiene, el sistema que se está utilizando, etcétera.

**Nota:**

*Durante el proceso de aprendizaje del libro, conocerá muchas funciones y sus aplicaciones inmediatas. En el capítulo 5 del libro aprenderá a crear sus propias funciones.*

El resultado de la ejecución del programa lo muestra la figura 1.5.

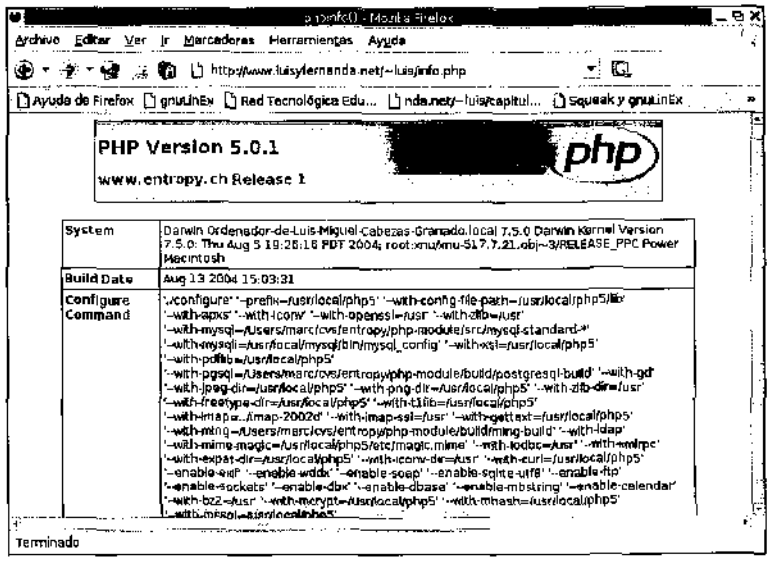



**Figura 1.5.** Resultado de nuestro primer script.

## Repaso de HTML

El lenguaje de marcas de hipertexto se ha convertido en el principal referente de la Web. Para seguir adecuadamente los contenidos del libro, necesitará conocer el funcionamiento de HTML para insertar sus programas escritos en PHP 5.

## Cabecera y cuerpo de una página Web

Todo documento escrito en HTML está contenido entre las etiquetas <HTML> y </HTML>. Se pueden estructurar en:

- **Cabecera:** Entre las etiquetas <HEAD> y </HEAD>: Contiene información relativa al documento.

- **Cuerpo:** Entre las etiquetas <BODY> y </BODY>: Debe contener la información que el usuario verá en el navegador.

## Cabecera

El título del documento debe ir entre las etiquetas <TITLE> y </TITLE>. Este título se mostrará en la parte superior del navegador. Además, se utiliza como descripción si guarda la página en el apartado de Favoritos.

Para añadir información sobre el autor, fecha de expiración o el programa con el que hemos diseñado la Web, puede utilizar la etiqueta <META>.

```
<HTML>
<HEAD>
<TITLE>Página en HTML</TITLE>
<META NAME="Author" CONTENT="Luis Miguel Cabezas Granado">
</HEAD>
<BODY>
</BODY>
</HTML>
```

Algunas etiquetas permiten la inclusión de atributos. La etiqueta <META> contiene el atributo ÑAME y CONTENT para señalar en el documento el autor de la página. Los buscadores harán buen uso del contenido de estas etiquetas para enlazarlas en sus motores de búsqueda.

## Cuerpo del documento

La etiqueta <BODY> también puede contener atributos que modifiquen el aspecto exterior de la página.

Los principales atributos son:

- **BGCOLOR:** Indica el color de fondo de la Web.
- **TEXT:** Color general del texto.
- **LINK:** Color del texto de los enlaces.
- **VLINK:** Color de los enlaces que ya han sido utilizados.
- **ALINK:** Color de activación del texto.
- **BACKGROUND:** Imagen de fondo de un documento.

En el siguiente ejemplo puede ver la utilización de los atributos de la etiqueta del cuerpo:

```
<HTML>
<HEAD>
```

```
<TITLE>Página en HTML</TITLE>
<META NAME="Author" CONTENT="Luis Miguel Cabezas Granado">
</HEAD>
<BODY BGCOLOR="#FF0 0 00" LINK="#000000" VLINK="#FF00FF">
</BODY>
</HTML>
```

Nota:

*El color se escribe en formato RGB. Esto quiere decir que un color puede dividirse en tres componentes (rojo, verde y azul) y, eligiendo la intensidad de cada componente, se forma el color deseado. Cada componente tiene 2 cifras en hexadecimal. Así, el color* #FF0000 *tiene al número* FF *hexadecimal (255 en decimal) en su componente rojo y los demás a 0; por lo tanto, el color resultante será Rojo intenso.*

## Párrafos y saltos de líneas

Los textos pueden escribirse dentro de la etiqueta <BODY> en el orden que quiera; los navegadores no interpretan los saltos de líneas o líneas en blanco, eliminando todos los espacios vacíos.

Para crear un salto de línea tiene que utilizar <BR> escrito después del texto. Para crear párrafos tendrá que escribir el texto entre las etiquetas **<P>** **</p>**.

```
<HTML>
<HEAD>
<TITLE>Página en HTML</TITLE>
</HEAD>
<B0DY>
<P>
Esto es párrafo
y también un salto de líneasbr>
</P>
</BODY>
</HTML>
```

## Estilo de texto

Podemos resaltar partes del texto con algunas etiquetas:

- <B>: Bloque de texto en negrita.
- <I>: Texto en cursiva.

- <U>: Bloque de texto subrayado.
- **<BIG>:** Texto con su mayor tamaño.
- <SMALL>". Texto con \m tamaño pequeño.
- **<CENTER>:** Texto centrado.
- <SUP>:.Superíndice.
- **<SUB>:** Subíndice.

El ejemplo siguiente, que también se muestra en la figura 1.6, muestra la utilización de alguna de las etiquetas anteriores:

```
<HTML>
<HEAD>
<TITLE>Página en HTML</TITLE>
<META NAME="Author" CONTENT="Luis Miguel Cabezas Granado">
</HEAD>
<BODY>
<CENTER>Texto centrado-:/CENTERxBR>
<UxIxBIG>Texto subrayado y cursiva</BIGx/I></UxBR>
<B>Texto en negrita</BxBR>
<SUP>Texto superíndice</SUPxBR>
Este contiene un <SUB>subíndice</SUB><BR>
</BODY>
</HTML>
```

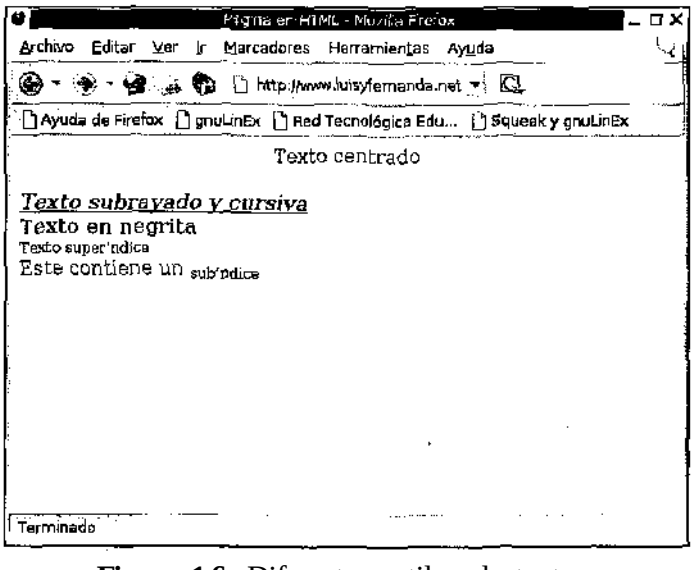



**Figura 1.6.** Diferentes estilos de textos.

Además, puede elegir el formato de las fuentes de cada bloque de texto.

La etiqueta <FONT> permite seleccionar entre varios atributos para utilizar distintas fuentes, colores o tamaños. Entre los atributos que podemos utilizar:

- **COLOR:** Color de la fuente.
- **SIZE:** Tamaño de la fuente.
- **FACE:** Tipo de fuente.

```
<FONT SIZE=1 FACE="Comic">Texto con tamaño y fuente
determinados</FONT>
```

## Enlaces de texto

Un enlace es un mecanismo que permite dirigirse a una página determinada después de hacer clic en el texto. Si entre las etiquetas <A> y </A> aparece algún contenido (texto o imagen), al hacer clic en la situación dentro de navegador, la página será dirigida a la nueva dirección.

El atributo HREF contiene la dirección del enlace.

```
<A HREF="pagina2.htm">enlace</A>
```

## Listas

Las listas son capaces de presentar ordenadamente una serie de conceptos. Los tipos de lista son:

- <UL>: Lista desordenada.
- <OL>: Lista ordenada.
- <DL>: Lista de definición.

La diferencia entre los tipos de lista radica en el carácter que aparece en la zona izquierda de cada línea de la lista. Las listas desordenadas tendrán un gráfico formal (una forma geométrica), las ordenadas tendrán un número de orden y las listas de definición una letra.

```
<HTML>
<HEAD>
<TITLE>Página en HTML</TITLE>
</HEAD>
<BODY>
<B>Lísta desordenada-:/BxBR>
<UL>
   <LI>Línea 1.
   «Lis-Línea 2.
   <LI>Línea 3.
```

```
<UI-i>
   <LI>Línea 3.1.
   <LI>Línea 3.2.
  </UL>
  <LÍ>Línea 4.
</UL>
<B>Lísta ordenada</B><BR>
<OL>
  <LÍ>Línea 1.
  <LJ>Línea 2.
  <Ll>Línea 3.
  <ot>
     <LI>Línea 3.1.
     <LI>Línea 3.2.
  </OL>
  <LÍ>Línea 4.
</OL>
</BODY>
</HTML>
```

Cada líitea de una lista debe estar precedida de la etiqueta <LI>. El código anterior genera la Web de la figura 1.7.

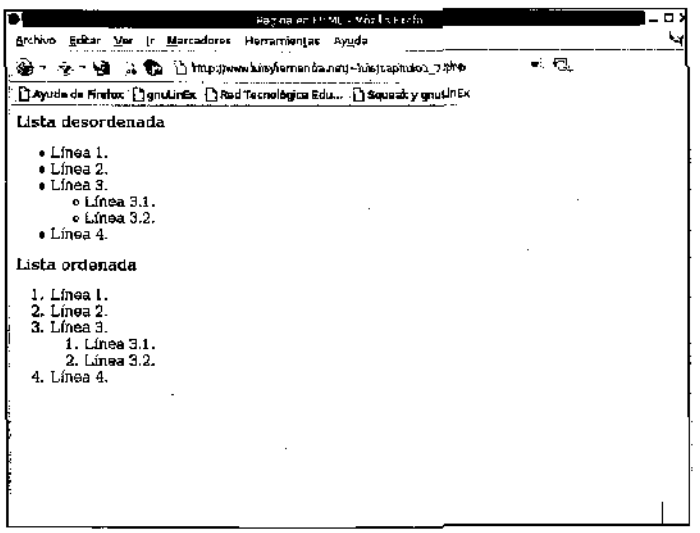



Figura 1.7. Listas ordenadas y desordenadas.

## Imágenes

La etiqueta < IMG> utiliza el atributo SRC para insertar la imagen seleccionada en el documento. .

Además, puede utilizar ALT para añadir una descripción a la imagen. Entre las imágenes que se pueden incluir en la Web están los tipos de archivos GIF, PNG y JPG.

```
<IMG SRC="fotol.jpg" ALIGN="center">
```

La alineación de la imagen con respecto a la pantalla se puede controlar con el atributo ALIGN.

## Tablas

El elemento estrella en una página Web es la tabla. Actualmente, las tablas se utilizan para colocar en lugares determinados de la pantalla textos, imágenes o el menú de selección. Para definir una tabla, tiene que insertar todos los datos entre las etiquetas <TABLEi, y </TABLE>.

A continuación, añada las filas y las columnas de nuestras tablas. Para añadir una fila, tendrá que utilizar la etiqueta <T^> y para incluir una nueva columna la etiqueta <TD>.

Para diseñar una tabla de 3 filas y 4 columnas siga los pasos que muestra el ejemplo:

```
<HTML>
<HEAD>
<TITLE>Página en HTML</TITLE>
<META NAME="Author" COMTENT="Luis Miguel Cab<bzas Granado">
</HEAD>
<BODY>
<TABLE B0RDER=1>
<TR>
<TD>Columna 1.1</TD>
<TD>Col"umna 1.2</TD>
<TD>Columna 1.3</TD>
<TD>Columna 1.4</TD>
</TR>
<TR>
<TD>Columna 2.1</TD>
<TD>Columna 2.2</TD>
<TD>Columna 2.3</TD>
<TD>Columna 2.4</TD>
</TR>
<TR>
<TD>Columna 3.1</TD>
<TD>Columna 3.2</TD>
<TD>Columna 3.3</TD>
<TD>Columna 3.4</TD>
</TR>
```

```
</TABLE>
</BODY>
</HTML>
```

Como muestra el código, la etiqueta <TABLE> puede contener atributos como BORDER, que añade un borde a todas las celdas de la tabla. Esta forma de actuar la vemos en la figura 1.8.

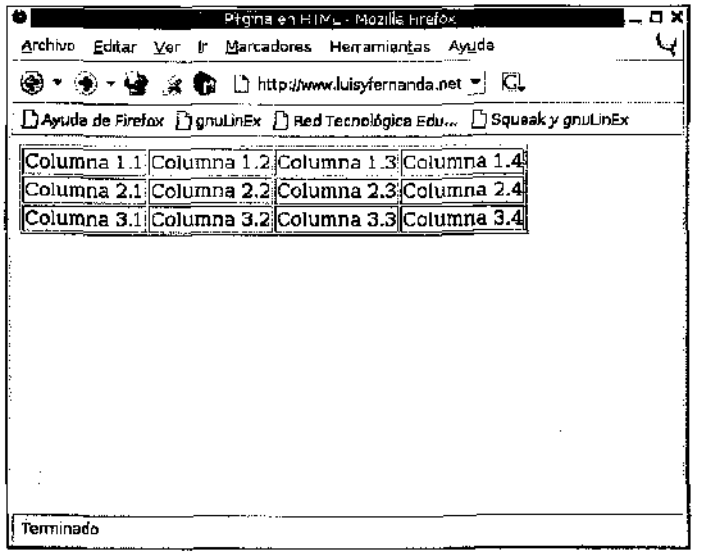



**Figura 1.8.** Tabla de 3 filas y 4 columnas.

# Resumen

Es muy importante para la lectura de los capítulos del libro que conozca el lenguaje HTML en profundidad. PHP 5 se encarga de generar datos dinámicamente desde una base de datos, archivos o haciendo cálculos complejos, pero al final será una etiqueta HTML la que muestre esos datos en pantalla. Puesto que el objetivo de este libro no es conocer los aspectos de HTML, invito al lector a indagar en otras publicaciones, o en Internet, para suplir las posibles carencias que pueda tener.

Los ejemplos del libro se han escrito directamente en un editor de textos, pero esta no es la única posibilidad. También puede utilizar para realizarlos algún programa de diseño Web, con la ventaja de que las etiquetas HTML serán añadidas a golpe de clic.


# Capítulo 2

# Variables, constantes y tipos de datos

**En este capítulo aprenderá a:**

- Almacenar información en variables.
- Manejar variables, constantes y constantes predefinidas por PHP 5.
- Utilizar funciones de manejo de datos.

## Variables en PHP 5

Como en todos los lenguajes de programación, PHP 5 permite almacenar datos de distintos tipos en memoria. Estas zonas de memoria se llaman variables. Las variables comienzan por el símbolo de dólar ($) y no necesitan ser declaradas antes de comenzar el programa, como en otros lenguajes.

## Tipos de Variables

Existen varios tipos de variables:

- **Entero *(integer):*** Almacena números sin decimales. Se puede utilizar la notación decimal (83), octal (0123) o hexadecimal (0x12).
- **Coma Flotante *(double):*** Números con decimales.
- **Carácter *(string):*** Texto o información numérica escrita entre comillas dobles ("") o simples ('').
- ***Bootean:*** Sólo tiene dos posibles valores: Verdadero o Falso.
- **Nulo *{NULL):*** Es un tipo especial que solo contiene un valor: NULL.
- **Vectores *(array):*** Colecciones de datos.
- **Objetos** *(object):* Conjunto de datos y funciones independientes.

Los 5 primeros tipos son simples y los 2 siguientes *(array y object)* son compuestos. Trataremos los tipos compuestos en capítulos sucesivos.

```
<?php
//Asignación de números enteros, de coma flotante y cadenas de
caracteres
$numero_entero = 12343;
$numero_flotante = 12343.123;
$cadena_caracter = "12 34 3";
//Asignación de los tipos especiales boolean y NULL
$verdadero = TRUE;
$vacio = NULL;
?>
```

Hay que tener en cuenta que el nombre de las variables no puede empezar por un número, pero sí puede contenerlos. También pueden empezar por un guión bajo (_). Puesto que PHP 5 es sensible a las mayúsculas y minúsculas las variables $numero y $NUMERO serán distintas.

```
<?php
$4numero = 23;    //Esta línea da error
```

```
$_numero = 45;
//Las líneas siguientes muestran que PHP 5 es sensible a
mayúsculas
$numero = 23;
$NUMERO = 24;
$Numero = 25;
echo ("numero es: $numero<br>") ;
echo ("NUMERO es: $NUMERO<br>" ) ;
echo ("Numero es: $Numero<br>") ;
?>
```

**Nota:**

Es *una buena práctica de programación llamar a las variables de una forma descriptiva. Si tenemos un formulario que intenta pasar datos sobre el nombre y apellido de una persona, lo más correcto es llamar a las variables* $nombre *y* $apellido, *en vez de* $n *y* $a. *Hay que buscar siempre la similitud entre el dato que pasamos y el nombre de la variable para que el programa resulte legible.*

## Asignación de variables

La asignación es simplemente dar un valor a la variable. Esto se hace poniendo un símbolo = entre la variable y el dato que queremos asignar.

```
<?php
$numero_pi = 3.14159 // Aproximadamente
?>
```

Si se fija en el ejemplo, acabamos de asignar un número de coma flotante a la variable $numero_pi. En este momento podemos cambiar el valor de la variable asignando un nuevo valor, que puede ser de un tipo diferente.

```
<?php
$numerojí = "3.14159"; //El tipo se convierte a cadena de
caracteres
?>
```

# Tipos simples

Los tipos de variables simples (enteros, coma flotante, cadenas de caracteres, *boolean* y *NULL)* deberían ser familiares si conocemos algún lenguaje

de programación como C. Puesto que la base de la programación consiste en almacenar datos correctamente, vamos a ver las peculiaridades de los diferentes tipos.

## Enteros (integer)

Corresponde a un número entero sin decimales, pudiendo ser negativo o el número cero. Pueden ser almacenados en diferentes formatos correspondiendo con las bases decimal (base 10), octal (base 8) y hexadecimal (base 16). Por defecto se utiliza la numeración decimal. Para utilizar base 8 debe ponerse un 0 delante del número asignado y un Ox si queremos *utilizar base 16. Para añadir un número negativo basta con poner el signo* menos (-) delante del número.

```
<?php
$entero_base10 = 1234;
$entero_base8 = 01234;
$entero_base16 = 0x1234;
$entero_negativo = -1234;
echo ("Decimal: $entero_base10<br>") ;
echo ("Octal: $entero_base8<br>") ;
echo ("Hexadecimal: $entero_base16<br>") ;
echo ("Negativo: $entero_negativo<br>") ;
?>
```

La salida del navegador es:

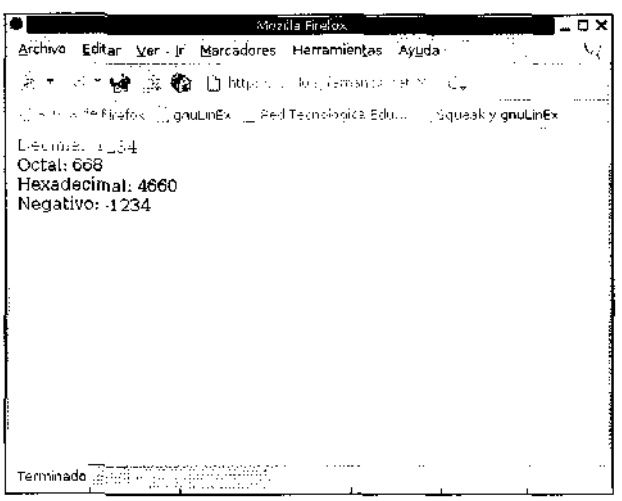



**Figura 2.1.** Asignación de enteros en decimal, octal y hexadecimal.

La salida de los números se hace en decimal, aunque la asignación se haya hecho en hexadecimal o en octal. Esto es así porque PHP trabaja internamente en binario y hace la conversión a decimal de todos los números enteros y de coma flotante.

## Números de coma flotante (double)

Este tipo de dato contiene números enteros con decimales. Se almacenan de forma diferente a los números enteros, por lo que las sumas de cantidades de coma flotante con cantidades enteras darán como resultado un número de coma flotante.

```
<?php
//Números de coma flotante
$numero__double = 1234.123;
$numero_double2 = 0.1213;
$numero_double3 = -12 3 4.0;
echo ("Salida de la función echo(): $numero_double3<br>") ;
?>
```

La salida en el navegador sería:

```
Salida de echo: -1234
```

Como puede comprobar, parece que el número que se muestra en pantalla es del tipo entero. Esto es porque la función `echo` 0 no muestra el formato del dato. Para que esto ocurra debemos utilizar otro tipo de función.

```
<?php
$numero_double3 = -1234.0;
echo ("Salida de printft): " ) ;
printf("%f","$numero_double3");
?>
```

El código anterior muestra ahora una salida por pantalla más coherente:

```
Salida de printf() es: -1234.000000
```

## Cadena de caracteres (string)

Es un conjunto de caracteres encerrados entre comillas dobles ("") o simpies (' ')•

```
<?php
//Definición de variables de tipo string
$cadenal = "Esto es una cadena de caracteres";
$cadena2 = 'Esto es una cadena entre comillas simples';
$cadena3 = "" ; //Cadena con 0 caracteres.
?>
```

La diferencia entre las comillas simples y las comillas dobles radica en:

- **Comillas simples:** Permiten imprimir el contenido íntegro de caracteres que esté entre las dos comillas.
- **Comillas dobles:** Permiten incluir variables para imprimirlas junto al texto.

```
<?php
$dato = "IMPRIMIBLE";
$texto_simple = 'No puede imprimir la variable $dato';
$texto_doble = "Imprime la variable dato como $dato";
echo  ("$texto_simple<br>");
echo  ("$texto_doble");
?>
```

Existen algunos caracteres especiales que no pueden ser añadidos a una variable de tipo *string*, a menos que avisemos antes a PHP para que tenga cuidado a la hora de manejar el dato. Este aviso se conoce como "secuencia de escape" y no es más que la inclusión delante del carácter especial de un símbolo \. Así, si quiere insertar unas comillas dobles entre un texto, tendrá que escribir \" para que no se produzca un error.

```
<?php
//Secuencias de escape
$cadenal = 'Aquí se pueden añadir comillas "dobles"<br>';
$cadena2 = "Aquí se pueden añadir comillas 'simples'<br>" ;
$cadena3 = "Esta cadena da errores por las comillas
"dobles"<br>";
$cadena4 = "Esta cadena no da errores por las comillas
\"dobles\"<br>";
echo $cadenal;
echo $cadena2;
echo $cadena3;
echo $cadena4;
?>
```

El resultado muestra la impresión de las cadenas, excepto la cadena3 que da un error por no incluir la secuencia de escape para las comillas.

**Tabla 2.1.** Secuencias de escape comunes.

| Secuencia | Valor | % |
|---|---|---|
| \$ | Signo de dólar $ | |
| \" | Comillas dobles | |
| V | Comillas simples | |
| \\ | Barra invertida | |

| Secuencia | Valor |
|---|---|
| \n | Nueva línea |
| \r | Retorno de carro |
| \t | Tabulador |

## Boolean

Una variable de tipo *boolean* sólo puede contener dos valores: Verdadero o Falso. Se suelen utilizar para comprobar si se cumple una condición en el programa.

```
<?php
$variable_booleana = TRUE;
if ($variable_booleana) {
echo ("El resultado es VERDADERO");
}
?>
```

El resultado se muestra en la figura 2.2.


**Figura 2.2.** Comprobación del valor de la variable boolean.

Aunque todavía no hemos visto las estructuras de control, podemos advertir que la salida del programa anterior da como resultado la impresión en pantalla del texto que contiene la función `echo ()` . Además de las variables *boolean*, podemos tomar otros tipos de datos como Verdaderos o Falsos para hacer nuestras comprobaciones.

- Los números enteros o decimales, positivos o negativos son todos Verdaderos. El 0 se considera Falso. En cualquier caso se recomienda no utilizar variables de coma flotante para comprobar si son Verdaderos o Falsos.
- Las cadenas de caracteres son Verdaderas, excepto cuando no contienen ningún dato o contienen únicamente el carácter "0".
- Las variables del tipo *NULL* son siempre Falsas.

### NULL

Es un tipo especial de datos que sólo contiene el valor *NULL.* Aunque se suele utilizar por norma en mayúscula, PHP 5 lo acepta escrito de diversas formas: *NULL, nuil, NuLl.*

```
<?php
$variable_nula = NuLl;
if ($variable_nula) {
echo ("La comprobación es VERDADERA");
}
else {
echo ("La comprobación es FALSA");
}
?>
```

## Variables de variables

Las variables son, en definitiva, nombres descriptivos que asocian un espacio en memoria del servidor con una palabra entendible por nosotros. En ese espacio de memoria podemos almacenar diferentes tipos de datos. Podemos definir una variable de la siguiente forma:

```
<?php
$variablel = "hola";
?>                    '
```

En este momento tenemos un espacio de memoria llamado $variablel, donde se aloja el dato "hola". ¿Qué pasa si queremos tratar la palabra "hola" como una variable?. Podemos hacerlo añadiendo un signo de $ delante de $variablel, por lo que se reservará otro espacio en memoria para alojar el nuevo contenido esta vez con el nombre $hola:

```
<?php
$variablel = "hola";
$$variablel = "mundo";
```

```
//Las dos líneas siguientes producen la misma salida
echo ("$variable1 $$variable1<br>") ;
echo ("$variable1 $hola<br>");
?>
```

La salida por pantalla nos da el siguiente resultado:

```
hola mundo
hola mundo
```

# Constantes

Las constantes son tipos de datos que no varían en el desarrollo de un programa. En la vida real existen muchos tipos de constantes, el número pi, la temperatura de congelación del agua, el nombre de la Empresa, etcétera. Para crear una constante tiene que usar la función `define ()` de la siguiente forma:

```
<?php
define("EMPRESA","Zend.S.A.");
define("Autor","Luis Miguel Cabezas Granado");
?>
```

Para mostrar el valor de las constantes únicamente hay que invocar su nombre, esta vez sin utilizar el símbolo de $.

```
<?php
//Definimos primero la constante
define ("COLOR_ROJO","#FF0000");
//Y ahora escribimos en pantalla su resultado
echo COLOR_ROJO;
//Hay que destacar que el signo de $ no hace falta ponerlo
//Existe otra función que nos permite averiguar el valor de la
constante
echo constant ("C0L0R_R0JO") ;
?>
```

## defined()

Puede utilizar `defined ()` para averiguar si una constante ya se ha creado.

```
<?php
define("Editorial","Anaya");
if (defined("Editorial") ) {
echo "La Editorial es: ".Editorial;
}
?>
```

## Constantes predefinidas

PHP 5 define varias constantes cada vez que se ejecuta un script y que son accesibles por nosotros.

**Tabla 2.2.** Algunas constantes definidas por PHP 5.

| Nombre | Descripción |
|---|---|
| PHP_VERSION | Versión del parse de PHP que estamos utilizando. |
| PHP_OS | Sistema operativo del servidor de PHP. |
| PEAR_EXTENSION_DIR | Ruta donde está instalada la extensión PEAR. |
| PHP_LIBDIR | Ruta donde están almacenadas las librerías de PHP 5. |

Finalmente tenemos 5 constantes mágicas, tal y como las definen en el sitio Web www.php.net, que cambian de valor dependiendo del lugar donde se usen. Por ejemplo, el valor de la constante __LINE__ depende de la línea donde la estemos usando en nuestro programa.

**Tabla 2.3.** Constantes mágicas.

| Nombre | Descripción |
|---|---|
| __LINE__ | Indica el número de línea desde la que imprimimos el valor. |
| __FILE__ | Ruta completa al fichero. |
| __FUNCTION__ | Nombre de la función que la contiene. |
| __CLASS__ | Nombre de la clase. |
| _METHOD_ | Nombre del método. |

**Advertencia:**

*Hay que tener cuidado con el nombre de las constantes mágicas. Los símbolos que hay a la izquierda y ala derecha del nombre son dos guiones bajos seguidos (_).*

```
<?php
echo __LINE__.",- VERSIÓN de PHP 5:".PHP_VERSION;
echo "<br>";
```

```
echo __LINE__.".- SISTEMA Operativo Servidor: ".PHP_OS;
echo "<br>";
echo __LINE__.".- Ruta de las extensiones: ".PHP_EXTENSION_DIR;
echo "<br>";
echo __LINE__.".- API de Servidor: ".PHPJ3API
?>
```

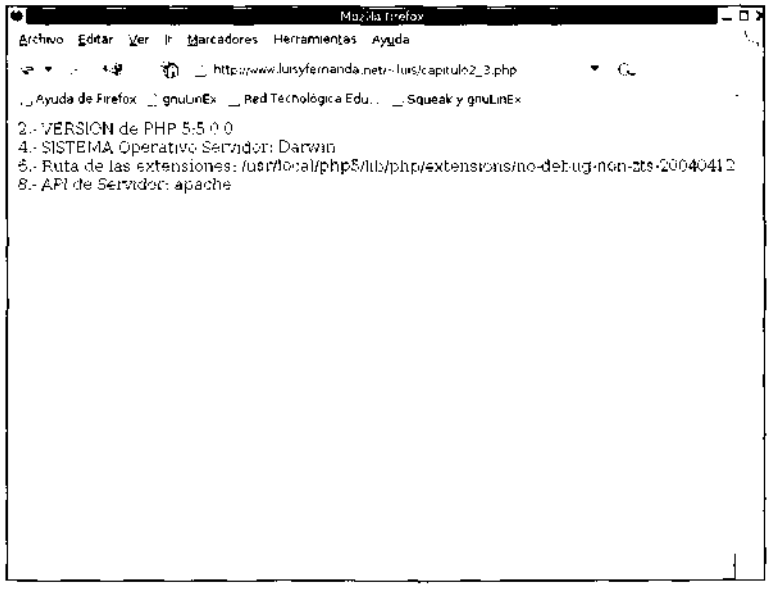



**Figura 2.3.** Las constantes almacenan información muy útil.

## Funciones relacionadas con variables

PHP 5 brinda al programador una serie de funciones para el manejo de variables.

### isset()

Con esta función podemos averiguar si una función existe dentro de nuestro programa.

Si existe devuelve *true* y si no existe *false.*

```
<?php
$DNI = "8868543-Z";
if (isset($DNI)) {
echo ("La variable DNI existe!!!");
```

```
}
?>
```

## unset()

Libera la memoria ocupada por una variable, destruyendo su nombre y su contenido. Después de usar unset (), la variable destruida aparecerá como/a/se al utilizar la función isset () .

```
<?php
$Nombre = "María Fernanda";
if (isset($Nombre)) {
echo ("El nombre existe!!!");
}
//Podemos comprobar qué pasa si liberamos la variable $Nombre
unset($Nombre);
if (isset($Nombre)) {
echo ("El nombre existe!!!");
}
else {
echo ("El nombre ya no existe!!!");
}
?>
```

El resultado es el siguiente:

```
El nombre existe!!!
El nombre ya no existe!!!
```

## gettype()

Con esta función podemos averiguar el tipo de dato almacenado en la variable. Nos puede devolver uno de los siguientes valores:

- *integer*
- *double*
- *string*
- *array*
- *object*
- *class*
- *unknown type*

```
<?php
$correo = "luis@nccextremadura.org";
echo "la variable correo es del tipo: ";
```

```
echo gettype($correo);
?>
```

## settype()

Convierte el tipo de la variable al especificado en la función. El tipo debe especificarse eligiendo uno de los siguientes: *array, double, integer, object o string.* Si la función no es capaz de convertir el tipo de la variable devuelve el valor *false.*

```
<?php
$correo = "luis@nccextremadura.org";
if (settype($correo,"integer")) {
   echo ("Variable correo convertida a Entero<br>");
}
else {
   echo ("Imposible convertir al tipo Entero<br>") ;
}
echo ("Valor actual de correo es $correo");
?>
```

Al tratar de convertir un tipo string (una cadena de caracteres) a un entero, PHP 5 comprueba si existe algún número. Si no existe cambia el valor a O y la función settype () la evalúa como correcta.

## empty()

Comprueba si una variable está vacía, no existe, o su valor es 0.

```
<?php
$correo = "luis@nccextremadura.org";
if (empty($nombre)) {
   echo ("La variable nombre no existe");
}
$numero_entero = 0;
if (empty($numero_entero)) {
   echo ("La variable numero_entero no existe o tiene el valor
0");
}
?>
```

## isjntegerf), is_double(), is_string()

Estas funciones devuelven *true* si la variable pasada coincide con el tipo que indica la función.

Si la variable $numero_entero se evalúa con la función is_integer (), devolverá *true*.

```
<?php
$numero_entero = 0;
if (is_integer()($numero_entero)) {
   echo ("numero_entero es del tipo integer");
}
?>
```

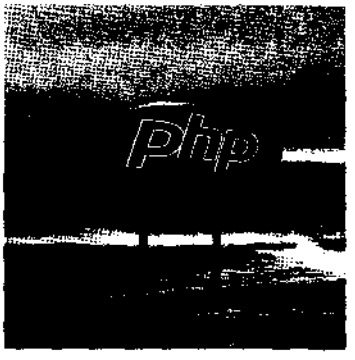
## intval(), doublevalQ, strval()

Convierte el valor de una variable al tipo indicado en la función. Esta función no permite la conversión a tipos *object* o *array*.

```
<?php
//Conversión de un tipo string a un integer
$cadena = "232";
echo "El tipo de la variable cadena es
".gettype($cadena)."<br>";
$numero = intval($cadena);
echo ("el numero es $numero");
?>
```

# Resumen

La base de la programación reside en la creación de datos y su procesamiento. En este capítulo hemos visto cómo declarar variables de diferentes tipos y cómo utilizar funciones para averiguar las propiedades de cada una de ellas. Además de las variables, las constantes juegan un papel muy importante, sobre todo las predefinidas por PHP 5, que nos dan información muy útil con la que poder trabajar.




# Capítulo 3

# Operadores

**En este capítulo aprenderá a:**

- Utilizar los diferentes tipos de operadores existentes.
- Distinguir entre operadores unarios, binarios y ternarios.
- Comprender la preferencia de ejecución entre operadores.

# Introducción

En el capítulo anterior hemos visto ejemplos de código escrito en PHP que utilizan símbolos muy comunes en el ambiente matemático e informático.

Estos símbolos, que aparecen entre variables o dentro de funciones, son conocidos como operadores.

Existen varios tipos de operadores:

- De asignación.
- Unario.
- Aritméticos.
- De comparación.
- Lógicos.
- Ternario.
- Bit a bit.
- Asignación combinados.
- De ejecución.
- Supresión de errores.

# Operador de asignación

El más básico es el símbolo de asignación (=), utilizado para dar valores a las variables que usamos en nuestro código.

```
<?php
$variable = 34;
$variable2 = "Asignación de valores";
?>
```

Las variables que están a la izquierda del operador toman el valor que se encuentra en la expresión de la derecha.

**Advertencia:**

*No hay que confundir el operador de asignación (=) con el operador de comparación (==), sobre todo en bucles de control, donde sería difícil encontrar algún fallo.*

## Operador Unario

El signo menos (-) se utiliza delante de un número o variable numérica. Este operador tiene la propiedad de hacer a los números, negativos o positivos, dependiendo del signo actual.

```
<?php
$entero = 23;
$entero_negativo = -$entero; // El valor es ahora -23
entero2 = -$entero__negativo; // El valor cambia ahora a 23
?>
```

## Operadores Aritméticos

Este tipo de operadores forman parte de la aritmética básica. Nos resultará familiar porque son símbolos muy utilizados en el aprendizaje de las matemáticas.

**Tabla 3.1.** Operadores aritméticos.

| Ejemplo | Nombre | Resultado |
|---|---|---|
| $a + $b | suma | Suma las dos variables. |
| $a - $b | resta | Hace la diferencia de las dos variables. |
| Sa * $b | multiplicación | Producto de las variables. |
| $a / $b | división | Cociente entre las dos variables. |
| $a % $b | Módulo | Resto de la división de $a entre $b. |

**Nota:**

*PHP 5 ignora los espacios en blanco entre las variables y los operadores. Aunque* $a + $b *es equivalente a* $a+$b, *es preferible utilizar la primera forma de escribir, porque se hace más legible.*

## Operadores de comparación

En algunos ejemplos del capítulo anterior puede ver que se utiliza la estructura de control if . else. Como veremos más adelante, esta estructu-

ra compara dos valores y elige el camino a seguir. El valor de la comparación siempre es *true ofalse.*

**Tabla 3.2.** Operadores de comparación.

| Ejemplo | Nombre | Resultado |
|---|---|---|
| $a == $b | Igualdad | Devuelve true si $a y $b son iguales. |
| $a === $b | Identidad | Verdadero si son iguales y del mismo tipo. |
| $a != $b | Distinto | True si son distintos. |
| $a <> $b | Distinto | True si son distintos. |
| $a < $b | Menor que | Cierto si $a es menor que $b. |
| $a > $b | Mayor que | Cierto si $a es mayor que $b. |
| $a <= $b | Menor o igual | Correcto si $a es menor o igual que $b. |
| $a >= $b | Mayor o igual | Correcto si $a es mayor o igual que $b. |

```
<?php
$a = 23; // Asignación de los valores
$b = 75;
if ($a >= $b) { //La condición no se cumple. El resultado es false
   echo "Esta parte no se ejecuta";
} else {
   echo "La comparación es false porque $a es menor que $b";
}
?>
```

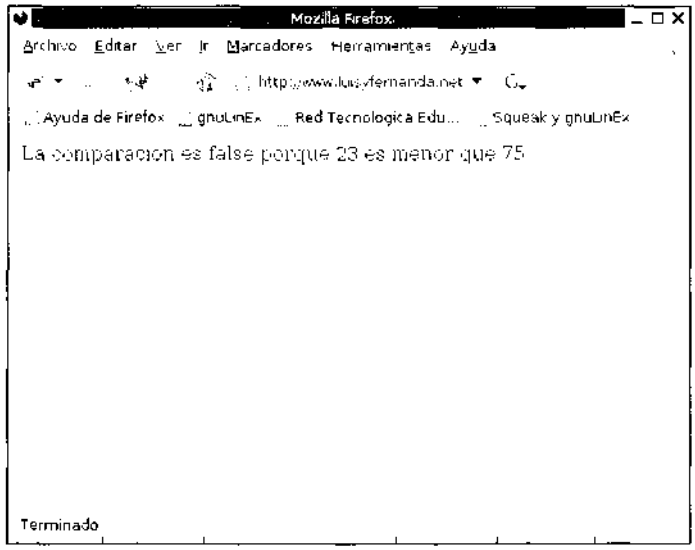



**Figura 3.1.** Resultado de una comparación.

# Operadores Lógicos

Durante el desarrollo de su proyecto, puede encontrarse con situaciones en las que necesite hacer varias comparaciones seguidas para que se cumpla una determinada condición. PHP 5 permite unir todas las comparaciones en una mediante el uso de los operadores lógicos.

**Tabla 3.3.** Operadores lógicos.

| Ejemplo | Nombre | Resultado |
|---|---|---|
| expresionl and expresion2 | Y | Si las dos expresiones son verdaderas el valor es true. |
| expresionl or expresion2 | O | Si una de las expresiones es verdadera el valor es true. |
| expresionl xor expresion2 | O exclusivo | True si una expresión es verdadera y la otra falsa. |
| ! expresionl | Negación | Verdadero si la expresión no es cierta. |
| expresionl && expresion2 | Y | Si las dos expresiones son verdaderas el valor es true. |
| expresionl \|\| expresion2 | O | Si una de las expresiones es verdadera el valor es true. |

El listado siguiente muestra dos condiciones anidadas:

```
<?php
$a = 2 3;
Sb = 75;
$c = true;
if ($a < $b) {
   if ($c) {
        echo ("Se cumplen las dos condiciones");
   }
>
?>
```

E>ta expresión se puede hacer más legible utilizando algún operador lógico:

```
<?php
$a = 23;
3b = 75;
$c = true;
```

```
if ($a < $b and $c) {
   echo "Se cumplen las dos condiciones";
}
?>
```

## Operador Ternario

Los operadores que hemos visto hasta ahora son capaces de manejar un operando (Unarios) o dos operandos (binarios). El operador ternario, o de comparación, evalúa un operando y, dependiendo de si es falso o verdadero, evalúa el segundo operando o el tercero.

La expresión que se quiere evaluar se escribe delante de un símbolo (?), después la expresión que tiene que ejecutarse si la evaluación anterior es *true,* seguida del símbolo (:) con la expresión que debe ejecutarse si es *false.*

```
<?php
$valor = false;
$valor == true ? $resultado = "OK" : $resultado = "FALLO";
// Si $value es true $resultado será OK
// Si es false $re sultado será FALLO
echo $resultado;
?>
```

## Operadores bit a bit

Estos operadores son complicados de entender si no conoce la lógica binaria. Afortunadamente, se utilizan en muy pocas ocasiones. Los operadores de *bit* utilizan las variables a nivel bajo, tal y como se almacenan en memoria física y comparan *bit* a *bit* los valores. Lo mejor es verlo con un ejemplo.

```
<?php
$a = 4 ; // Valor binario 100
$b = 5; // Valor binario 101
$c = $a & $b;
echo $c; // El valor de c es 100
?>
```

El operador binario Y (símbolo &) compara *bit* a *bit* las variables $a y $b. Si los *bits* de una misma posición son *true* (tienen el valor 1), el *bit* resultado es 1. En este caso, sólo existe una pareja de *bits* que es igual a *true* (sus

dos valores son 1), por lo tanto el valor de la variable $c es 4 (en binario 100).

**Tabla 3.4.** Operadores bit a bit.

| Ejemplo | Nombre | Resultado |
|---|---|---|
| $a & $b | Y | Si las parejas de bits son verdaderas el resultado es verdadero. |
| $a \| $b | O | Si algún bit de la pareja es verdadero el resultado es verdadero. |
| $a ^ $b | O exclusiva | Si un bit de la pareja es true y el otro false el resultado es verdadero. |
| ~$a | No | Los bits 1 se vuelven 0 y viceversa. También cambia el bit que se refiere al signo positivo o negativo. |
| $a << $b | Desplazamiento a la izquierda de bits | Desplaza a la izquierda los bits de la variable $a tantos bits como indique la variable $b. |
| $a >> $b | Desplazamiento a la derecha | Desplaza a la derecha los bits de la variable $a tantos bits como indique la variable $b. |

Nota:

*Puesto que la notación binaria se escapa del enfoque de este libro, le recomiendo algunas lecturas posteriores referidas a la lógica de* bit, *si es que tiene curiosidad. Probablemente no utilizará estos operadores nunca, pero eso depende de la magnitud de los proyectos a los que se enfrente.*

# Operadores de asignación combinados

En numerosas ocasiones se nos presentan situaciones en las que una variable debe incrementar o disminuir su valor en 1.

```
<?php
$a =23;
$a = $a +1; //Incrementamos en 1 el valor de la variable
?>
```

HP 5 provee operadores combinados que permiten asignar rápidamente
'crementos de valor, concatenaciones de caracteres, etcétera.

**Tabla 3.5.** Operadores de asignación combinados.

| Ejemplo | Nombre | Equivalencia |
|---|---|---|
| $a++ | Incremento | $a = $a +1 |
| $a~ | Decremento | $a = $a -1 |
| ++$a | Incremento | $a = $a +1 |
| ~$a | Decremento | $a = $a -1 |
| $a += $b | Suma | $a = $a + $b |
| $a -= $b | Resta | $a = $a - $b |
| $a *= $b | Multiplicación | $a = $a * $b |
| $a /= $b | División | $a = $a / $b |
| $a %= $b | Módulo | $a = $a % $b |
| $a &= $b | Y | $a = $a & $b |
| $a \|= $b | O | $a = $a \| $b |
| $a ^= $b | O exclusiva | $a = $a ^ $b |
| $a .= $b | Concatenación | $a = $a . $b |
| $a >>= $b | Desplazamiento a la derecha | $a = $a >> $b |
| $a <<= $b | Desplazamiento a la izquierda | $a = $a << $b |

**Advertencia:**

*Como vemos en la tabla 3.5, ¡os operadores de incremento y decremento pueden aparecer a la derecha o a la izquierda de un variable. La posición del operador influye en el orden en el que la variable se asigna o se incrementa. Mejor verlo en un ejemplo.*

```
<?php
$a = 23;
$b = $a++; // $b es igual a 23 porque $a se incrementa
después de la asignación

$a = 2 3;
$c = + + $a; // $c es igual a 24 porque $a se incrementa
antes de la asignación

echo "la variable b es: $b, y c es: $c";
?>
```

# Operador de ejecución

Si tiene experiencia con la programación en *shell* de Unix o gnuLinux, sabrá que el apostrofe invertido sirve para ejecutar comandos del sis-

tema. PHP 5 ha adoptado esta nomenclatura y funciona exactamente igual.

```
<?php
$listado_archivos = "Is -la~; //Hacemos un listado de los
ficheros del directorio actual
echo $listado_archivos; // y lo sacamos por pantalla
?>
```

Nota:

*El comando* ls *del ejemplo anterior es propio de sistemas Unix y gnuLinux. Sí hace pruebas sobre un sistema basado en el Sistema Operativo Windows, el ejemplo anterior no funcionaría, tendría que cambiar el comando por* dir.

## Operador de supresión de errores

La mayoría de las funciones que se utilizan en PHP 5 muestran errores en el navegador cuando algo falla.

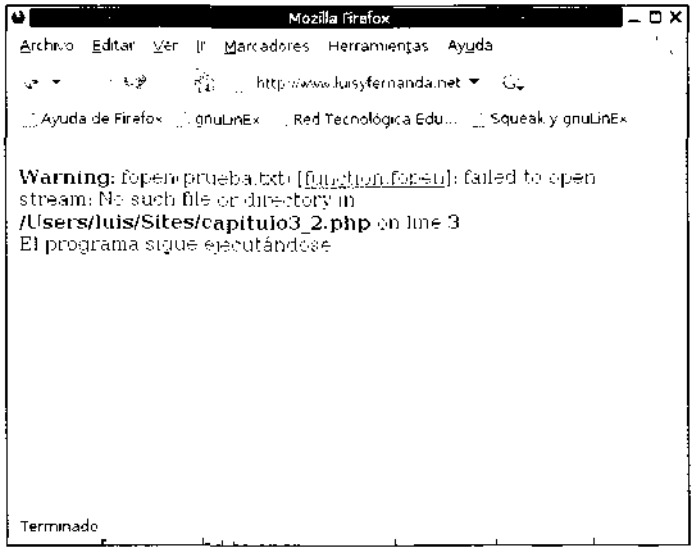



**Figura 3.2.** Error al intentar abrir un fichero que no existe.

Si intenta abrir un fichero que no existe, PHP mostrará un mensaje de error y continuará la ejecución del programa.

El operador (@), colocado delante de una función, evitará que se muestre el error.

```
<?php
$fichero = fopen("prueba.txt", "r") ; //Muestra un error en el
navegador
echo ("El programa sigue ejecutándose");
?>
```

El código siguiente oculta la salida de error por pantalla:

```
<?php
$fichero = @fopen("prueba.txt","r");
echo ("El programa sigue ejecutándose");
?>
```

## Precedencia de Operadores

Vamos a echar un vistazo al código siguiente:

```
<?php
$resultado = 20 + 10 * 10;
echo $resultado;
?>
```

Aparentemente es un código muy simple, pero encierra un problema matemático.

Existen dos resultados posibles en función del operador que debe ejecutarse antes.

Si la suma se ejecuta antes, la variable $resultado tendrá el valor 300 y si la multiplicación se ejecuta antes, tendremos que $resultado equivale a 120.

Si comprobamos el código, el resultado que nos muestra es 120, por lo tanto el operador de multiplicación (*) es preferente con respecto al operador de suma (+).

Si los operadores que aparecen son idénticos, existe un orden de ejecución que puede ser desde la izquierda a la derecha o derecha a izquierda.

```
<?php
$resultado  = 2 0 / 1 0 / 2 ;
echo $resultado;
?>
```

El valor de $resultado es ahora 1, porque primero se calcula 20 / 10 y después ese mismo resultado se divide entre 2, es decir, la asociación entre operadores de división es desde la izquierda.

**Tabla 3.6.** Orden de preferencia de los operadores.

| Operador | Operación | Asociación |
|---|---|---|
| () | Paréntesis de preferencia | N/A |
| new | Instancia de objeto | N/A |
| [] | array | Derecha |
| ! | NO lógico | Derecha |
| | Signo menos | Derecha |
| ++ -- | Incremento, decremento | Derecha |
| @ | Supresión de errores | Derecha |
| * / % | Multiplicación, división y módulo | Izquierda |
| + - . | Suma, resta, concatenación | Izquierda |
| << >> | Desplazamiento izquierda y derecha | Izquierda |
| < <= > >= | Menor que, menor o igual, mayor que, mayor o igual | N/A |
| == != | Igual, no igual | N/A |
| & | Y | izquierda |
| ^ | O exclusivo | Izquierda |
| \| | O | Izquierda |
| && | Y lógico | Izquierda |
| \|\| | O lógico | Izquierda |
| ?: | condicional | Derecha |
| = += .= *= /= = %= &= != -= <<= >>= | Asignación | Derecha |
| and | Y lógico | Izquierda |
| xor | O exclusivo lógico | Izquierda |
| or | O lógico | Izquierda |

El operador paréntesis tiene la propiedad de dar preferencia en la evaluaron a todos los operadores que contiene. El siguiente ejemplo muestra :ómo podemos utilizar correctamente el operador paréntesis.

```
<?php
$resultado  = 2 0 / 1 0 + 2 ;
echo $resultado;  // El resultado es 4
$resultado = 20 / (10 + 2);
echo $resultado;  // El resultado es 1,66666667
?>
```

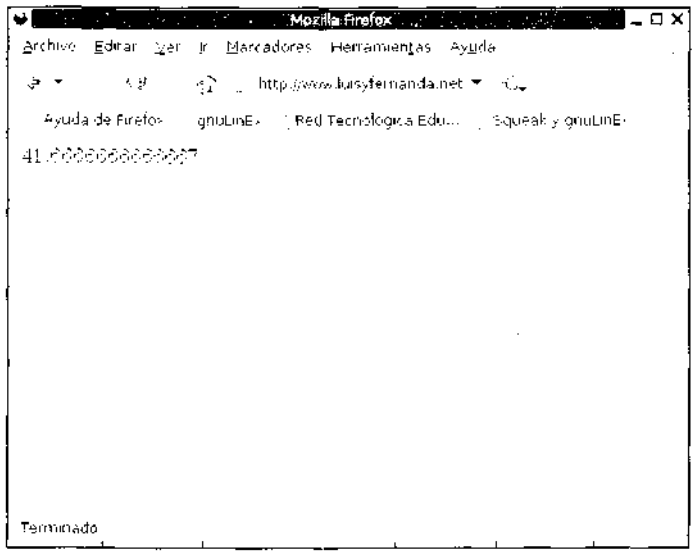
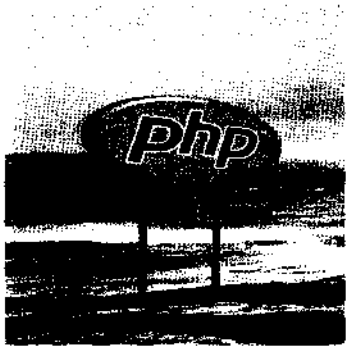



**Figura 3.3.** Resultado de la preferencia de operadores.

## Resumen

Los operadores son responsables de todas las operaciones que realizamos en nuestros programas. Por eso, es importante conocer las propiedades v los casos de uso de cada uno de ellos para poder sacarles el máximo partido. Sin embargo, si ya conoce algún lenguaje de programación, la intuición le hará utilizarlos adecuadamente sin pararse a pensar en el tipo de operador.

Por último, debe prestar un especial interés a la preferencia de uso ya que, una mala utilización redundará en un error muy difícil de controlar.




**En este capítulo aprenderá a:**

- Utilizar las estructuras de control de elección simple y múltiple.
- Crear bucles de diferentes tipos.
- Crear sus primeros programas.
- Conocer las sentencias para interrumpir un bucle.

# Introducción

Es difícil imaginar un programa sin estructuras de control. Éstas nos permiten elegir diferentes caminos en función de los datos que evaluamos en cada momento. En este capítulo trataremos dos tipos de estructuras de control:

- Estructuras de elección.
- Estructuras de bucle.

Las estructuras de elección permiten evaluar una condición o varias y elegir el camino correcto. Las de bucle repiten un número determinado de veces un conjunto de instrucciones.

# Estructuras de elección

Existen dos tipos:

- Elección simple.
- Elección múltiple.

## if-else

La sintaxis de esta estructura es:

```
if (condición) instrucción;
```

Si se cumple la condición se ejecuta la instrucción que le sigue. Si quiere que se ejecuten varias instrucciones, debe utilizar el símbolo llave ( { instrucciones ¡ ).

```
if (condición) {
   instruccionl;
   instruccion2¡
   instruccion3;
}
```

La estructura if puede ampliarse para que se pueda elegir entre condición verdadera y falsa.

```
if (condición) {
   instruccionl;
   instrucción2;
   instruccion3;
} else {
```

```
    instruccionl,-
    instrucción2;
    instruccion3;
}
```

Si la condición es verdadera se ejecutan las instrucciones inmediatamente después del if y si la condición es falsa se ejecutan las instrucciones dispuestas después de la palabra `else`. Puede ver un ejemplo completo:

```
<?php
$valor = 23;
$valor2 = 27;
if ($valor < $valor2) {
   echo "La variable valor es menor que valor2";
} else {
   echo "La variable valor es mayor que valor2";
}
?>
```


**Figura 4.1.** Resultado de una comparación con if-else.

## elseif

Es muy común hacer comparaciones en cascada para comprobar varios valores:

```
<?php
$dia=4;
if ($dia == 1) {
```

```
    echo "El día es Lunes";
} else {
    if ($dia == 2) {
        echo "El día es Martes";
    } else {
        if ($dia == 3) {
            echo "El día es Miércoles";
        } else {
            if ($dia == 4) {
                echo "El día es Jueves";
            }
        }
    }
}
?>
```

El patrón anterior es tan común, que existe una estructura especial para manejarlo.

Puede escribir el ejemplo anterior con la palabra reservada elseif de la siguiente forma:

```
<?php
$dia=4;
if ($dia == 1) {
    echo "El día es Lunes";
} elseif ($dia == 2) {
    echo "El día es Martes";
} elseif ($dia == 3) {
    echo "El día es Miércoles";
} elseif ($dia == 4) {
    echo "El día es Jueves";
}
?>
```

El ejemplo anterior evalúa la variable $dia hasta que encuentra el valor correcto y ejecuta las instrucciones.

## switch

La construcción switch comprueba el valor de una expresión y permite elegir entre un conjunto de instrucciones. El formato es el siguiente:

```
switch     (expresión)  {
           case   valorl :
                  instruccionl;
                  instruccion2;
                  instruccion3;
                  break;
```

```
   case     valor2 :
            instruccionl;
            instruccion2;
            instruccion3;
            break;
   case     valor3:
            instruccionl;
            instruccion2;
            instruccion3;
            break;
   default:
            instruccionl;
            instruccion2;
            instruccion3;
}
```

La expresión puede ser de cualquier tipo, siempre que devuelva un valor de tipo entero, de coma flotante o de cadena de caracteres. Una vez evaluada la expresión, se busca el valor en la instrucción `case` y, si coincide, se ejecutan todas las instrucciones hasta la palabra reservada `break`. Si no coincide ningún valor, se ejecutan las instrucciones por defecto.
Veamos un ejemplo completo:

```
<?php
$dia = 4 ;
switch ($dia)  {
   case 1:
        echo "El día es Lunes";
        break;
   case 2 :
        echo "El día es Martes";
        break;
   case 3 :
        echo "El día es Miércoles";
        break;
   case 4 :
        echo "El día es Jueves";
        break;
   case 5:
        echo "El día es Viernes";
        break;
   case 6 :
        echo "El día es Sábado";
        break;
   case 7:
        echo "El día es Domingo" ;
        break;
   default:
```

```
        echo "El día de la semana es incorrecto";
}
?>
```

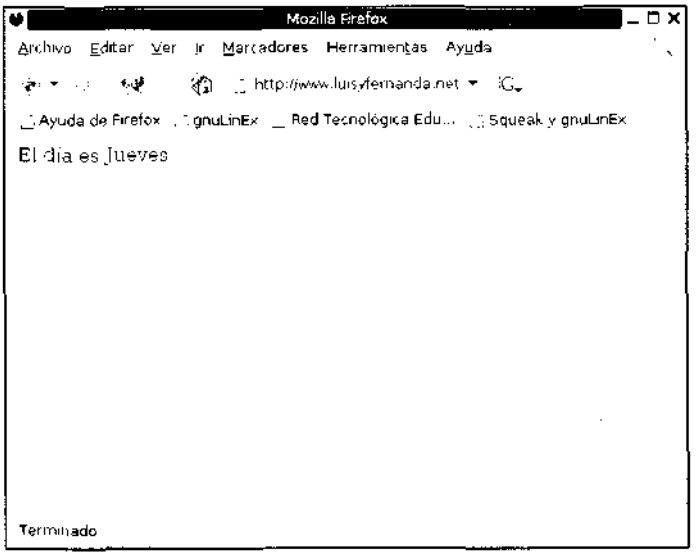



**Figura 4.2.** Comparación múltiple con switch.

# Bucles

Los bucles son estructuras de control que permiten repetir varias veces las mismas instrucciones.

## while

El bucle while es el más básico de todos. La construcción básica es la siguiente:

```
while (condición) {
instrucción1•
instrucción2;
instrucción3;
}
```

La condición se evalúa al principio.

Si es verdadera, se ejecutan las instrucciones que están dentro del bucle y se vuelve a evaluar la condición.

Si la condición es falsa no se ejecutan las instrucciones y se continúa con el desarrollo del programa.

Puesto que la condición se evalúa antes de ejecutar las instrucciones, es posible que algunos bucles no se ejecuten ninguna vez.

El siguiente ejemplo muestra una instrucción while que no se ejecuta nunca, porque la condición es falsa:

```
<?php
$variable = false;
while ($variable) {
   echo "Esta linea no se ejecuta nunca";
}
?>
```

Existe la posibilidad de que un bucle se ejecute infinitas veces, si dentro de las instrucciones no existe nada que cambie la condición que se evalúa al principio.

```
<?php
$variable = true;
while ($variable) {
   echo "CUIDADO: Esta línea se ejecuta siempre";
}
?>
```

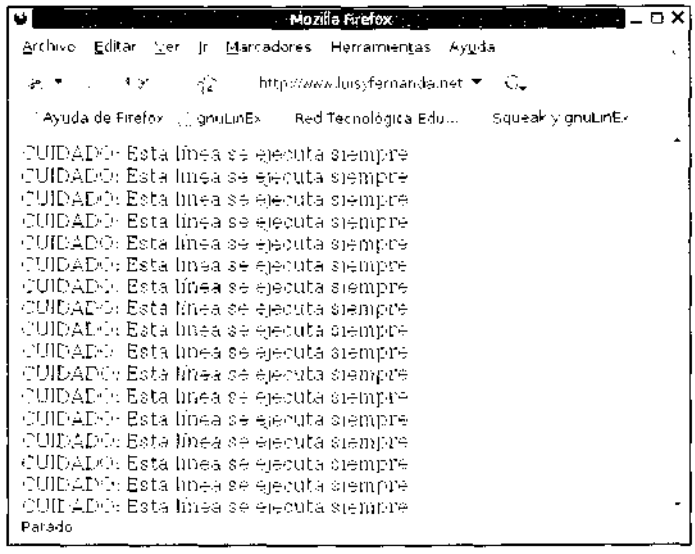



**Figura 4.3.** Comparación múltiple con switch.

El bucle while se puede utilizar para ejecutar un número determinado de veces las instrucciones que implica.

Véase el ejemplo de la serie de Fibonacci.

```
<?php
$numero_anterior = 1;
$numero_posterior = 1;
$serie = 1;
$fin = 10000;
echo "Serie de Fibonacci:";
while ($serie < $fin) {
echo $serie.", ";
$serie = $numero_anterior + $numero_posterior;
$numero_anterior = $numero_posterior;
$numero_posterior = $serie;
}
?>
```

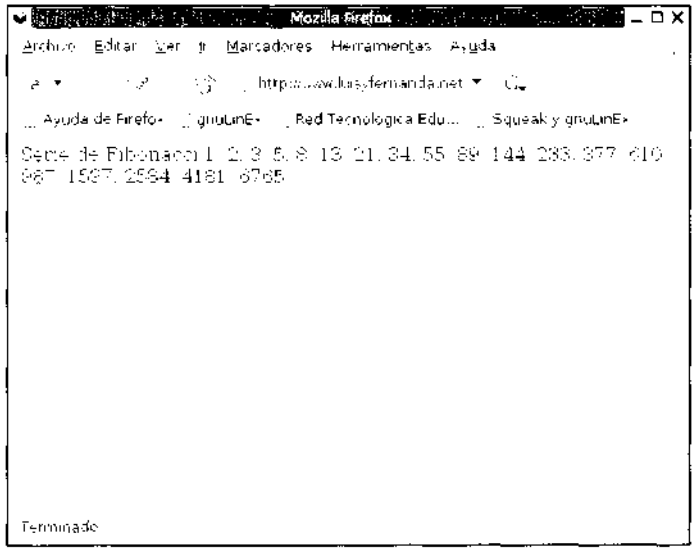



**Figura 4.4.** Serie de Fibonacci.

Cada número de la serie de Fibonacci se forma sumando los dos número anteriores.

En el ejemplo, la condición es que el número de la serie sea menor que el número que marca la variable $fin, es decir, si el número que indica la serie es mayor que 10.000 el bucle finaliza.

La variable $fin sirve de valor máximo y puede cambiarlo en función de la cantidad de números que quiera tener en pantalla.

## do-while

Este bucle es igual que el anterior, pero la condición se evalúa al final de las instrucciones. Por lo tanto, el código que está entre las llaves se ejecuta al menos una vez. El formato básico es el siguiente:

```
do {
   instrucciónl;
   instrucción2;
   instrucción3;
}
while (condición);
```

Puede comprobar que la diferencia es mínima cuando utiliza esta estructura de control para la serie de Fibonacci.

```
<?php
$numero_anterior = 1;
$numero_posterior = 1;
$serie = 1;
$fin = 6765;
echo "Serie de Fibonacci:";
do {
   echo $serie.", ";
   $serie = $numero_anterior + $numero_posterior;
   $numero_anterior = $numero_posterior;
   $n'umero_posterior = $serie;
}
while ($serie < $fin);
?>
```

## for

La construcción de bucle más complicada es la del for. Tiene la siguiente sintaxis:

```
for (expresión inicial; condición de fin; expresión de fin) {
   instrucciónl;
   instrucción2;
   instrucción3;
}
```

El funcionamiento es muy sencillo. La expresión inicial se ejecuta una sola vez al principio del bucle.

La condición de fin se evalúa cada vez que se ejecuta el bucle. Si es verdadera se continúa la ejecución y si es falsa se sale del bucle. Al final de cada interacción se ejecuta la expresión de fin.

Puede simular la instrucción for con una instrucción while de esta forma:

```
expresión inicial;
while (condición de fin) {
   instrucciónl;
   instrucción2;
   instrucción3;
   expresión de fin;
}
```

Otra forma poco corriente es eliminar alguna o todas las expresiones de. bucle. Si quita todas las expresiones se crea un bucle infinito:

```
fOr(;;){
   instrucciónl;
   instrucción2;
   instrucción3;
}
```

equivale a:

```
while (true) {
   instrucciónl;
   instrucción2;
   instrucción3;
}
```

El caso contrario es añadir más de una condición en cada clase de expresión, separadas por comas. En este caso, la condición de fin se evaluara : falso si alguna de las cláusulas es falsa. En realidad funciona como un C lógico.

```
<?php
for ($x = 1 , $y = 1, $z = 1; $y < 10, $z < 10; $x++, $y = $y -
2, $z = $z + 3) {
   echo ("$x, $y, $z<br>");
}
?>
```

Un ejemplo clásico es la tabla de multiplicar:

```
<html>
<body>
<b>TABLA DE MULTIPLICAR</b>
<table border="l">
<?php
echo ("<tr>");
echo ("<thx/th>" ) ;
for ($cabecera = 1; $cabecera <= 10; $cabecera++) {
        echo ("<th>") ;
        echo $cabecera;
```

```
        echo  ("</th>") ;
}
echo  ("</tr>");
for  ($x = l;$x <= 10;  $x++ )  {
        echo  ("<tr>") ;
        echo  ("<th>") ;
        echo $x;
        echo  ("</th>");
        for  ($y = l;$y <= 10;  $y++ )  {
                  $multiplicacion = $x * $y;
                  echo  ("<td>") ;
                  echo  ("$multíplícacíon");
                  echo  ("</td>");
    }
   echo  ("</tr>");
>
?>
</table>
</body>
</html>
```

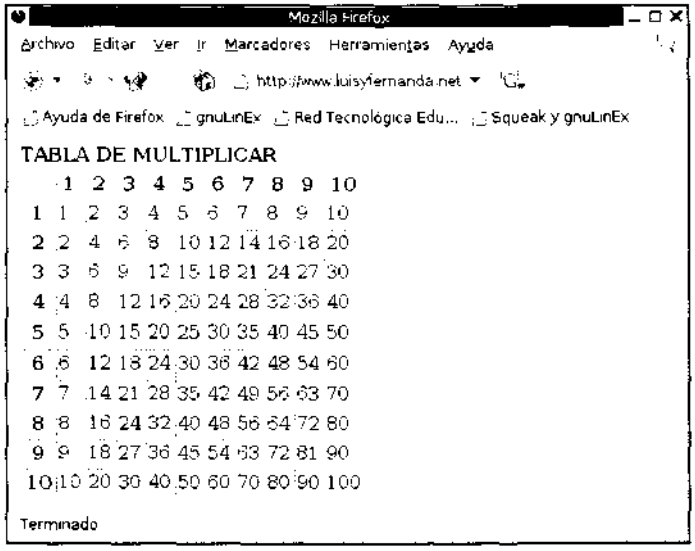



**Figura 4.5.** Tabla de multiplicar.

*['*, ejemplo se sirve de un bucle `for`, que crea una cabecera de tabla con la ::queta especial <th>, donde inserta los números del uno al diez. Estos umeros escritos en horizontal se podrán cruzar con los verticales para atener así el resultado de una multiplicación.

El siguiente paso es un conjunto de dos bucles anidados. El primer bucle almacena en la variable $x los valores numéricos que aparecerán como cabecera de la columna uno. El segundo bucle será el encargado de multiplicar el valor actual de $x por los valores del 1 al 10.

Cada vez que se ejecuta el segundo for se obtiene una celda nueva con un valor determinado por la multiplicación y cada vez que se ejecuta el primer bucle, aparece una nueva fila cuya cabecera es un número del 1 al 10.

## break y continué

El camino ordinario para salir de un bucle es que la condición se evalúe a *false*. Existe una forma especial de salir de un bucle, utilizando las palabras reservadas break y continué. Su forma de actuar es la siguiente:

- **break:** Sale del bucle actual y continúa el programa.
- **continué:** Salta hasta el final de la interacción y continúa la ejecución desde el principio del bucle.

El código siguiente muestra la forma de utilizar break:

```
<?php
for ($x = 1; $x < 20; $x++) {
   if ($x == 10) {
        break;
   } else {
        echo "$x<br>";
   }
}
```

La salida por pantalla es una sucesión de números del 1 al 9. Al llegar la variable $x al número 10, la condición del if se cumple y se ejecuta la instrucción break, saliendo del bucle. Veamos ahora el mismo ejemplo, pero utilizando continué:

```
<?php
for ($x = 1; $x < 20; $x++) {
   if ($x == 10) {
        continué;
   } else {
        echo "$x<br>";
   }
}
?>
```

En este caso, cuando $x alcanza el valor 10,1o que hace la instrucción continué es ir hacia el final de bucle y volver al principio comprobando la condición.

El resultado por pantalla es una sucesión de números desde el 1 hasta el 19, sin incluir el número 10.

## Finalizar la ejecución de un programa

Hay veces que necesitamos parar la ejecución de un programa por diversas causas: ha ocurrido un error, un fallo en la entrada del nombre y la contraseña. La función para la ejecución es exit() o die(). Estas dos funciones aceptan un parámetro que se imprime en pantalla. Por ejemplo, considere el siguiente código que asume la conexión a una base de datos:

```
<?php
$conexion = conectar_base_datos("libros");
if ( !$conexion) {
   die ("Se ha producido algún error en la conexión");
}
?>
```

Se puede apreciar que si la variable $conexión se evalúa *false* (no se ha conseguido la conexión), se detiene la ejecución del programa. Una versión más compacta se consigue utilizando el operador or:

```
<?php
$conexion = conectar_base_datos("libros") or
die ("Se ha producido algún error en la conexión");
?>
```

**Nota:**

*exit() y die() eran ¡as únicas alternativas para el control de errores críticos en versiones anteriores de PHP. En PHP 5, afortunadamente, se ha incluido un potente manejo de excepciones que hace la ejecución más flexible.*

## Sintaxis alternativa

Algunas estructuras de control pueden escribirse de una forma distinta. Las estructuras if, while, for y switch pueden escribirse sustituyen-

do la llave de inicio por un símbolo de dos puntos (:) y la llave de cierre por endif, endwhile, endfor y endswitch, respectivamente.

Como ejemplo podemos ver la estructura if:

```
<?php
$variable = 10;
if ($variable == 10) {
   echo "Nueva estructura de control";
}
?>
```

El código siguiente es equivalente al anterior:

```
<?php
$variable = 10;
if ($variable == 10):
   echo "Nueva estructura de control";
   endif;
?>
```

## Resumen

Si ha llegado hasta aquí, ya conocerá los secretos básicos de la programación en PHP 5. Como puede darse cuenta, lo único necesario para hacer un programa es tener claro la forma de utilizar las variables, los operadores y las estructuras de control. Una vez dominado esto, lo demás es aprender funciones de PHP y técnicas de recuperación e inserción de datos.

Los capítulos siguientes hacen un estudio más concienzudo de las variables y las funciones que facilitan el proceso de interacción entre ellas.


# Capítulo 5

# Funciones

**En este capítulo aprenderá a:**

- Crear sus propias funciones.
- Utilizar parámetros fijos y variables.
- Realizar funciones recursivas.
- Separar el código fuente en varios ficheros.

## Introducción

Las funciones son grupos de instrucciones independientes que tienen un propósito determinado.

Así, tenemos funciones que calculan la raíz cuadrada de un número, nos dan un número aleatorio o permiten contar los caracteres de una palabra. La sintaxis básica de una función es la siguiente:

```
nombre_función(parámetrol, parámetro2, parámetro3...parámetro_n)
```

Las funciones pueden ser llamadas con varios parámetros o con ninguno, dependiendo de su definición.

Cuando PHP encuentra en el código la llamada a una función, primero, evalúa cada argumento y los utiliza como parámetro de entrada. Después, ejecuta la función y devuelve el valor solicitado o realiza alguna acción sin enviar ningún valor de salida.

El siguiente ejemplo contiene un conjunto de llamadas a funciones con 0,1 ó 2 parámetros de entrada:

```
<?php
echo sqrt(9);     // Raíz cuadrada de 9 es 3
echo rand(10,20); // Número aleatorio entre 10 y 20
echo pi();        //Número pi
?>
```

## Valores de las funciones

Cada función en PHP se considera como una expresión. Se pueden utilizar las funciones para lo mismo que se utilizan las expresiones, como almacenar su resultado en una variable o formar parte de una expresión más compleja.

```
<?php
$numero_aleatorio = rand (1,100);   // Se almacena en una
variable
$numero_aleatorio = rand (1,100) + rand (1,100) * sqrt(9); //
Expresión compleja
?>
```

No todas las funciones devuelven un valor numérico. También pueden devolver caracteres, *array* o *true/false* si la función ha tenido éxito o no. Este tipo de funciones se utiliza para conectar a bases de datos, escribir a ficheros de texto, manipular imágenes, etcétera.

# Función de ejemplo. Obtención de la fecha actual

Existen un conjunto de funciones que permiten averiguar la fecha actual del sistema. Una de estas funciones es date (), que permite recuperar el año, día, mes, etcétera, del momento en el que se hace la consulta. Para averiguar el día de hoy podemos utilizar la construcción:

```
<?
date ("d");
?>
```

El parámetro que pasamos entre paréntesis indica a la función el formato de fecha que queremos obtener. En este caso, la d indica que queremos obtener el día en formato numérico. La siguiente tabla relaciona los caracteres dentro de la función date () con su aplicación práctica:

**Tabla 5.1.** Formato de fecha con date().

| Carácter | Valor |
|---|---|
| a | Imprime "am" o "pm" |
| A | "AM" o "PM" |
| h | La hora en formato (01-12). |
| H | Hora en formato 24 (00-23). |
| g | Hora de 1 a 12 sin un cero delante. |
| G | Hora de 1 a 23 sin cero delante. |
| i | Minutos de 00 a 59. |
| s | Segundos de 00 a 59. |
| d | Día del mes (01 a 31). |
| j | Día del mes sin cero (1 a 31). |
| w | Día de la semana (0 a 6). El 0 es el domingo. |
| m | Mes actual (01 al 12). |
| n | Mes actual sin ceros (1 a 12). |
| Y | Año con 4 dígitos (2004). |
| y | Año con 2 dígitos (04). |
| z | Día del año (0 a 365). |
| t | Número de días que tiene el mes actual. |

Se pueden utilizar varios caracteres dentro de la función, indicando el formato de fecha que nos interesa.

```
<?php
$meses = array ("Enero", "Febrero", "Marzo", "Abril", "Mayo",
                "Junio", "Julio", "Agosto", "Septiembre",
                "Octubre", "Noviembre", "Diciembre");
echo "Fecha actual: " . date("d-m-Y") . "<br>";
echo "Día del año: " . date("z") . "<br>";
echo "Estamos en el mes: " . $meses[date("n")] . "<br>";
?>
```

El resultado en pantalla es el siguiente:

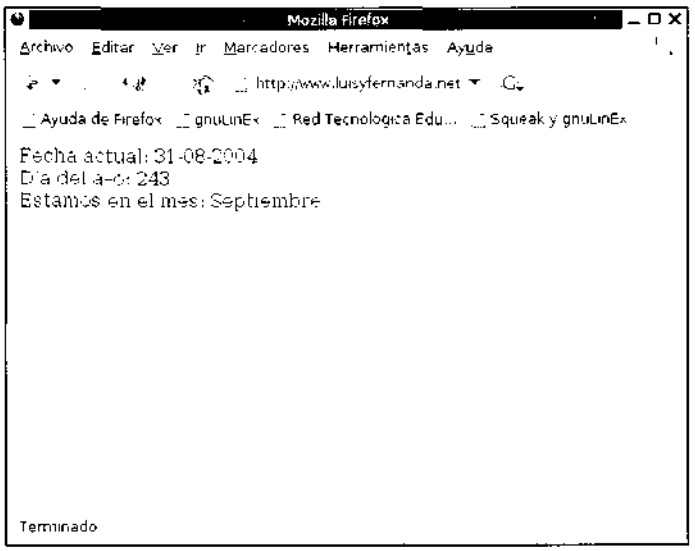



**Figura 5.1.** Función de fecha.

## Documentación sobre funciones

Desde sus inicios, PHP se diseñó para que fuera sencillo de utilizar y de extender por la comunidad de desarrolladores.

La potencia reside en la gran cantidad de funciones escritas de todo tipo.

Este libro cubre las funciones más utilizadas en el desarrollo de un proyecto profesional, pero sería imposible cubrir todas las funciones existentes con detalle. Por esta razón es imprescindible tener como referencia la

guía *on-line* de PHP en www.php.net, donde podemos encontrar clasificadas por temática todas las funciones existentes hasta ahora.

Además, contamos con un buscador de apoyo que nos permite buscar la definición de una función y ejemplos de uso escritas por usuarios de PHP.

Para los usuarios que no conocen la forma de trabajar con el lenguaje C, se hace necesario explicar cómo aparecen las funciones descritas en el manual de www.php.net. El formato es el siguiente:

```
tipo_devuelto nombre_función (tipol argl, tipo2 arg2...),-
```

El tipo-devuelto es el tipo de valor que la función dará como salida. Puede ser un *integer, double, array,* etcétera. El nombre de la función es la forma correcta de ejecutar el código. Los parámetros de entrada (argumentos) deben pertenecer al tipo que define la función. Un ejemplo de definición es:

```
string substr (string string, int start[, int length] ) ;
```

Puede ver que la función substr () devolverá como resultado una cadena de caracteres y que tiene 2 argumentos de entrada como mínimo. El parámetro que se encuentra entre dos corchetes es opcional y puede decidir usarlo o no.

# Funciones de usuario

Para realizar programas complejos en PHP 5, no es necesario escribir funciones. Se pueden escribir programas utilizando todo lo aprendido hasta ahora.

A medida que el proyecto se haga más complejo, el código se hará más extenso y menos legible. En este punto debería comenzar a pensar en crear funciones que realicen determinadas tareas.

## Definición de funciones

El formato es el siguiente:

```
function nombre_función($argumentol, $argumento2, ..)
{
   instrucciónl;
   instrucción2;
   instrucción3;

}
```

Como puede ver, se comienza con la palabra reservada function seguida del nombre de la función y el número de parámetros necesario. El bloque de código que pertenece a nuestra función debe introducirse entre dos símbolos de llave ( { } ). Un ejemplo clásico consiste en calcular el factorial de varios números. El factorial del número x se obtiene multiplicando el número x por el factorial de (x-1), teniendo en cuenta que el factorial de 1 es 1. Por ejemplo, el factorial de 3 sería 3 por el factorial de 2, es decir 3 * 2 * 1. El código sin utilizar funciones es:

```
<?php
$resultado = 1;
$factorial = 3;// Queremos calcular el factorial de 3
for ($x = $factorial; $x > 0; $x--) {
                ^resultado = $resultado * $x;
        }
echo "El factorial de $factorial es $resultado<br>",•

$resultado = 1;
$factorial = 4 ; // Queremos calcular el factorial de 4
for ($x = $factorial; $x > 0; $x--) {
                $resultado = $resultado * $x;
        }
echo "El factorial de $factorial es $resultado<br>";

$resultado = 1;
$factorial = 5 ; // Queremos calcular el factorial de 5
for ($x = $factorial; $x > 0; $x--) {
                $resultado = $resultado * $x;
        }
echo "El factorial de $factorial es $resultado<br>";
?>
```

Si se fija en el código anterior, podrá comprobar que sigue un patrón muy evidente. El cálculo del factorial se realiza en un bucle que va disminuyendo el valor de una variable y multiplicando todos los valores entre sí. Aprovechando este patrón puede crear una función que realice el factorial del número x ahorrando líneas de código:

```
<?php
function  factorial($numero)
{
   $resultado = 1;
   for ($x = $numero; $x > 0; $x--) {
        $resultado = $resultado * $x;
   }
   return $resultado;
}
echo "El factorial de 3 es ".factorial(3)."<br>";
```

```
echo "El factorial de 4 es ". factorial(4)."<br>" ;
echo "El factorial de 5 es ".factorial(5). "<br>";
?>
```

Si comparamos los dos ejemplos, podemos llegar a la conclusión de que el segundo es mucho más legible y más sencillo de realizar. Como puede ver, la función `factorial ()` hace uso de la palabra reservada `return`, que es la encargada de devolver el valor que estamos solicitando cuando se invoca la función.

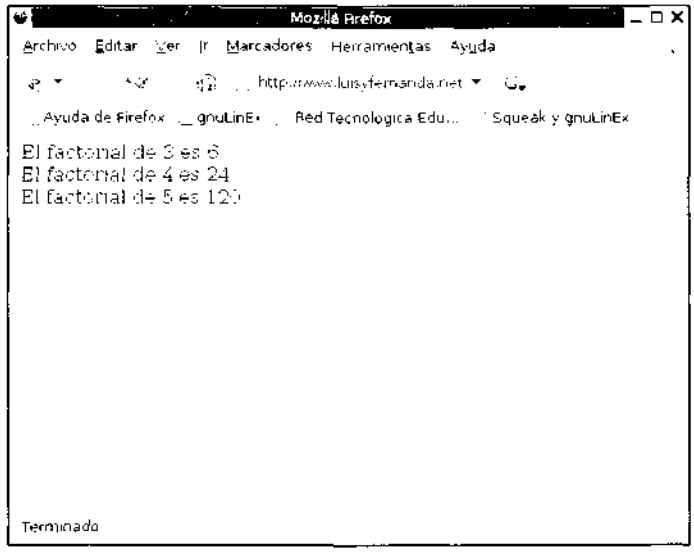



**Figura 5.2.** Factorial de un número.

**Nota:**

*Aunque en el ejemplo, la función* `factorial ()` *recibe como parámetro el número 3, 4 ó 5, también es posible llamarla con una variable del tipo* integer *o* double.

## Parámetros insuficientes

Si llama a una función con menos parámetros de los que debe utilizar por definición aparecerá el siguiente error por defecto:

```
Warning: Missing argument 1 for factorial()
```

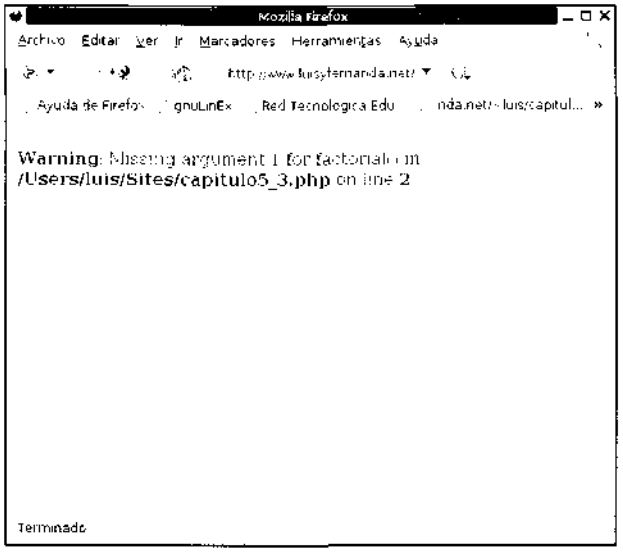



**Figura 5.3.** Argumentos insuficientes en una llamada.

Este tipo de error no es crítico y puede controlarse con excepciones, como veremos en capítulos sucesivos. Además de las excepciones podemos utilizar el operador de supresión de error @ delante de la función de esta forma: `@factorial()`.

No se recomienda eliminar los mensajes de error en este caso, ya que hace complicado la corrección de los posibles fallos.

## Parámetros en exceso

Si al contrario que en el apartado anterior le pasa a una función más parámetros que los definidos, éstos serán omitidos y la ejecución continuará normalmente sin ningún mensaje de error. Esta propiedad de PHP 5 permite utilizar variables con un número de parámetros variable.

# Ámbito de las variables

Las variables definidas en un archivo de PHP permanecen declaradas en todo el fichero, menos en las funciones. Las funciones crean un entorno de ejecución donde sólo pueden ver las variables que están dentro de las mismas. Aunque no parece muy lógico, esto nos permite tener variables

con el mismo nombre en diferentes funciones, haciendo el código reutilizable.

Si nos fijamos en el ejemplo:

```
<?php
$pi = 3.14;
function valor_pi()  {
   if (üsset($pi))  {
         $pi = pi();
   }
   return $pi;
}
echo valor_pi();
?>
```

Se puede observar que definimos, al comienzo del código, la variable $pi como un valor aproximado del número *pi* (3,14). La función `valor_pi()` comprueba si la variable $pi está definida, y si no lo está recupera ese valor con una función de PHP 5. Para entender el alcance de las variables vamos a ir paso a paso viendo qué es lo que ocurre:

- Se define $pi a nivel global en el programa.
- Se define la función `valor_pi()` que por ahora no se utiliza.
- Se lanza la ejecución de la función.
- La función evalúa si existe la variable $pi y da como resultado *false*. Esto es así porque la variable está definida fuera de la función y no dentro de ella, por lo tanto se considera que la variable $pi dentro de la función es distinta (entorno local) a la variable $pi definida fuera de la función (entorno global).
- El resultado por pantalla es 3.1415926535898.

Esta forma de programar se considera la correcta en la comunidad de desarrolladores de PHP. Aún así, existen mecanismos que permiten que una variable definida fuera de una función pueda ser utilizada dentro de la misma, simplemente anteponiendo la palabra reservada `global` a la variable definida dentro de la función.

Utilizando el ejemplo anterior tenemos:

```
<?php
$pi = 3.14;
function valor_pi()  {
   global $pi;
   if (Üsset($pi))  {
         $pi = pi();
   }
```

```
    return $pi;
}
echo valor_pi();
?>
```

Si vemos paso a paso el ejemplo:

- Se define $pi a nivel global en el programa.
- Se define la función valor_pi() que por ahora no se utiliza.
- Se lanza la ejecución de la función.
- Dentro de la función, la variable $pi se define como global, es decir, se utiliza la variable creada al principio del código.
- La función evalúa si existe la variable $pi y da como resultado *true*.
- El resultado por pantalla es 3.14.

## Variables estáticas

Por defecto, las funciones que creamos en PHP 5 no retienen en memoria el valor de las variables que se utilizan. Cada llamada a una función implica la nueva creación de las variables locales con su valor inicial. La declaración static añadida a una variable causa que la función retenga en memoria el valor de esa variable en cada llamada. Como puede ver en el ejemplo, las 100 llamadas a la función contador() dan siempre el mismo resultado, porque la variable $contador se crea con cada llamada.

```
<?php
function contador() {
   $contador = 0;
   $contador = $contador +1;
   return $contador;
}
for ($x = 1; $X <= 100; $x++) {
   echo contador() .", ";
}
?>
```

Sin embargo, podemos hacer que la función recuerde los valores, añadiendo la palabra static a la definición de la variable:

```
<?php
function contador() {
   static $contador = 0;
   $contador = $contador + 1;
   return $contador;
}
for ($x = 1; $x <= 100; $x++) {
```

```
    echo contador() ." ,  ";
}
?>
```

## íncludeO y requireO

A medida que su proyecto se hace más complejo, comenzará a crear funciones muy útiles para conectar a bases de datos, crear imágenes al vuelo, conectar por FTP, etcétera y, seguramente, quiera emplear estas funciones en otros proyectos distintos para ahorrar tiempo. El camino para hacer esto es muy sencillo. Sólo tiene que guardar las funciones necesarias en un fichero con extensión php, htm, css, inc, etcétera y utilizar las directivas include () orequiref) desde la Web que vaya a utilizarlas.

En PHP 5 no existen apenas diferencia entre ambas funciones, salvo que la función include () es más tolerante a fallos que require () y permite continuar la ejecución del programa aunque se haya encontrado un fallo.

La conclusión es que debe utilizarse require () para aplicaciones que necesiten obligatoriamente algún archivo crítico y no pueda continuarse la ejecución sin él.

Si en el desarrollo de nuestro proyecto nos encontramos con que vamos a utilizar varios archivos que hacen llamadas unos a otros podría pasar que hagamos varias inclusiones del mismo archivo en distintos puntos, provocando algún fallo difícil de detectar. Las funciones require_once () e include_once () solucionan éste problema, ya que si se intenta incluir un fichero varias veces con dichas funciones, PHP 5 simplemente ignorará las llamadas.

El formato es el siguiente:

```
require "base_datos.php";
include "contador_vistas.php";
require_once"test,php",-
include_once "visitas.php";
```

## Recursividad

El paradigma de la programación nos lleva a los desarrolladores a inventar fórmulas que minimicen el esfuerzo en nuestros programas pecando,

algunas veces, de complejidad. Las funciones recursivas son aquellas capaces de llamarse a sí mismas. Como único requisito es que haya una forma de salir en la llamada recursiva. Si volvemos al ejemplo del número factorial nos encontramos en que se puede escribir de la siguiente forma:

```
<?php
function factorial($numero)
{
   if ($numero == 1) {
        return $numero;
   } else {
        return $numero * factorial($numero-l);
   }
}
echo "El factorial de 7 es ". factorial(7) ."<br>";
?>
```

Lo primero que llama la atención es que carece de un bucle for para ir disminuyendo los números implicados en el factorial. Lo que hace la función es multiplicar el número que se pasa por el factorial del número-1. Si el número que se pasa es igual a 1, la función deja de hacer llamadas recursivas y comienza a devolver los valores hasta llegar al resultado final.

# Funciones con número de argumentos variables

Es habitual que el número de parámetros que se le pase a una función dependa de la situación en la cual es llamada. Hay tres formas de hacer esto en PHP:

- Definiendo la función con argumentos por defecto. Este método permite hacer llamadas con menos parámetros sin que aparezca un error.
- Usando un *array* para pasar las variables.
- Usando las funciones de argumento variable func_num_args () , func_get_arg () y func_get_args () , ya utilizadas en PHP 4.

## Argumentos por defecto

Para definir parámetros de este tipo, se sustituyen las variables por expresiones de asignación. El formato es el siguiente:

```
function nombre_función($argumentol=valorl, $argumento2=valor2, ..)
```

De esta forma, al llamar a la función, podemos hacerlo con varios parámetros. Si alguno de los parámetros es obviado, la variable tendrá como valor, el valor por defecto de la definición.

```
<?php
function capitales($Pais,$Capital = "Madrid",$habitantes = "muchos") {
   return ("La capital de $Pais es $Capital y tiene $habitantes
habitantes.<br>");
}
echo capitales("España");
echo capitales("Portugal","Lisboa");
echo capitales("Francia","Paris","muchísimos");
?>
```

La salida por pantalla es la siguiente:

```
La capital de España es Madrid y tiene muchos habitantes.
La capital de Portugal es Lisboa y tiene muchos habitantes.
La capital de Francia es Paris y tiene muchísimos habitantes.
```

La función `capitales()` tiene un parámetro fijo y dos parámetros por defecto. El parámetro `$Pais` es obligatorio, pero los demás pueden quedarse en blanco, recibiendo el valor por defecto. La limitación de este método es que los argumentos tienen un orden y no podemos saltarnos ninguno, es decir, no podemos dar los valores `$Capital` y `$habitantes`, porque en medio hay otro parámetro.

## Argumentos mediante un array

Si no encontró alguna utilidad a la anterior forma de pasar valores variables, quizás necesite utilizar un *array* para pasarlos. El ejemplo siguiente usa esta estrategia y algunos trucos más, como el operador ternario y los *array* asociativos.

```
<?php
function capitales($datos) {
   $Pais = isset($datos['Pais']) ? $datos['Pais'] : "España";
   $Capital = isset($datos['Capital']) ? $datos['Capital'] : "Madrid";
   $habitantes = isset($datos['habitantes']) ? $datos
   ['habitantes'] : "muchos";
   return ("La capital de $Pais es $Capital y tiene $habitantes
   habitantes.<br>");
}
// Introducimos en el array los datos uno por uno para que sea
más fácil de entender
$datos['Pais'] = "España";
echo capitales($datos);
$datos['Pais'] = "Portugal";
```

```
$datos ['Capital'] = "Lisboa";
echo capitales($datos);
$datos ['Pais'] = "Francia";
$datos ['Capital'] = "Paris",
$datos ['habitantes'] = "muchísimos";
echo capitales($datos);
?>
```

Como puede ver, la función tiene un único argumento, que contiene todos los datos que nuestro programa necesita. Esta vez, los valores por defecto los introducimos mediante el operador ternario, que chequea si existe el dato referido al `Pais`, `Capital` o `habitantes` y, si no existe, se añaden los datos "España", "Madrid" y "muchos ". El ejemplo utiliza la función con 0, 2 y 3 parámetros. La ventaja de este método es que podrá utilizar los argumentos que quiera y en el orden que necesite, sin encasillarse en una forma especial de pasar los datos.

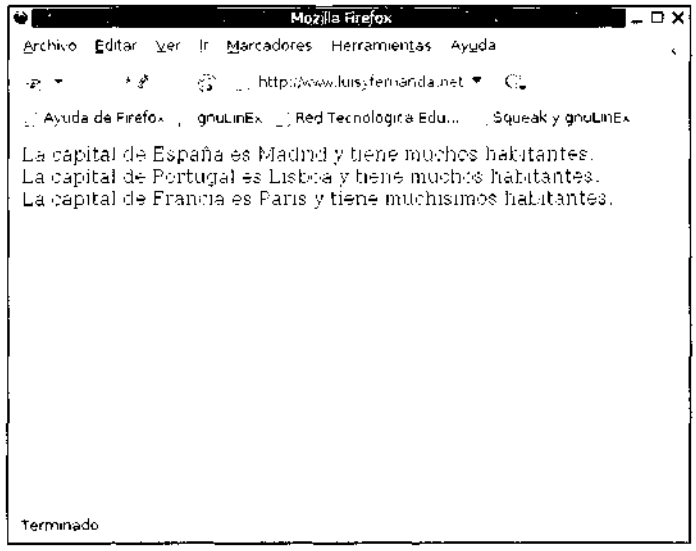



**Figura 5.4.** Paso de parámetros mediante array.

**Nota:**

*Aunque todavía no hemos visto el capítulo 7, que trata con detalle el manejo* dearrays, *el lector puede hacerse una idea de cómo funcionan en el ámbito de los parámetros defunciones. Si queda alguna duda de cómo utilizarlos, siempre puede echar un breve vistazo al capítulo 7 para aclarar esas lagunas.*

## Múltiples argumentos con func_num_args()

Desde la versión 4, PHP ofrece algunas funciones que pueden ser utilizadas para recuperar los argumentos.

Son muy similares al lenguaje C:

- **func_num_args():** Devuelve el número de argumentos que recibe la función desde la que es llamada.
- **func_get_arg():** Devuelve uno a uno los argumentos pasados de la siguiente forma: func_get_arg(0), func_get_arg(l), func_get_arg(5).
- **func_get_args():** Devuelve un *array* con todos los argumentos pasados a la función, con los índices del *array* empezando desde 0.

Cualquiera de estas funciones dará un error si son llamadas fuera del entorno de una función, y `func_get_arg()`, producirá un fallo si es llamada con un número más alto que los argumentos que se reciben.

Las funciones anteriores dan una ventaja a largo plazo, ya que si, durante el período de vida de una función necesita añadir algún argumento más, puede capturar sin necesidad de cambiar el código de las llamadas o el de definición de la función.

En el ejemplo siguiente puede comprobar cómo se utiliza este método:

```
<?php
function capitales() {
   $numero_argumentos = func_num_args();
   $Pais = $numero_argumentos > 0 ? func_get_arg(0) : "España";
   $Capital = $numero_argumentos > 1 ? func_get_arg(1) : "Madrid";
   $habitantes = $numero_argumentos > 2 ? func_get_arg(2) :
   "muchos";
   return ("Número de argumentos es: $numero_arguraentos. La capital
   de $Pais es $Capital y tiene $habitantes habitantes.<br>");
}
echo capitales();
echo capitales("Portugal", "Lisboa");
echo capitales("Francia", "Paris", "muchísimos");
?>
```

Esta forma de utilizar los argumentos sigue teniendo una limitación. Los argumentos deben ser pasados en un lugar determinado, sino la función no hará bien su trabajo.

Utilizar los parámetros como array, es el método más flexible y el más utilizado en los programas PHP.

Aún así, es muy útil cuando no sepa cuántos datos necesita manejar una función. Podemos utilizarlo para funciones que sumen todos los parame-

tros que introduzca o concatenen todas las palabras, como el ejemplo siguiente:

```
<?php
function concatenar() {
   $resultado = "";
   $numero_argumentos = func_num_args();
   $array_parametros = func_get_args();
   for ($x = 0; $x <= $numero_argumentos; $x++){
       $resultado = $resultado . $array_parametros[$x];
   }
   return $resultado."<br>";
}
echo concatenar("Hola ","Mundo");
echo concatenar("Esto ","es ","una ","prueba ","de","la
potencia"," de este"," método");
?>
```

## Llamadas por valor

Por defecto, las llamadas a funciones se hacen por valor. Esto quiere decir que las variables que se utilizan como parámetros se copian al entorno de la función.

Todos los cambios de valor que sufra la variable en este entorno no afectarán al programa fuera de la función.

```
<?php
function elevado($nuraero,$índice) {
   $resultado = $numero;
   for ($x = $indice; $x > 0; $x--) {
       $resultado = $resultado * $nuraero;
   }
   $numero = $resultado;
   return $numero;
}
$numero = 2;
$indice = 5;
echo $numero." elevado a ".$indice." es igual a
".elevado($numero,$indice)."<br>";
echo $numero." elevado a ".$indice." es igual a
".elevado($numero,$indice)."<br>";
?>
```

El resultado por pantalla es:

```
2 elevado a 5 es igual a 64
2 elevado a 5 es igual a 64
```

El ejemplo anterior calcula el resultado de $numero elevado a $Índice. La función elevado () crea un bucle para multiplicar $numero por las veces que indica el $índice y lo va almacenando en la variable temporal $resultado.

Cuando el bucle termina, este resultado se vuelve a pasar a la variable $numero, que es devuelta como salida de la función. Cuando volvemos a ejecutar la función con los mismos parámetros, vemos que la variable $numero no ha cambiado en la ejecución anterior, porque, en realidad, lo que se pasa es una copia del valor y todas las modificaciones se hacen sobre la copia y no sobre la variable original.

## Llamadas por referencia

PHP 5 ofrece actualmente dos caminos diferentes para cambiar sus argumentos: en la definición de la función y en la llamada a la misma.

Las variables pasadas por referencia pueden ser modificadas durante el proceso de una llamada, porque lo que se pasa no es una copia, sino la variable en sí misma.

Para pasar variables por referencia hay que utilizar el operador (&) delante de la variable.

Lo mejor es verlo en un ejemplo:

```
<?php
function elevado(&$numero,&$indice)  {
   $resultado = $numero;
   for ($x = $índíce; $x > 0; $x--)  {
       $resultado = $resultado * $numero;
   }
   $numero = $resultado;
   return $numero;
}
$numero = 2 ;
$índice = 5;
echo $numero." elevado a ".$indice." es igual a
" .elevado($numero,$indice) ."<br>";
echo $numero." elevado a ".$indice." es igual a
".elevado($numero,$indice) ."<br>";
?>
```

El resultado por pantalla es ahora:

```
2 elevado a 5 es igual a 64
64 elevado a 5 es igual a 68719476736
```

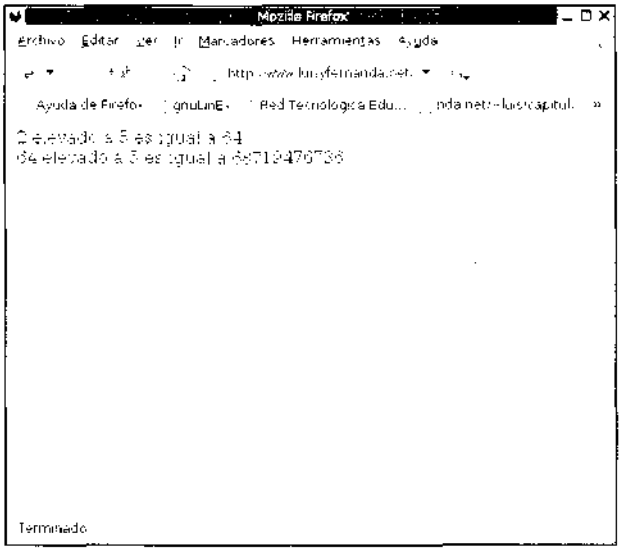



**Figura 5.5.** Cálculo de potencias.

El funcionamiento es exactamente igual, pero el resultado es distinto. El operador (&) hace que la variable $numero pueda ser modificada dentro de la función y guarde este número fuera de ella.

También puede forzar a una función a tomar argumentos por referencia, aunque esta capacidad de PHP puede desaparecer en futuras versiones del lenguaje. Las llamadas pueden hacerse de la siguiente forma:

```
$numero = 2;
$indice = 5;
echo $numero." elevado a ".$indice." es igual a
".elevado(&$numero,&$indice)."<br>";
echo $numero." elevado a ".$indice." es igual a
".elevado(&$numero,&$indice)."<br>";
```

El resultado es exactamente igual que al poner la referencia en la definición de la función.

## Referencia a variables

Las referencias pueden usarse también fuera de las funciones, en el ámbito de las variables. El ejemplo siguiente muestra cómo trabaja el operador (&) con las variables:

```
<?php
$nombre = "Luis Miguel";
```

```
$apellídos = "Cabezas Granado";
//$nombre auxiliar es simplemente una copia de $nombre
$nombre_auxiliar = $nombre,-
//$nombre_referencia es una referencia a la variable
$nombre_referencia = &$nombre;
echo $nombre." ".$apellidos."<br>";
$nombre_auxiliar = "Alberto";
echo $nombre." ".$apellidos."<br>";
$nombre_referencia = "Felipe";
echo $nombre." ".$apellidos."<br>";
?>
```

$nombre__auxiliar es una copia de la variable $nombre *y* por mucho que la cambiemos, $nombre permanecerá con el mismo valor. En cambio, la variable $nombre_referencia es un *alias* de $nombre, por lo tanto todos los cambios que se hagan en cualquiera de las dos variables les afectará por igual.

## Funciones variables

Uno de los trucos que se pueden hacer con PHP es utilizar variables para almacenar el nombre de funciones. Si una variable almacena el nombre de una función, simplemente tendrá que añadir unos paréntesis al final de la variable para hacer la llamada correctamente.

```
<?php
function saludo_maniana() {
   return ("Buenos Días");
}
function saludo_tarde() {
   return ("Buenas Tardes");
}
function saludo_noche() {
   return ("Buenas Noches");
}
$horario = "tarde";
$funcion_variable = "saludo_".$horario;
echo $funcion_variable();
?>
```

En el ejemplo anterior tenemos tres funciones que nos "saludan" dependiendo del horario en el que ejecutemos el programa. Este valor se recoge en la variable $horarío. Sabemos que todas las funciones comienzan con la palabra "saludo_". Si a la variable $funcion_varíable le concatenamos el valor de $horario, tenemos como resultado el nombre de la fun-

ción que nos dará el saludo. Para que esto funcione, añadimos dos símbolos de paréntesis al final de la expresión.

## Resumen

Si el capítulo anterior daba rienda suelta a su imaginación para comenzar con el desarrollo de pequeños *scripts*, después de leer este capítulo estará en situación de realizar un gran proyecto. Ahora que conoce el secreto para almacenar en distintos archivos funciones y partes de un programa escrito en PHP, podrá crear aplicaciones más extensas y complejas.

Ahora bien, debe tener en cuenta que hay que tener cuidado con el alcance de las variables y los pasos por valor y referencia, aspecto muy relevante en el capítulo referido a los objetos.


Capítulo 5

# Cadenas de caracteres y expresiones regulares

**En este capítulo aprenderá a:**

- Diferenciar entre las comillas simples y las comillas dobles.
- Trabajar con cadenas de múltiples líneas,
- Concatenar varias cadenas.
- Generar nuevas cadenas de caracteres a partir de una original.
- Eliminar los espacios en blanco de las cadenas.
- Convertir a mayúsculas o minúsculas.
- Definir patrones de verificación de cadenas.

# Introducción

Si pensamos en los tipos de datos que circulan por la red, llegaremos a la conclusión de que una gran porción la ocupan las imágenes, animaciones en Macromedia Flash, videos o subprogramas escritos en Java. La otra gran porción, mayoritaria, son los textos o las cadenas de caracteres.

Las cadenas de caracteres o *string* son secuencias de caracteres que pueden ser tratadas como una unidad, asignadas a variables, pasadas como parámetros a funciones o enviadas como salida al navegador. *Un string* se diferencia de otro tipo de dato en PHP 5, porque va encerrado entre comillas dobles ( ") o simples ( ').

```
"Cadena entre comillas dobles"
'Cadena entre comillas simples'
```

PHP interpreta de distinta forma las cadenas que van entre comillas dobles y las que van entre comillas simples. *Losstrings* entre comillas dobles pueden sustituir ciertos símbolos por acciones, como la inclusión del valor de una variable. Las comillas simples, simplemente muestran todo el contenido, sin atender a caracteres especiales. Podemos ver un ejemplo donde sucede esto:

```
<?php
$variable = "Domingo";
$frase_l = "Hoy es $variable, el cielo está gris";
$frase_2 = 'Hoy es $variable, el cielo está gris';
echo $frase_l;
echo $frase_2;
?>
```

El resultado en el navegador es:

```
Hoy es Domingo, el cielo está gris
Hoy es $variable, el cielo está gris
```

Como se puede observar, la cadena $frase_l es capaz de sustituir la variable por su valor, por el simple hecho de estar entre comillas dobles.

# Propiedades de las cadenas

## índices de string

Si pensamos en las cadenas como una sucesión de caracteres en un orden determinado, podemos llegar a desear acceder libremente a parte de los

caracteres. Esto es posible gracias a los símbolos de llave ({ } ) y un índice numérico que se corresponderá con la posición del carácter que buscamos. El ejemplo muestra cómo crear una función que duplica las letras de una cadena aprovechando esta forma de acceder a los caracteres:

```
<?php
function duplicar_caracteres($cadena)  {
   $tamanio = strlen ($cadena) ;
   $cadena_auxiliar =  "";
   for   ($x = 0;$x < $tamanio;  $x++)  {
        $cadena_auxiliar = $cadena_auxiliar  .  $cadena{$x}
$cadena{$x};
   }
   return  $cadena_auxiliar;
}
$cadena = "Duplicar las letras";
echo  duplicar_caracteres($cadena);
?>
```

La salida por pantalla, como puede adivinar, es:


**Figura 6.1.** Caracteres duplicados.

**Advertencia:**

*En versiones anteriores de PHP se podía utilizar el símbolo de corchete ( [ ] ) para acceder a los caracteres. Esta forma de actuar está en desuso y se recomienda utilizar la expuesta en este libro.*

## Operadores

En este punto, aprovecharemos para hacer un breve repaso de los operadores de *string* vistos en el capítulo 3. En otros lenguajes, como Java, se utiliza el operador suma (+) para unir dos cadenas. En PHP este mismo resultado se obtiene con el operador punto (. ). Así, podemos concatenar varias cadenas de la forma siguiente:

```
<?php
$cadenal = "Hola";
$cadena2 = "Mundo";
$cadena3 = "¡Qué típico!";
$supercadena = $cadenal . " " . $cadena2 . " " . $cadena3;
echo $supercadena;
?>
```

Es fácil intuir que el operador puede concatenar caracteres y variables de tipo *string* de forma conjunta. Es posible que desee ahora añadir texto a una cadena ya existente; esto se puede hacer de dos formas muy similares. La primera es asignando a la variable su valor más el valor a añadir, de la siguiente forma:

```
<?php
$cadenal = "Hola";
$cadena2 = "Mundo";
$cadenal = $cadenal . $cadena2;
echo $cadenal;
?>
```

O utilizando el operador de concatenación y asignación ( .= ), como en el ejemplo:

```
<?php
$cadenal = "Hola";
'$cadena2 = "Mundo";
$cadenal .=  $cadena2;
echo $cadenal;
?>
```

## Sintaxis para múltiples líneas

Existe en PHP una forma más de introducir cadenas, aparte de las comillas, muy recomendable para largos textos o un considerable conjunto de etiquetas HTML. La sintaxis es muy sencilla y comienza con el operador (<<<) seguido de una etiqueta que indica el principio del texto. Después de esto podemos escribir un conjunto de caracteres muy numerosos y, para

finalizar, la etiqueta de fin. El ejemplo muestra cómo almacenar un formulario en una variable:

```
<?php
$formulario = <<<INICIO
<form>
<input type="text" name="Nombre" value="Luís Miguel Cabezas">
<br>
<input type="submit" name="submit" value="Enviar" >
</form>
INICIO;
echo $formulario;
?>
```

Siempre debe comenzar por el operador (<<<), seguido de un identificador, por ejemplo INICIO. Después se incluye el texto, en este caso un formulario y, para finalizar, la misma etiqueta de inicio. Esta forma de almacenar datos permite utilizar indistintamente comillas simples o dobles dentro de una variable. La imagen 6.2 muestra el resultado del *script*.


**Figura 6.2.** Formulario almacenado en una variable.

## Funciones de string

Si conoce el lenguaje C, podrá comprobar que las funciones de manejo de caracteres son muy parecidas en nombre y uso. Los desarrolladores de PHP

siempre han querido que exista esta similitud para facilitar el aprendizaje del lenguaje.

La mayoría de las funciones existentes en PHP 5 tienen una homologa en C.

## Tamaño de la cadena

La función `strlen()` devuelve como resultado el tamaño en caracteres de la cadena que pasemos de parámetro. Este valor puede asignarse a una variable, como indica el ejemplo:

```
<?php
$cadena = "Esta cadena tiene muchas letras";
$numero_letras = strlen($cadena);
echo "cadena tiene $numero_letras caracteres";
?>
```

Simplemente imprime:

```
cadena tiene 31 caracteres
```

## Posición de los caracteres

Para crear buenos motores de búsqueda en nuestros sitios Web, necesitaremos potentes herramientas que sean capaces de buscar caracteres, cadenas o patrones coincidentes dentro de textos extensos.

La función `strpos()` encuentra en un *string* la posición de un carácter determinado.

```
<?php
$cadena = "Esta cadena tiene muchas letras";
echo "La primera ocurrencia de 'a' es " . strpos($cadena, "a")
"<br>";
echo "La primera ocurrencia de 'm' es " . strpos($cadena, "m")
"<br>";
?>
La salida en el navegador no es otra que:
La primera ocurrencia de 'a' es 3
La primera ocurrencia de 'm' es 18
```

Si `strpos()` se utiliza para averiguar la posición de la primera "L" mayúscula de `$cadena` vemos que nos devuelve un 0; esto es así porque el índice de caracteres comienza en 0, como pasa en muchos lenguajes de programación. Si no se encuentra ninguna ocurrencia del carácter buscado el resultado será/a/se.

La función strpos () también puede utilizarse para buscar varios caracteres seguidos en una misma cadena:

```
<?php
$cadena = "Esta cadena tiene muchas letras";
echo "La primera ocurrencia de 'tiene' es " .
strpos($cadena,"tiene") . "<br>";
?>
```

Si lo que nos interesa es encontrar caracteres buscando desde el final de la cadena la función que tenemos que utilizar es strrpos () (la letra r que se añade viene del inglés *reverse).* El ejemplo anterior nos sirve para comprobar el funcionamiento de la función:

```
<?php
$cadena = "Esta cadena tiene muchas letras";
echo "La primera ocurrencia de 'a' es " . strpos($cadena, "a" )
"<br>";
echo "La primera ocurrencia desde atrás de 'a' es " .
strrpos($cadena, "a") . "<br>";
?>
```

siendo el resultado:

```
La primera ocurrencia de 'a' es 3
La primera ocurrencia desde atrás de 'a' es 29
```

## Comparación

El operador (==) nos puede servir para evaluar si dos cadenas de caracteres son iguales dentro de una estructura de control. Además, tenemos la función strcmp (), que compara *bit* a *bit* dos cadenas de caracteres. Atendiendo al valor de salida de la función, podemos obtener los siguientes resultados:

- Si el valor que se obtiene es 0, las dos cadenas son exactamente igual.
- Si obtenemos un valor negativo, el primer *string* es más pequeño que el segundo.
- Si obtenemos un valor positivo, el primer *string* es más grande que el segundo.

```
<?php
$cadenal = "Prueba";
$cadena2 = "Prueba";
if (strcmp($cadenal,$cadena2) == 0) {
   echo "Las dos cadenas son iguales";
} elseif (strcmp($cadenal,$cadena2) < 0) {
   echo "La cadenal es menor que la cadena2";
```

```
} else {
  echo "La cadenal es mayor que la cadena2";
}
?>
```

La comparación anterior sólo da como resultado "dos cadenas iguales", si las dos cadenas son exactamente iguales, haciendo distinción entre mayúsculas y minúsculas.

Para evitar este problema podemos utilizar la función strcasecmp(), que no es sensible a mayúsculas. Así strcasecmp("HoLa", "hOlA") da como resultado 0, es decir, las dos cadenas son iguales.

## Búsqueda de caracteres

Siguiendo con las búsquedas de texto, la función strstr() recibe como parámetros una cadena de caracteres donde buscar y otra con un conjunto de caracteres que queremos encontrar. Si se encuentra el patrón de búsqueda, el resultado será una cadena de caracteres que comenzará justo en el patrón, hasta el final del *string* primero. Si no se encuentra el patrón el resultado *seráfalse.*

```
< ?php
$cadena = "Esta cadena tiene muchas letras";
echo "La primera ocurrencia de 'cadena' es: " .
strstr($cadena,"cadena") . "<br>";
?>
```

Como cabe esperar, el resultado es:

```
La primera ocurrencia de 'cadena' es: cadena tiene muchas
letras
```

## Selección de subcadenas

La función substr() permite seleccionar un conjunto de caracteres de una cadena, quedando intacto *el string* original. Puede tomar varios parámetros:

- substr(cadena, índice): Cadena es el conjunto de caracteres que queremos cortar e índice la posición a partir de la cual se cortará la cadena hasta el final.
- substr(cadena, índice, numero) : Cadena es el conjunto de caracteres que queremos cortar e índice la posición a partir de la cual se cortará la cadena tantas posiciones como indique el número.

Por ejemplo:

```
<?php
$cadena = "Esta cadena tiene muchas letras";
//Devuelve la cadena completa
echo substr($cadena, 0);
echo "<br>";
//Desde el carácter 12 hasta el final
echo substr($cadena, 12);
echo "<br>";
//Devuelve 6 caracteres desde el carácter 18
echo substr($cadena, 18,6);
echo "<br>";
//Devuelve los 6 últimos caracteres
echo substr($cadena, -6);
echo "<br>";
//Desde la posición 26, contando desde atrás, 6 caracteres
echo substr($cadena, -26,6);
echo "<br>";
//Empezando en el carácter 4 y terminando en el 7 desde atrás
echo substr($cadena, 4,-7);
?>
```

El resultado lo podemos ver más abajo. Recuerde que la variable `$cadena` queda intacta.

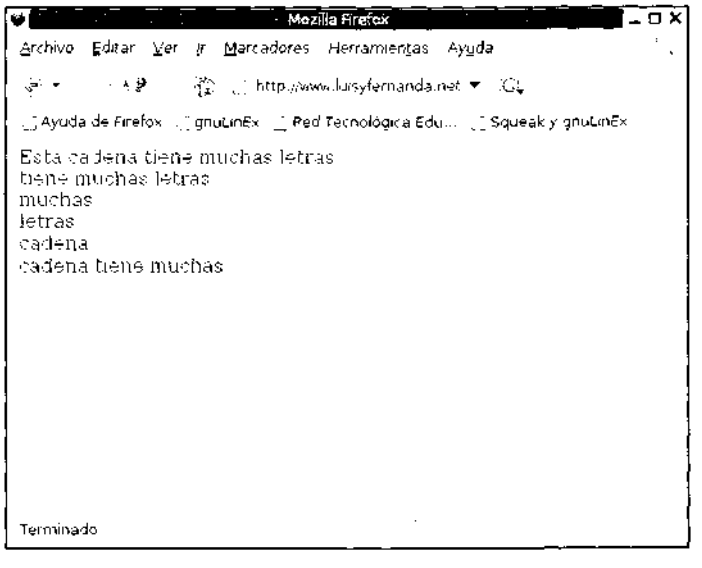



**Figura 6.3.** Selección de subcadenas.

Como se ve en el ejemplo, también pueden emplearse números negativos.

Dependiendo de la posición del número negativo, `substr()` actuará de una forma u otra.

**Tabla 6.1.** Respuestas de la función `substr()`.

| Ejemplo | Resultado |
|---|---|
| substr("Hola",2) | Devuelve la cadena completa empezando por el carácter 2. |
| substr("Hola",-2) | Devuelve la cadena completa empezando por el carácter 2, esta vez empezando a contar desde atrás. |
| substr("Hola",1,3) | El resultado son 3 caracteres empezando desde el carácter 1. |
| substr("Hola",-3,2) | Dos.caracteres, empezando desde el carácter 3 contando desde la última letra. |
| substr("Hola",2,-1) | Devuelve un conjunto de caracteres que están desde la posición 2 a la posición 1 comenzando desde atrás. |
| substr("Hola",-2,-3) | No tiene aplicación posible. |

## Funciones de limpieza de cadenas

Sirven para limpiar espacios en blanco al principio de la cadena, al final o en cualquier parte desde el principio al final y son respectivamente `ltrim()`, `chop()` y `trim()`. Su aplicación es tan sencilla que bastará un ejemplo para entender su funcionamiento:

```
<?php
$salto_linea = "\n";
$cadena = " cadena con   varios   espacios en blanco   ";
echo $salto_linea . $cadena . " tamaño: " . strlen($cadena);
echo "<br>";
echo $salto_linea . ltrim($cadena) . " tamaño: " .
strlen(ltrim($cadena));
echo "<br>";
echo $salto_linea . chop($cadena) . " tamaño: " .
strlen(chop($cadena));
echo "<br>";
echo $salto_linea . trim($cadena) . " tamaño: " .
strlen(trim($cadena));
?>
```

Cada línea realiza una de las operaciones vistas, e imprime el resultado de la cadena en el navegador junto con el tamaño de la cadena.

La figura 6.4 muestra el resultado por pantalla.

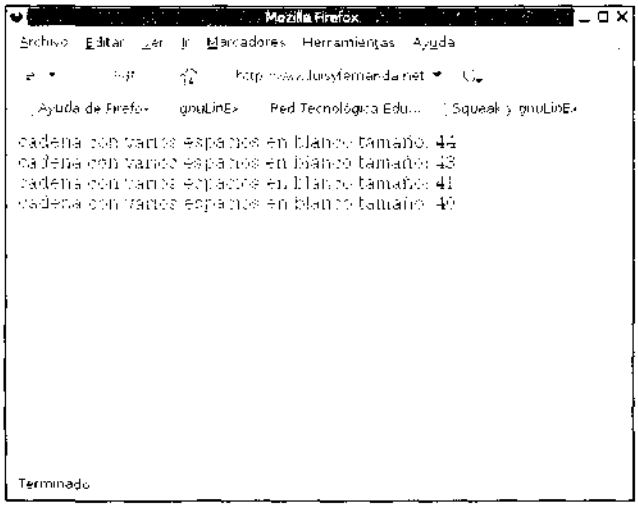



**Figura 6.4.** Cadenas después de eliminar los espacios en blanco.

**Nota:**

*Si se fija en el código y en el resultado, a primera vista parece que el resultado no es el esperado. En realidad sí se han aplicado correctamente los cambios, pero sobre el código HTML. Lo que ocurre es que el navegador elimina el exceso de espacios en blanco e imprime todas las cadenas como si no hubiera pasado nada. La figura 6.5 muestra el código HTML, donde se pueden apreciar correctamente los cambios.*

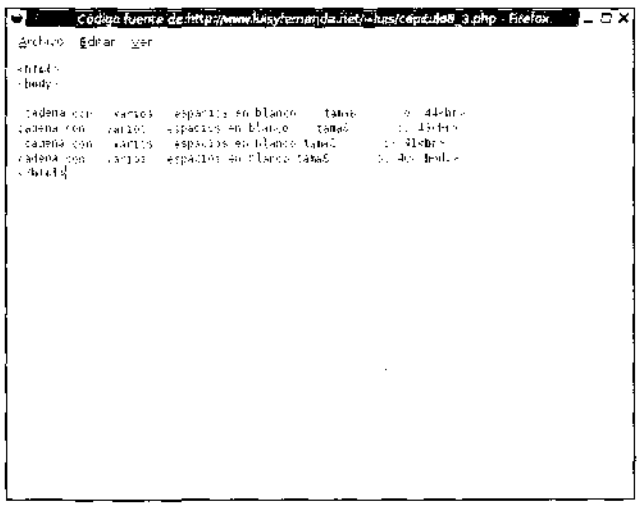



**Figura 6.5.** Código HTML después de la eliminación de espacios.

## Sustitución de cadenas

La función str_replace () toma como parámetros un conjunto de caracteres a buscar, un grupo que debe sustituirse por el anterior y una cadena de caracteres sobre la que actuar.

```
<?php
$cadena = "Esta cadena tiene muchas letras";
$cadena = str_replace("Esta","Este",$cadena);
echo  str_replace("cadena","conjunto",$cadena);
?>
```

Si en la cadena existiera más de una instancia del conjunto buscado, todas las ocurrencias quedarían cambiadas.

```
<?php
{cadena = "Esta cadena tiene muchas letras y cadenas";
$cadena = str_replacé("Esta","Este",$cadena);
echo  str^replace("cadena","conjunto",$cadena);
?>
```

El resultado es:

```
Este conjunto tiene muchas letras y conjuntos
```

**Truco:**

*Puede probar esta función para cambiar todos ¡os caracteres de un tipo de una cadena de ¡a siguiente forma:* str_replace ("h", "j", "hola hola").

## Funciones de mayúscula y minúscula

Las funciones strtolower () y strtoupper () devuelven la cadena que se pasa como argumento, completamente en minúscula o mayúscula respectivamente.

```
<?php
$cadena = "Hay palabras en MAYÚSCULAS y en minúsculas<br>" ;
echo strtolower($cadena) ;
echo strtoupper($cadena);
?>
```

El resultado es:

```
hay palabras en mayúsculas y en minúsculas
HAY PALABRAS EN MAYÚSCULAS Y EN MINÚSCULAS
```

Si lo que realmente nos interesa es que la primera letra de un texto aparezca como letra Capital o, que letras iniciales de cada palabra en una fra-

se aparezcan en mayúscula, tenemos las funciones `ucfirst()` y `ucwords()`.

```
<?php
$cadena = "había una vez...<br>";
echo ucfirst($cadena);
$cadena2 = "linux user group";
echo "LUG significa " . ucwords($cadena2);
?>
```

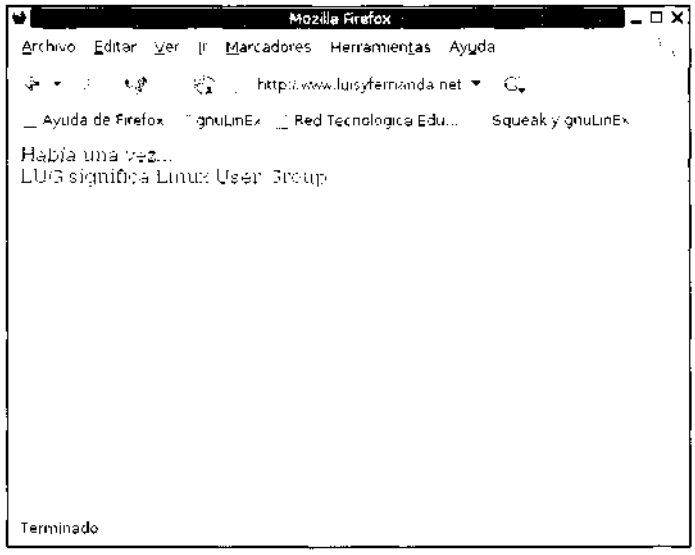



**Figura 6.6.** Control de mayúsculas y minúsculas.

## Expresiones regulares

Las funciones vistas anteriormente basan su potencia en la búsqueda o sustitución de los caracteres de una cadena, pero son poco útiles cuando tratamos de comprobar si un conjunto de caracteres cumple un formato determinado. Si solicitamos mediante un formulario un correo electrónico, necesitamos saber, a priori, si ese correo por lo menos está bien construido:

luis@nccextremadura.org
luis@zend.com

La idea es poder descartar de alguna forma los correos escritos de esta manera:

luis (Snccextremadura.org //Espacio en medio

```
LUIS@zend.com //En mayúsculas
luis@@aupex.org //Dos @
```

Las expresiones regulares son patrones de búsqueda dentro de cadenas. Estos patrones se construyen mediante caracteres especiales que cumplen unas reglas determinadas.

**Tabla 6.2.** Reglas de expresiones regulares.

| Ejemplo | Regla |
|---|---|
| aabb | Los caracteres no especiales, como un grupo de letras, se escriben como de costumbre. |
| ^ | Indica que hay que buscar el patrón desde el principio de la cadena. |
| $ | Este simbolo obliga al patrón a cumplirse hasta el final de la cadena. |
| . | Simboliza cualquier carácter. |
| * | Indica que puede haber O o más instancias de una expresión. |
| + | Indica que puede haber 1 o más instancias de una expresión. |
| [ab] | Indica que se puede encontrar el carácter a o el b y la expresión sería correcta. Lo podemos acompañar de [ab] * para evaluar si hay varias letras seguidas del tipo a o b. |
| [a-z] | Indica que se puede elegir entre un rango de caracteres que va desde la a hasta la z. |
| \ | Si queremos utilizar cualquiera de los anteriores caracteres como parte del patrón tendremos que utilizar éste símbolo para indicarle a PHP que es un literal. |

Como ejemplo vamos a intentar hacer un patrón que verifique si un correo electrónico está correctamente construido o no. Sería fácil pensar en una expresión parecida a la siguiente:

```
[a-z]+ @[a-z]+\.org
```

La primera parte, [a-z] +, nos dice que aceptará una letra, o conjunto de letras, sin espacios y en minúsculas, que pueden corresponderse con el nombre de usuario de la cuenta de correo. Después aceptará un símbolo @, seguido de otro conjunto de caracteres, correspondientes al servidor de correo. La última parte de la expresión la componen los símbolos \ ., que indican que tiene que aparecer un punto que separe la descripción del servidor del dominio y la cadena org, que obliga a que todos los correos sean de dominio no gubernamental.

La expresión anterior tiene algunas limitaciones, que veremos como solventar. La primera es que se puede aplicar a cualquier texto, con independencia del tamaño. Si en alguna parte de ese texto aparece un conjunto de caracteres que cumpla con la expresión, la validación sería *true.* Esto no es bueno cuando queremos chequear si un correo es introducido correctamente. La solución es aplicar las reglas de inicio y fin que obligan a que el comienzo de la cadena y el final sean parte de la expresión.

```
[a-z]+@[a-2]+\.org$
```

También podemos encontrarnos con que existen correos introducidos correctamente, pero el dominio es otro distinto a `org`. Esta vez la solución es introducir el operador O lógico `|`, que permite elegir entre varias opciones:

```
^[a-z]+@ [a-z]+\ . (org|com|net)$
```

Por último, un problema menor, aunque habitual en el ámbito de los correos electrónicos, es la utilización de signos de puntuación para separar nombres y apellidos y servidores de servidores virtuales. Por ejemplo, el correo siguiente es correcto, pero no sería evaluado por la expresión anterior:

```
luís.cabezas@ncc.aupex.org
```

Una posible solución es:

```
*[a-z|\.]+@[a-z|\.]+\.(org|com(net)$
```

## Comprobar expresiones regulares

La función `ereg ()` es capaz de comprobar si una cadena se corresponde con el patrón que se pasa como parámetro. Toma dos argumentos obligatorios, una expresión regular y una cadena de caracteres donde buscar el patrón. Seguidamente puede ver un ejemplo completo de utilización de expresiones regulares y la función:

```
<?php
$correo = "luis.cabezas@ncc.aupex.org";
$correol = "LUIS@ncc.aupex.org";
$correo2 = "lu@is.cabezas@ncc.aupex.org";
if (ereg ("* [a-z|\.]+@[a-z|\.]+\. (org|com|net)$",$correo)) {
          echo "El correo $correo se acepta.<br>";
} else {
          echo "El correo $correo no cumple el patrón.<br>";
}
if (ereg("^[a-z|\.]+@[a-z|\.]+\. (org|comjnet)$",$correol) ) {
          echo "El correo $correol se acepta.<br>";
```

```
}else {
          echo "El correo $correol no cumple el patrón.<br>";
}
if (ereg("^[a-zj\.]+@[a-zj\.]+\.(org|com|net)$",$correo2)) {
          echo "El correo $correo2 se acepta.<br>";
}else {
          echo "El correo $correo2 no cumple el patrón.<br>";
}
?>
```

El resultado por pantalla es:

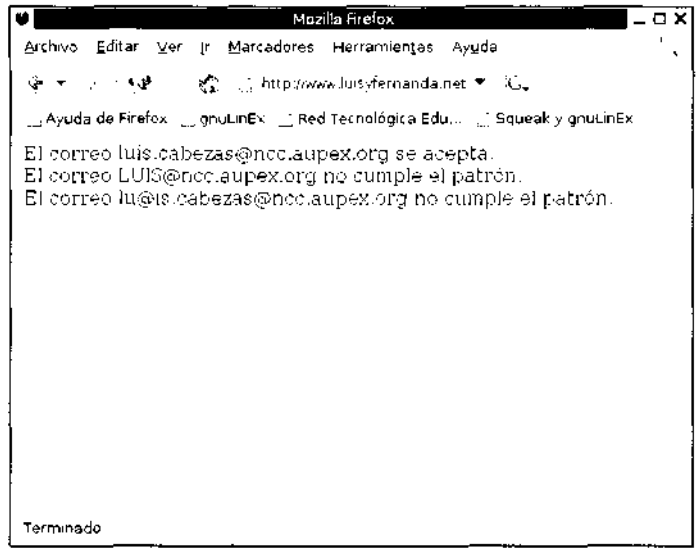



**Figura 6.7.** Verificación de un correo electrónico.

Puesto que hay muchos usuarios acostumbrados a escribir su correo electrónico completamente en mayúsculas, se hace necesario comprobar este contratiempo también. Lo más rápido es utilizar la función `eregi()`, que funciona exactamente igual que `ereg()`, pero no es sensible a mayúscula o minúscula. El ejemplo anterior lo podemos arreglar de la siguiente forma:

```
<?php
$correo = "LUIS@ncc.aupex.org";
if (ereg(""[a-z|\.]+@[a-z|\.]+\.(org|com|net)$",$correo,$pepe)) {
        echo "El correo $correo se acepta.<br>" ;
} else {
        echo "El correo $correo no cumple el patrón.<br>";
}
?>
```

**Nota:**

ereg() *y* eregi() *tienen un tercer argumento opcional del tipo* array. *Si ¡a expresión regular la dividimos en pequeñas partes entre paréntesis, cada ocurrencia encontrada en la cadena se va almacenando como un elemento del* array.

## Reemplazar patrones

ereg_replace() y eregi_replace() tiene tres parámetros. El primero es una expresión regular, el segundo un *string* por el que se cambiarán todas las ocurrencias del patrón y, por último, una cadena donde buscar. Como antes, la diferencia entre las dos es únicamente que la primera es sensible a mayúsculas. El siguiente ejemplo muestra el funcionamiento:

```
<?php
$texto = "El correo es de luis@ncc.aupex.org";
echo  ereg_replace("[a-z|\.]+@[a-z|\.]+\.(org|com|net)","Luis
Miguel",$texto);
?>
```

Así da como resultado el cambio del correo electrónico por el nombre de la persona asociada.

## Resumen

Desde hace unos años se ha puesto de moda una serie de productos donde prima el contenido frente al diseño. Los CMS *(Content Manager System)* son programas, la mayoría escritos en PHP, que permiten gestionar mucho contenido escrito por diferentes personas. El tratamiento de los textos se agudiza al máximo en este caso, siendo indispensable la utilización de expresiones regulares para verificar los datos nuevos que puedan incorporarse.

Además, suelen llevar un buscador capaz de encontrar patrones determinados dentro de las noticias expuestas.

Este capítulo le ha servido para conocer funciones para concatenar, recortar o verificar cadenas. Con esto bien aprendido ya puede comenzar a crear su propio CMS.


Capítulo 7

# Conjuntos de datos del tipo array

**En este capítulo aprenderá a:**

- Comprender qué es un array y cómo se define.
- Trabajar con funciones que definen listas de datos.
- Diferenciar entre los distintos métodos para recorrer los array.
- Ordenar los conjuntos de datos con diferentes métodos.
- Crear pilas y colas.

## Introducción

Un *array* es una colección de valores con un único nombre. Para acceder a los distintos valores de la variable se utiliza un índice numérico o alfanumérico.

Nos vamos a servir de la definición de un array en el lenguaje C, para ver la potencia de este tipo de datos en PHP. La definición es la siguiente:

```
int mi_array[100]; //Esto es C
```

Lo primero que debe llamarnos la atención es que la variable está predefinida como `int` (entero). En PHP los *arrays* no tienen que definirse de una forma concreta, sino que pueden tomar distintos tipos de valores: enteros, caracteres, objetos, etcétera. Lo siguiente es que, en C, se necesita saber de antemano el número de valores máximo que podrá tomar el *array*. Esto no es necesario en PHP, ya que podrá ir creando valores nuevos a medida que lo vaya necesitando. Lo último es que en C el índice para acceder a los 100 valores distintos debe ser numérico; en cambio, en PHP, el índice puede ser numérico o alfanumérico.

```
<?php
$mi_array[l] = 23;
$mi_array[2] = "Este valor es un string";
$mi_array["ejemplo"] = "Esto es un array asociativo";
?>
```

## Creación de arrays

Vamos a ver tres formas distintas de crear un *array* dentro de un script de PHP.

## Asignación directa

El camino más simple y, por otro lado lógico, es asignar valores cuando se necesiten. La primera vez que asignemos un valor, el *array* se creará en el entorno:

```
<?php
$mi_array[l] = 23; // Asignación directa
?>
```

De esta forma tenemos un valor asignado al índice 1 del *array*. Puede asignar cualquier índice en la creación de este tipo de dato, e incluso no asig-

nar ninguno, de forma que PHP se encarga de asociar un índice distinto para cada valor.

```
<?php
$mi_array[] = 23; // Empieza en el índice 0
$mi_array[] = 54; // índice 1
?>
```

## array()

Esta función crea un array con los valores que pase como datos de entrada. Los índices serán añadidos automáticamente empezando desde 0.

Si no asigna parámetros a `array()`, la función le devolverá un *array* vacío.

```
<?php
$mi_array = array(23,45,76,23,65);
?>
```

El método es similar a:

```
<?php
$mi_array[0] = 23;
$mi_array[l] = 45;
$mi_array[2] = 76;
$mi_array[3] = 23;
$mi_array[4] = 65;
?>
```

La función `array()` permite también añadir índices a los valores que se introducen.

Para ello se utiliza el operador => de esta forma:

```
<?php
$mi_array = array{0 => 23, 1 => 45, 2 => 76);
?>
```

También es posible añadir índices que no sean correlativos o índices alfanuméricos, incluso mezclar los dos tipos.

```
<?php
$mi_array = array("cero" => 23, "uno" => 45, 2 => 76);
?>
```

Para recuperar cualquier valor se utiliza el índice dentro de los corchetes:

```
<?php
$mi_array = array("cero" => 23, "uno" => 45, 2 => 76);
echo $mi_array["uno"]."<br>";
echo $mi_array[2];
?>
```

## Funciones que devuelven arrays

La última forma de obtener un *array* es utilizando alguna de las funciones que devuelven este tipo de datos.

Es muy frecuente que las funciones que manejan bases de datos devuelvan las ocurrencias dentro de un *array.*

Por ejemplo, la función `range ()` devuelve un *array* con valores numéricos, que van desde un número de inicio hasta un número final tal y como se muestra en el ejemplo:

```
<?php
$mi_array = range(120,130) ;
?>
```

Esta función crea un *array,* empezando desde el índice 0 y el valor 120, hasta el índice 10 y el valor 130.

## Arrays multidimensionales

Hasta aquí hemos visto ejemplos de *arrays* de una sola dimensión. PHP soporta el uso de arrays de varias dimensiones fácilmente, aunque son complejos de entender y de usar. El array siguiente es de 5 dimensiones:

```
<?php
$mi_array[l][1] [23][43][0] = "Array de 5 dimensiones";
?>
```

Cada índice encerrado entre corchetes corresponde a una dimensión distinta. En realidad, cada dimensión es un array que se introduce como valor dentro de la dimensión anterior.

Así, podrá crear un array multidimensional de la siguiente forma:

```
<?php
$colores = array("fuertes","suaves");
$colores ["fuertes"] = array (     "rojo" => "FF0000",
"verde" => "00FF00",
"azul" => "0000FF");
$colores["suaves"] = array( "rosa" => "FE9ABC",
"amarillo" => "FDF189"?
"malva" => "9A2F68");
echo $colores["fuertes"] ["rojo"];
?>
```

Para recuperar los valores se utilizan un índice del primer array (fuertes o suaves) y un índice de alguno de los dos arrays secundarios.

Otra forma de definir el *array* anterior es:

```
<?php

$colores = array( "fuertes" => array (  "rojo" => "FF0000",
                                        "verde" => "00FF00",
                                        "azul" => "0000FF"),
                  "suaves" =>  array (  "rosa" => "FE9ABC",
                                        "amarillo" => "FDF189",
                                        "malva" => "9A2F68"));
echo  $colores["fuertes"] ["rojo"];
?>
```

## Propiedades de arrays

Existen numerosas funciones que son capaces de averiguar datos de los *arrays,* tales como el tamaño, si un valor forma parte del conjunto o si un determinado índice está registrado.

### count()

Cuenta el número de elementos que contiene un *array.*

```
<?php
echo "elementos de 1 dimensión " . count ($colores) . "<br>";
echo "elementos de 2 dimensiones " . count
($colores["fuertes"]);
?>
```

El ejemplo anterior se basa en la definición del *array* de colores inicializado antes. Para contar el número de elementos debe ponerse también la dimensión. Si el *array* es de una sola dimensión no hace falta. La función `sizeof ()` actúa de la misma forma.

### in_array()

Busca dentro de un *array* un valor pasado como parámetro y, si lo encuentra devuelve el valor *true,* si no, devuelve *false.* Toma dos argumentos, el valor a buscar y el *array* dónde buscar.

```
<?php
$colores = array ("rojo","verde","amarillo","azul");
if (in_array("rojo",$colores)) {
   echo "Se ha encontrado el valor rojo";
```

```
} else {
   echo "No se ha encontrado";
}
?>
```

## Borrar ocurrencias

Para borrar un elemento, simplemente se utiliza la misma función que borra las variables definidas: unset () .

```
<?php
$colores = array ("rojo","verde","amarillo","azul","rosa") ;
echo "El número de elementos de colores es: " . count
($colores) . "<br>";
unset ($colores[2]);
echo "El número de elementos de colores es: " . count
($colores) . "<br>";
?>
```

Se puede observar que para borrar un elemento, hay que conocer el índice. El resultado del ejemplo por pantalla es:

```
El número de elementos de colores es: 5
El número de elementos de colores es: 0
```

**Advertencia:**

*Debe tener cuidado el lector con utilizar* unset() *con el nombre de un* array *sin índice, pues esto causará el borrado del conjunto de datos en su totalidad.* Unset($colores) *borra el* array *completo,* unset($colores[2] *borra sólo el índice 2 del conjunto.*

## Interactuar con arrays

Anteriormente, tuvo la oportunidad de conocer las estructuras de control existentes en PHP. La estructura foreach quedó en el aire para tratarlo a fondo en el presente capítulo. El uso es así:

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia") ;
foreach ($colores as $valor) {
   echo ("El valor es $valor<br>");
}
?>
```

La construcción anterior recorre el *array* desde el principio. En el ejemplo se puede ver que f o r e a c h toma el *array* a recorrer y sus valores los va almacenando en la variable $ v a l o r a medida que el bucle se ejecuta. Existe una segunda construcción que permite recuperar el índice y el valor. Veamos un ejemplo de esto:

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia");
foreach ($cíudades as $índice => $valor) {
   echo ("El índice $indice tiene el valor: $valor<br>") ;
}
?>
```

La salida en el navegador nos la muestra la figura 7.1.

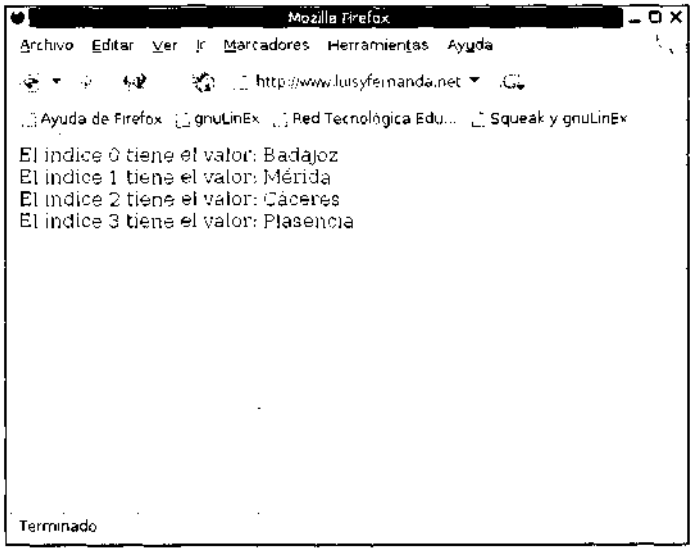



**Figura 7.1.** Recorrido de un array.

Es un método muy bueno para imprimir los valores de un *array* que tiene índices numéricos y alfanuméricos.

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia");
$ciudades["España"] = "Madrid";
$ciudades["Portugal"] = "Lisboa";
$ciudades["Francia"] = "Paris";
foreach ($ciudades as $índice => $valor) {
   echo ("El Índice $indice tiene el valor: $valor<br>") ;
}
?>
```

## Funciones para avanzar en un array

Cada vez que creamos un *array* dentro de un programa PHP, se crea un puntero que permite recorrer en su totalidad el conjunto de valores. Este puntero se inicializa al valor inicial del *array*. La función `current ()` devuelve el valor al que apunta el puntero. La función `next ()` hace avanzar el puntero una posición en el conjunto de datos.

Si el puntero se encuentra al final del conjunto `next ()` devuelve un valor *false*. El ejemplo siguiente muestra cómo utilizar estas funciones para recorrer un *array:*

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia") ;
$ciudades["España"] = "Madrid";
$ciudades["Portugal"] = "Lisboa";
$ciudades["Francia"] = "Paris";
do {
   $valor = current($ciudades);
   echo ("El valor es: $valor<br>");
}while (next($ciudades) ) ;
?>
```

Podemos crear una función que realice este trámite:

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia") ;
Sciudades["España"] = "Madrid";
$ciudades["Portugal"] = "Lisboa";
$ciudades["Francia"] = "Paris";

function recorre($ciudades) {
         do {
                 $valor = current($ciudades) ;
                 echo ("El valor es: $valor<br>") ;
          Jwhile (next($ciudades));
}

recorre($ciudades);
recorre($ciudades);
?>
```

La función `recorre ()` utiliza un *array* como parámetro y va utilizando las funciones de introspección para sacar los resultados. Puesto que los valores son pasados por valor, la función `recorre ()` crea una copia de los datos que recibe y genera el resultado de la figura 7.2.

De esta forma no hay manera de controlar el puntero, porque siempre que pase la variable `$ciudades` el puntero comenzará desde el principio.

La forma de arreglar esto es hacer las llamadas por referencia:

```
recorre (Sc$ciudades) ;
recorre (&$ciudades) ;
```

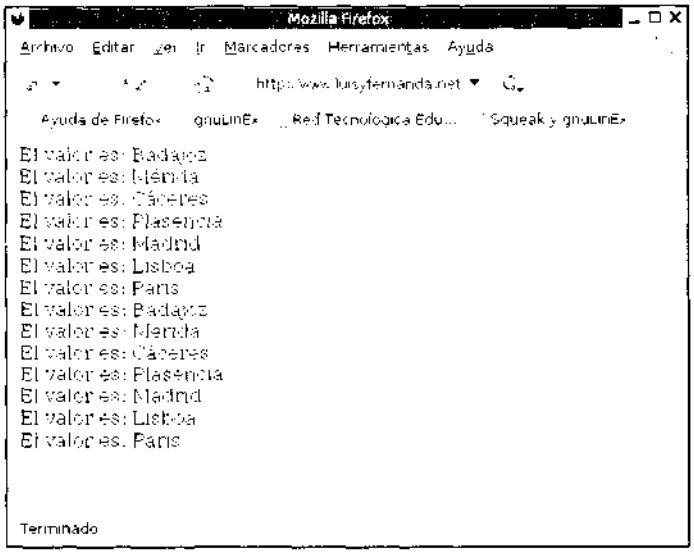



**Figura 7.2.** Recorrido de un array mediante la función recorre().

De esta forma tendrá un resultado distinto en la figura 7.3.

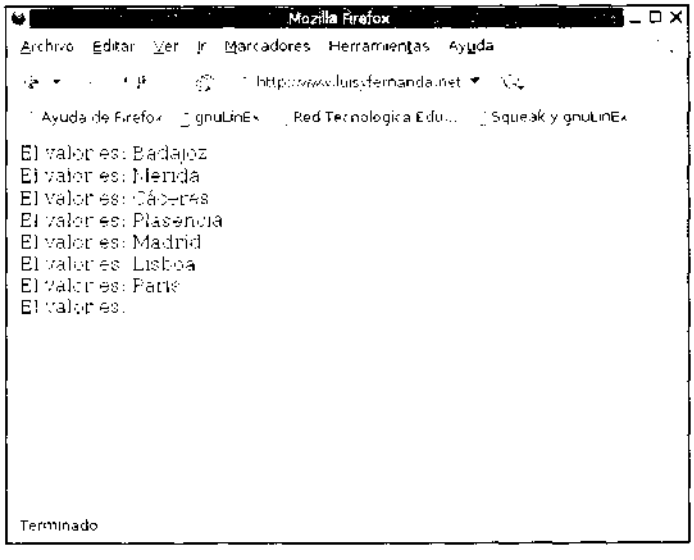



**Figura 7.3.** Recorrido de un array pasado por referencia.

Ahora, ya no es una copia de $ciudades lo que se pasa como parámetro, sino una referencia al valor real. Esto hace posible tener una memoria de la situación del puntero.

Una vez recorrido *elarray,* el puntero queda fijado al final del conjunto de datos. Para volver al principio se puede utilizar la función reset (), que envía al puntero al principio. La función puede quedar así:

```
function recorre($ciudades) {
         if (!current($ciudades)) {
                  reset($ciudades);
         }
         do {
                  $valor = current($ciudades) ;
                  echo ("El valor es: $valor<br>");
          Jwhile (next($ciudades));
}
```

Ahora lo que hacemos es comprobar si el puntero está al final del *array;* si esto es así se llama a la funciónreset 0 y el puntero se coloca al principio.

## Funciones para retroceder en un array

La función prev () retrocede una posición el puntero y la función end () se coloca al final de una lista de valores. Esto puede ser útil para mostrar el contenido en sentido inverso, como puede ver en la nueva función que hemos creado:

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia");
$ciudades["España"] = "Madrid";
$ciudades["Portugal"] = "Lisboa";
$ciudades["Francia"] = "Paris";
function recorre($ciudades) {
         if ( lcurrent($ciudades)) {
             reset($ciudades);
         }
         do {
             $valor = current($ciudades);
             echo ("El valor es: $valor<br>") ;
         jwhile (next($ciudades));
}
function recorre_atras($ciudades) {
         end($ciudades);
         do {
             $valor = current($ciudades);
```

```
            echo ("El valor es: $valor<br>") ;
        }while (prev($ciudades));
}
recorre(&$ciudades);
recorre_atras(&$ciudades);
?>
```

El navegador muestra:

```
El valor es: Badajoz
El valor es: Mérida
El valor es: Cáceres
El valor es: Plasencia
El valor es: Madrid
El valor es: Lisboa
El valor es: Paris
El valor es: Paris
El valor es: Lisboa
El valor es: Madrid
El valor es: Plasencia
El valor es: Cáceres
El valor es: Mérida
El valor es: Badajoz
```

Por último, para obtener el valor del índice puede utilizar la función key(). La función `recorre()` la puede actualizar de la siguiente forma:

```
function recorre($ciudades) {
         if ( ! current($ciudades) ) {
            reset($ciudades);
         }
         do {
            $valor = current($ciudades);
            $indice = key($ciudades);
            echo ("$indice: valor: $valor<br>");
         }while (next($ciudades));
}
```

## Intercambio de valores

La función `array_flip()` intercambia los valores de índices y datos, es decir, los índices serán guardados como datos y los valores serán sus nuevos índices.

```
<?php
function recorre ($numero) {
         foreach ($numero as $indice => $valor) {
            echo "$indice: $valor<br>",-
         }
}
```

```
$numero = array("uno" => l,"dos" => 2, "tres" => 3,"cuatro" => 4);
echo ("Números<br>" ) ;
recorre($numero);
echo ("Números intercambiados<br>") ;
recorre(array_flíp{$numero));
?>
```

El resultado es el esperado:

```
Números
uno: 1
dos : 2
tres: 3
cuatro: 4
Números intercambiados
1: uno
2 : dos
3: tres
4: cuatro
```

Puede ver el resultado en la figura 7.4.

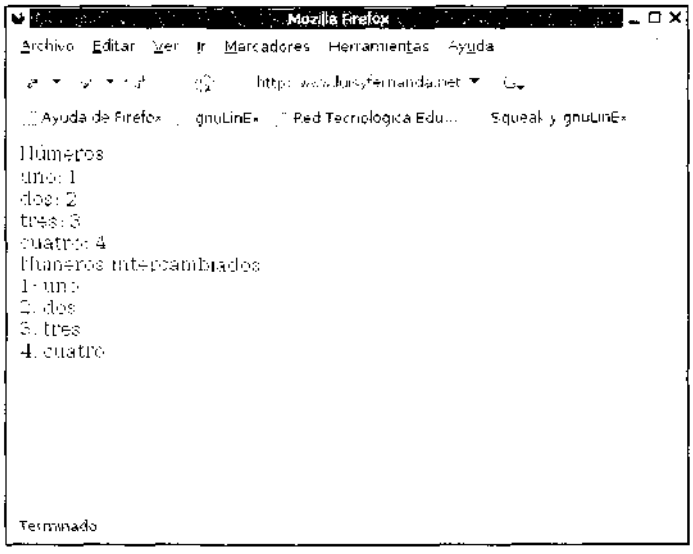



**Figura 7.4.** Intercambio de valores en un array.

## Inversión del contenido

También es posible ordenar a la inversa una lista de datos. Esto se consigue con la función `array_reverse ()` .

Si modifica el código anterior tendrá:

```
echo ("Números intercambiados<br>");
recorre(array^reverse($numero));
```

Dando como resultado:

```
Números
uno: 1
dos : 2
tres: 3
cuatro: 4
Números intercambiados
cuatro: 4
tres: 3
dos: 2
uno: 1
```

## Mezcla de los valores

Si necesita alguna vez elegir un valor aleatorio de un *array,* la función shuffle () puede ser de gran ayuda, ya que mezcla todos los valores cada vez que se ejecuta. Parte del código puede ser este:

```
echo ("Números mezclados<br>");
shuffle($numero);
recorre($numero);
```

## Pilas

Las pilas son estructuras de datos que se utilizan mucho en el mundo de la informática. La filosofía básica de las pilas es que todos los valores deben introducirse en un orden específico y sólo podemos recuperar esos valores en el orden inverso estricto al que fueron insertados.
Las pilas LIFO *(Last ln First Out:* el último en entrar es el primero en salir) utilizan unas funciones específicas que permiten manejar todos los datos. Array_push () permite insertar en la pila un dato y array_pop () extrae un valor. Es necesario, antes que nada, definir la variable como *array,* pues ninguna de las funciones descritas lo hace.

Array_push () y array_pop () toman como argumento la variable *array* y los valores a introducir.

```
<?php
$pila = array();
array_push($pila,"uno","dos","tres");
```

```
array_jpush($pila,"cuatro","cinco");
while ($valor = arrayjop($pila) ) {
       echo "valor extraído es $valor<br>";
}
?>
```

El resultado por pantalla muestra que la extracción de los valores se hace en orden inverso del que fueron introducidos:

```
valor extraído es cinco
valor extraído es cuatro
valor extraído es tres
valor extraído es dos
valor extraído es uno
```

## Ordenación de los valores

Finalmente, PHP ofrece una gran variedad de funciones para ordenar *arrays.* Las funciones pueden verse en la tabla siguiente:

**Tabla 7.1.** Funciones de ordenación.

| Función | Explicación |
|---|---|
| asort() | Ordena de forma ascendente el *array* pasado como argumento. Ordena las parejas índice/valor atendiendo al dato. Es un buen método para los *arrays* asociativos. |
| arsort() | Igual que `asort ()` , pero ordena en sentido descendente. |
| ksort() | Ordena de forma ascendente el *array* pasado como argumento. Ordena las parejas índice/valor atendiendo esta vez al índice. |
| krsort() | Igual que `ksort ()`, pero ordena en sentido descendente. |
| sort() | Ordena de forma ascendente el *array* pasado como argumento. Se pierde el valor asociativo entre el índice y el valor. |
| rsort() | Igual, pero en orden descendente. |

Puede ver actuar todas las funciones de ordenación en el ejemplo siguiente:

```
<?php
function recorre ($numero) {
          foreach ($numero as $indice => $valor) {
                   echo "$indice: $valor<br>";
```

```
    }
}
$pila =
array("cinco"=>5,"uno"=>l,"cuatro"=>4,"dos"=>2,"tres"=>3);
echo "Array sin ordenar<br>" ;
recorre($pila);
echo "Ordenación asort()<br>" ;
asort($pila);
recorre($pila);
echo "Ordenación arsort{)<br>" ;
arsort($pila);
recorre($pila);
echo "Ordenación ksort()<br>";
ksort($pila);
recorre($pila);
echo "Ordenación krsort()<br>";
krsort($pila),-
recorre($pila);
echo "Ordenación sort()<br>";
sort($pila);
recorre($pila);
echo "Ordenación rsort()<br>";
rsort($pila);
recorre($pila);
?>
```

La figura 7.5 muestra todos los tipos de ordenación que puede utilizar sobre un *array*.

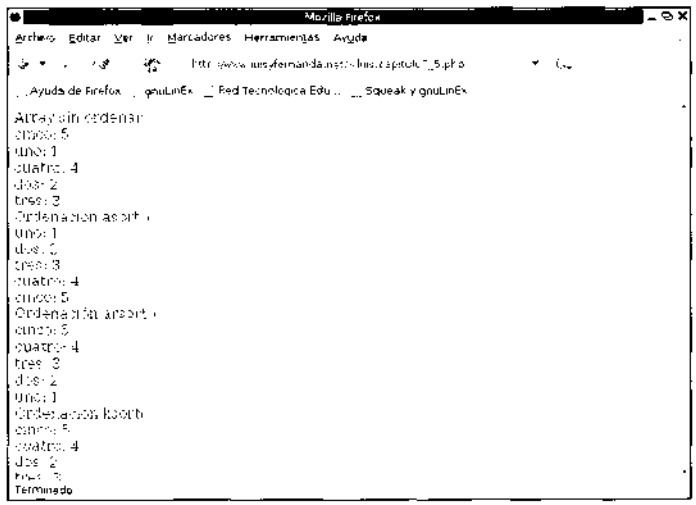



**Figura 7.5.** Diferentes ordenaciones de un array.

## Resumen

En este capítulo ha conocido el tipo de variable estrella de PHP, el *array*. Controlar los métodos y las funciones para extraer la información le dará ventaja en los capítulos posteriores. Muchas funciones que tienen que ver con el manejo de datos, como funciones de bases de datos, XML o generación de plantillas, devuelven un *array* como respuesta.

El tipo *array* es el vehículo estándar para la recuperación de datos estructurados como veremos en los próximos capítulos.


# Paso de información entre formularios

**En este capítulo aprenderá a:**

- Crear formularios para pasar datos entre páginas.
- Utilizar las variables súper-globales para recibir la información de un formulario.
- Diferenciar entre los métodos de envío GET y POST.
- Encontrar posibles errores revisando las variables súper-globales.

# Introducción

Hasta ahora, en los capítulos anteriores, hemos visto cómo construir diversos programas en PHP con un limitado radio de acción. La verdadera potencia del lenguaje reside en la aplicación de las técnicas aprendidas a través de varias páginas Web, pasando información de unas a otras. Como veremos más adelante, las dos técnicas principales son el uso de formularios y la utilización de la barra de dirección para pasar los valores.

# Argumentos GET

Los argumentos GET pasan la información como parte de la URL. Cuando utilice esta forma de trabajar, las parejas de variable / valor podrán verse en la casilla de dirección del navegador. Para probar esto podemos escribir un pequeño script que guardaremos como página Web.

```
<html>
<body>
<?php
echo "Variables pasadas mediante GET:<br>";
foreach  ($_GET as $indice => $valor)  {
         echo "$indice : $valor";
}
?>
</body>
</html>
```

Si ejecuta el código anterior, verá que únicamente se imprime en pantalla el texto que muestra la orden `echo`. En realidad esto es correcto, puesto que no hemos introducido ninguna variable tipo GET a la Web. Para hacerlo debemos escribir el nombre de la página Web seguido de un símbolo de interrogación (?) y el conjunto de parejas de variable / valor separadas del símbolo (&).

La dirección queda algo así: `formulario.php?variable1=Hola Mundo&variable2=1234`. Ahora el resultado es diferente. El bucle `foreach` inspecciona el *array* $_GET e imprime en pantalla:

```
Variables pasadas mediante GET:
variable1 : Hola Mundo
variable2 : 1234
```

El *array* $_GET es considerado en PHP 5 una variable súper-global. Este, en concreto, almacena todas las variables pasadas entre páginas mediante

el método GET. Para extraer el valor de una variable pasada por este método sólo hay que escribir $_GET["nombre_variable"] en la página que recibe los datos.

**Advertencia:**

*En versiones anteriores de PHP, para recuperar el valor de una variable, simplemente había que escribir un símbolo $ delante del nombre como* $nombre_variable. *Esto es porque el parámetro register_globals del fichero de configuración* php.ini *está, en PHP 5, a* off.

## Formularios con GET

La forma más utilizada para recabar información acerca de los usuarios es emplear formularios HTML. Los formularios pueden hacerse escribiendo en un editor de texto las etiquetas HTML o utilizando algún editor Web profesional como Macromedia Dreamweaver, Netscape Composer o Nvu. La siguiente página Web, llamada formulario.php, muestra cómo recoger información:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loóse.dtd">
<html>
<body>
<p>Introduzca sus datos personales:</p>
<form name="formulario" method="GET" action="formulario2.php">
<table width="50%"  border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td  width="24%">Nombre</td>
    <td width="76%"xinput name="nombre" type="text" id="nombre"x/td>
  </tr>
  <tr>
    <td>Apellidos</td>
    <tdxínput  name="apellidos"  type="text"  id="apellidos2"></td>
  </tr>
  <tr>
    <td>Correo</td>
    <tdxinput  name="correo"  type="text"  id="correo"x/td>
  </tr>
  <tr>
    <td>Estado  civil  </td>
    <tdxselect  name="estado"  id="estado">
      <option  value="Soltero">Soltero</option>
      <option  value="Casado">Casado</option>
```

```
    •coption  value="Divorciado">Divorciado</option>
    <option  value="Viudo">Viudo</option>
  </select></td>
 </tr>
 <tr>
  <td>N&uacute;mero hijos </td>
  <td> 0 <input name="hijos" type="radio" value="0" checked>
       1 <input name="hijos" type="radio" value="l">
       2 <input name="hijos" type="radio" value="2">
       3 <input name="hijos" type="radio" value="3"></td>
 </tr>
 <tr>
  <td>Gustos</td>
  <td> Inform&aacute;tica <input type="checkbox" name="gustos [] "
       value="Informática">
       Buceo <input type="checkbox" name="gustos [] " value="Buceo">
       Magia <input type="checkbox" name="gustos [ ] " value="Magia">
       Jazz <input type="checkbox" name="gustos [] " value="Jazz"></td>
 </tr>
 <tr>
  <tdxinput  type="submit"  name="Submit"  value="Enviar"></td>
  <td> </td>
 </tr>
</table>
</form>
<p>  </p>
</body>
</html>
```

La figura 8.1 muestra el aspecto en el navegador. Hay varios aspectos a tener en cuenta. Lo primero es que el apartado `action` de la etiqueta <FORM> contiene la página Web a la que se enviarán los datos y el apartado `method` permite seleccionar el método de envío; por ahora utilizaremos el método GET. La página `formulario2 . php` recoge la información del formulario y la imprime en pantalla utilizando un bucle `foreach` a la variable súper-global $_GET.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loóse.dtd">
<html>
<body>
<p>Datos  introducidos:</p>
<?php
foreach ($_GET as $indice => $valor) {
         echo "$indice: $valor<br>";
}
?>
</body>
</html>
```

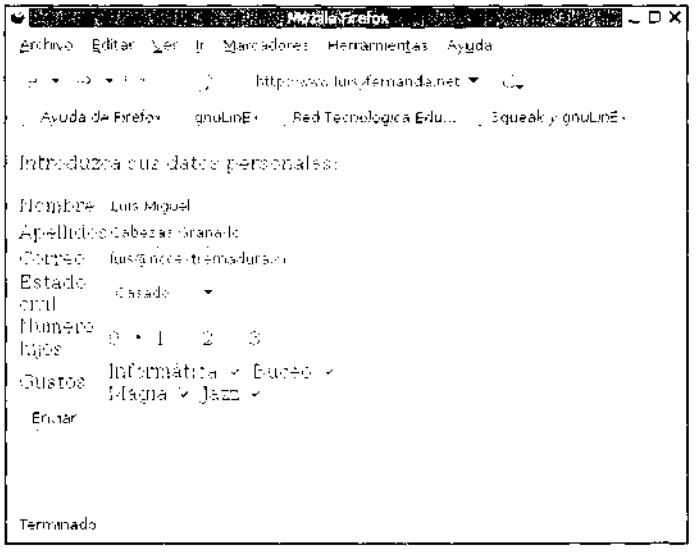



**Figura 8.1.** Formulario de datos personales.

La figura 8.2 muestra el resultado de enviar el formulario cumplimentado con nuestros datos.


**Figura 8.2.** Recepción del formulario de datos personales.

Para sacar los datos hemos utilizado un bucle que busca todos los valores dentro de $_GET. Otra forma de hacerlo es imprimir los datos uno a uno

si sabemos el nombre de las variables definidas en el formulario anterior de la siguiente forma: $_GET ["nombre"], $_GET ["apellidos"], $_GET ["correo"].

Existen en el formulario visto varias técnicas de recogida de información. Los cuadros de texto, que permiten escribir valores alfanuméricos, lista de selección y botones de opción, es donde podemos elegir una de las opciones existentes y casillas de verificación que permiten elegir 0 o más opciones existentes. Todas estas partes del formulario tienen una variable asociada que puede buscarse en $_GET.

El único método que puede plantear algún problema es el que utiliza casillas de verificación. Puesto que podemos elegir varias opciones, éstas son enviadas como un *array* y deben recuperarse atendiendo a esta razón.

**Advertencia:**

*Para que los valores de las casillas de verificación sean enviados como* array *hay que declararlas en el formulario como tal, escribiendo dos símbolos de corchetes en la propiedad* ñame *asociada a las casillas; por ejemplo:* <input type="checkbox" name="gustos[]" value="Informática">.

La página formulario2.php queda de esta forma, para que pueda recibir los gustos de los usuarios:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loóse.dtd">
<:html>
<body>
<p>Datos introducidos : </p>
<?php
foreach ($_GET as $indice => $valor) {
         echo "$indice: $valor<br>";
}
echo "<br>GUSTOS:<br>";
$gustos = $_GET["gustos"];
foreach ($gustos as $indice => $valor) {
         echo "$indice: $valor<br>";
}
?>
</body>
</html>
```

La figura 8.3 muestra la mejora incluida en el programa. Ahora es capaz de recuperar los gustos de los usuarios e imprimirlos en pantalla. Para esto

hemos añadido un bucle nuevo que inspecciona los valores enviados al *array* $gustos.


**Figura 8.3.** Recepción del formulario de datos personales con array.

## Paso de información con GET

Si se fija en la barra de dirección de los navegadores cuando visita páginas Web sobre noticias o carritos de la compra, en la URL van apareciendo variables que tienen que ver con los distintos menús que existen en la Web.

El siguiente ejemplo muestra una pequeña revista digital escrita en pocas líneas y en una sola página Web que cambia sus noticias eligiendo las distintas opciones del menú.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 TransitÍonal//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<?php
$menu = array (1 => "Editorial", 2 => "Opinión", 3 =>
"Regional", 4 => "Nacional");
echo  ("--Revista Digital--<brxbr>") ;
echo  ("--MENU--<br>" ) ;
foreach  ($menu as $indice => $valor)  {
          echo ("<a href = \"Índex.php?menu=$indice\">$valor</axbr>") ;
```

```
\
echo ("<br>--NOTICIAS--<br>");
switch ($_GET["menú"]) {
        case 1:
               echo ("Editorial") ;
               break;
        case 2:
               echo ("Opinión");
               break;
        case 3:
               echo ("Regional");
               break;
        case 4:
               echo ("Nacional");
               break;
        default:
               echo ("Noticias de Portada");
               break;
}
?>
</body>
</html>
```

El *array* $menu se carga de los distintos valores que podemos ir seleccionando.

En la actualidad estos valores se suelen extraer de una base de datos.

Para mostrar el menú se hace un bucle donde se extraen todos los valores e índices del *array* y se ponen como valores en los enlaces.

```
echo ("<a href=\"Índex.php?menu=$indice\">$valor</axbr>");
```

Los enlaces siempre se dirigen a la página principal, pero con un valor distinto en la variable menú.

La segunda parte de la Web es una elección múltiple mediante la estructura `switch`, donde, en función del valor de la variable menú pasada por GET, se eligen unas noticias vi otras.

Puede ver el resultado en la figura 8.4.


```
echo ("<a href=\"Índex.php?menu=$indice\">$valor</axbr>");
```

Los enlaces siempre se dirigen a la página principal, pero con un valor distinto en la variable menú.

La segunda parte de la Web es una elección múltiple mediante la estructura `switch`, donde, en función del valor de la variable menú pasada por GET, se eligen unas noticias u otras.

Puede ver el resultado en la figura 8.4.

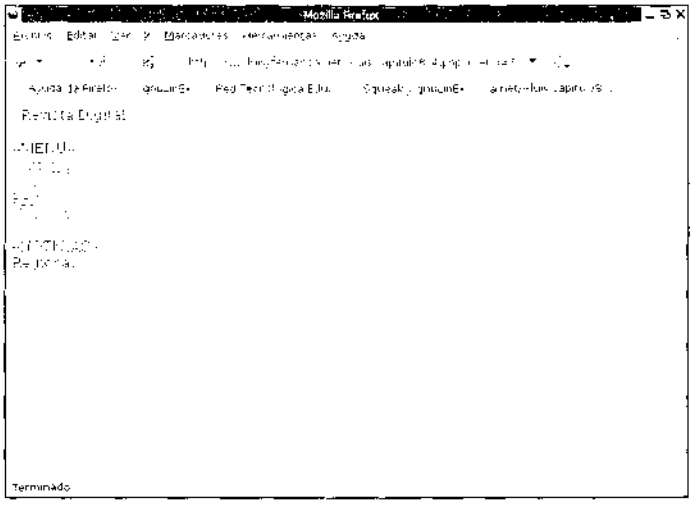



**Figura 8.4.** Estructura de la revista digital.

## Argumentos POST

La utilización del método GET es totalmente inseguro, porque no hay forma de ocultar datos privados en la dirección Web, tales como la contraseña de entrada o el número de cuenta bancaria.

El método POST arregla estos problemas, siendo el preferido de los desarrolladores. Este método es tan sencillo como cambiar en el formulario el método de envío de datos:

```
<p>Introduzca sus datos personales:</p>
<form name="formulario" method="POST" action="formulario2.php">
   <table width="50%" border="0" cellspacing="0" cellpadding="0">
```

En cuanto a la recuperación de los datos, también debe cambiar el nombre de la variable súper-global $_GET por $_POST de esta manera:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loóse.dtd">
<html>
<body>
<p>Datos introducidos:</p>
<?php
foreach ($_POST as $indice => $valor) {
         echo "$indice: $valor<br>";
}
echo "<br>GUSTOS:<br>";
$gustos = $_POST["gustos"];
```

```
foreach ($gustos as $indice => $valor) {
        echo "$indice: $valor<br>";
}
?>
</body>
</html>
```

## Variables súper-globales

Desde PHP 4 se vienen utilizando los *arrays* súper-globales para almacenar los valores que pasan de unas páginas a otras, se almacenan en el ordenador del usuario o en el servidor. En PHP 5 el uso de estas variables se hace obligatorio para recuperar todos los valores. Las variables a utilizar son:

- $_GET: Almacena las variables que se pasan desde un formulario mediante el método GET.
- $_POST: Almacena las variables pasadas por POST.
- $_COOKIE: Guarda los valores que están almacenados en *cookies.* Ya veremos en capítulos sucesivos cómo utilizarlo.
- $_SESSlON •. Guarda las variables que se pasan entre sesiones.
- $_SERVER: Contiene numerosos valores relativos al servidor.
- $_FILES : Los archivos que enviemos a través de un formulario serán recogidos en este *array.*

Una equivocación en el nombre de una variable puede llevarle a dar vueltas sobre el código hasta encontrar el error. Por eso, es necesario conocer el contenido de las variables súper-globales, así podrá averiguar si está pasando los datos correctamente. La función siguiente se puede utilizar cada vez que le surja un error de este tipo. La función muestra todos los valores de los *arrays* súper-globales:

```
<?php
function depuración() {
  echo "--Variables GET--<br>";
  foreach ($_GET as $indice => $valor) {
          echo "$indice: $valor<br>";
  }
echo "--Variables POST--<br>";
  foreach ($_POST as $indice => $valor) {
          echo "$indice: $valor<:br>";
  }
echo "--Variables COOKIES--<br>";
```

```
  foreach ($_COOKIE as $indice => $valor) {
          echo "$índice: $valor<br>";
  }
echo "--Variables SESSION--<br>";
  foreach ($_SESSION as $indice => $valor) {
          echo "$índice: $valor<br>";
  }
echo "--Variables SERVER-~<br>";
  foreach ($_SERVER as $indice => $valor) {
          echo "$indice: $valor<br>";
  }
}
depuración();
?>
```

La figura 8.5 muestra los valores de todas las variables súper-globales.


**Figura 8.5.** Variables súper-globales.

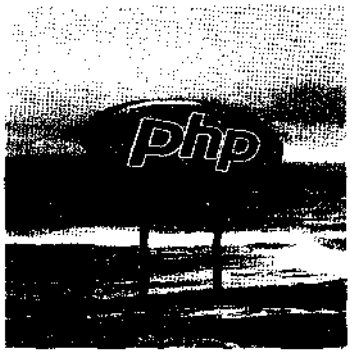
# Resumen

Hasta ahora, sólo habíamos tenido la oportunidad de crear programas en PHP que realizaran tareas en una única página Web.

Este capítulo ha dado una visión práctica de cómo utilizar los conceptos aprendidos para escribir programas distribuidos en varias páginas.

Aunque no ha aprendido ninguna función nueva, ha descubierto el verdadero secreto del diseño de una Web dinámica, el paso de variables.

Si viene de otras versiones de PHP, tendrá que tener en cuenta que la forma de recuperar las variables es, a partir de la versión 4.3, a través de las variables súper-globales.




# Programación orientada a objetos

**En este capítulo aprenderá a:**

- Diferenciar entre clase y objeto.
- Crear clases que hereden de otras clases.
- Utilizar los modificadores prívate, protected ypublic.
- Crear métodos estáticos y finales.
- Guardar el estado de un objeto mediante la señalización.
- Recuperar información referente a las clases,, métodos y propiedades.

# Introducción

PHP comenzó su andadura como un simple lenguaje de *scripts*. A medida que las versiones avanzaban, se fueron incluyendo algunas características que permitían programar con orientación a objetos. Con la aparición de PHP 5, los desabolladores tenemos una verdadera plataforma de programación orientada a objetos, gracias a la potencia del nuevo *engine* Zend 2.0, que permite ejecutar más rápido y eficientemente este tipo de programas.

La programación orientada a objetos utiliza los elementos clase y objeto con punto de inicio. Actualmente, en las carreras técnicas, es posible encontrar enseñanzas sobre lenguajes orientados a objetos como Java o C++.

**Nota:**

*En realidad, Java y C++ son considerados como seudo lenguajes orientados a objetos. Un ejemplo de lenguaje puro es SmallTalk o su versión gráfica Squeak (http://squeak.linex.org), con muchos años de antigüedad.*

La programación orientada a objetos en PHP 5 no necesita una sintaxis muy diferente, sino que es una aproximación a la realidad en otro camino diferente al acostumbrado. Por ejemplo, si piensa en un programa que tenga que averiguar la letra del DNI de un cliente, lo normal es que cree una función con un parámetro al que le pase el número de DNI y le devuelva la letra buscada.

Un desarrollo orientado a objetos pensaría en cada problema como una identidad única, como un objeto. Puesto que el DNI siempre va asociado a una persona, puede crear un objeto que identifique a personas, y dentro de ese objeto programar una función (método en este caso) que calcule la letra. La aproximación procedural (crear una función) no asocia esa función a personas directamente y le podría pasar cualquier número entero, recibiendo una respuesta válida. En cambio, la aproximación de objetos crea la función dentro del objeto cliente y sólo permite, mediante técnicas que veremos más adelante, recuperar la letra del DNI de una persona.

La programación orientada a objetos desarrolla nuevos conceptos y terminologías que deben aprenderse correctamente para generar buen código.

## Definición de clases

Una clase es un tipo de dato que contiene, en una misma estructura, variables y funciones. Una clase es una especie de plantilla desde la que los objetos son instanciados y toman su valor. Desde una clase se pueden construir varios objetos del mismo tipo. La definición es la siguiente:

```
class pagina_Web
{
   var  $titulo;
   function getTitulo()
   {
        return  $this->titulo;
   }
}
```

Mediante la palabra reservada `class` definimos una clase completa. Utilizando `var` podemos definir las variables que utilizaremos en el desarrollo del programa. La palabra reservada `var` desaparecerá en versiones siguientes de PHP, a favor de las palabras `public`, `private` y `protected`. Las funciones se definen como en el capítulo 5, siempre que estén dentro de la construcción de la clase. En este capítulo vamos a definir una clase completa cuya misión será mostrar una página Web. Podemos seguir con el ejemplo ampliando las funciones:

```
<?php
class pagina_Web
{
 var  $titulo;
 function setTitulo($titulo = "Titulo por defecto")
 {
  $this->titulo  =  $titulo;
 }
 function getTitulo()
 {
  return  $this->titulo;
 }
 function cabecera()
 {
  echo  ("<htmlxheadxtitle>");
  echo  $this->titulo;
  echo  ("</titlex/headxbody>"),•
 }
 function cuerpo()
 {
 echo("Este  es  el  cuerpo  de  la  página Web");
 }
```

```
    function pie ( )
    {
     echo  ("</bodyx/html>") ;
    }
    function  mostrar_pagina()
    {
     echo $thís->cabecera () ;
     echo $this->cuerpo() ;
     echo $this->pie();
    }
  }
```

Como puede ver, la clase se divide en varias funciones y en una sola variable. La variable $titulo guardará el título déla página Web. Las funciones cabecera (), cuerpo () y pie () son las encargadas de mostrar las distintas partes de una Web.

Estas funciones se llaman desde la funciónmostrar_pagina (). Es interesante observar que, en unas pocas líneas de código, hemos definido un objeto que podría utilizarse como base de muchas aplicaciones.

El operador $this es una variable que contiene el objeto actual desde el que la invocamos. Para acceder dentro de un objeto a funciones o variables propias, debe anteponerse al nombre de éstas la expresión $this- >. Para acceder desde la función mostrar_pagina () a cualquiera de las funciones de la clase tendrá que escribir:

```
$this->cabecera();
$this->cuerpo();
$this->pie();
```

## Instancia de clase

Para que el código funcione, necesita crear el objeto. Esto se hace utilizando el operador new seguido del nombre de la clase:

```
$pagina = new pagina_Web() ;
```

Con el código anterior hemos creado un objeto que contiene todas las variables y funciones de la clase pagina_Web (). La variable que contiene ese objeto es $pagina.

Para acceder a las funciones desde el nuevo objeto creado tenemos que utilizar la variable que contiene al objeto, en este caso $pagina seguido del operador - > y el nombre de la función. Para mostrar la Web tenemos que seguir estos sencillos pasos:

```
$pagina = new pagina_Web();
```

```
$pagina->setTitulo("Página  Web  nueva");
$pagina->mostrar__pagina();
```

## Función constructor

Existen algunas funciones especiales en la definición de una clase. La más importante es el constructor. Esta se ejecuta cada vez que se crea un nuevo objeto y permite crear las variables iniciales que se necesitan, como el título de la Web o el autor. El nombre de la función debe ser `__construct()`. Nuestro constructor se encargará de crear el título de la página Web que estamos creando:

```
function__construct($titulo)
{
   $this->setTitulo($titulo);
}
```

## Herencia

La programación orientada a objetos permite heredar de otras clases. Con esta técnica puede ahorrar mucho tiempo de trabajo. La clase hija (clase que hereda de otra) adquiere estas propiedades:

- Automáticamente obtiene todas las variables miembro de la clase padre.
- También obtiene todas las funciones miembro de la clase padre, que funcionarán exactamente de la misma forma.
- La clase hija puede a su vez definir nuevas variables y funciones.

La sintaxis es la siguiente:

```
class pagina_Web_formulario extends pagina_Web
{
 function formulario_inicio()
 {
  //Escribir el código necesario
}
}
```

La palabra reservada `extends` indica que la nueva clase creada será una extensión (heredará) de la clase que se escribe justo a la derecha de la definición. Puede crear un objeto formulario a partir de la clase `pagina_Web` de la siguiente forma:

```
class pagina_Web_formulario extends pagina_Web
{
```

```
    function  formularío_inicio($accion)
    {
     echo ("<form action=\"$accion\">");
    }
    function formulario_fin()
    {
     echo ("</form>");
    }
    function  formulario_caja_texto($nombre)
    {
     echo ("$nombre <input type=\"text\" name=\"$nombre\">");

    }
    function formulario_boton()
    {
     echo ("<input type=\"submit\" name=\"Submit\"
  valúe=\"Enviar\">");
    }
    function mostrar_pagina()
    {
     $this->cabecera();
     $this->formulario__inicio("Índex.php");
     $this->formulario__caja_texto("Nombre");
     $this->formulario_boton();
     $this->formulario_fin();
     $this->pie();
    }
  }
```

Si ahora ejecuta el código instanciando la clase que hemos creado, puede ver que las funciones `cabecera()`, `pie()`, incluso el constructor pueden ser utilizadas aún no perteneciendo a la clase `pagina_Web_formulario`.

```
$formulario = new pagina_Web_formulario{"Pagina con
formulario");
$formulario->mostrar_pagina();
```

La figura 9.1 muestra la ejecución del objeto anterior.

## Métodos o funciones de objeto

Cuando definimos una clase hija, las funciones de la clase padre son automáticamente heredadas. Se llama redefinición de métodos a la creación de funciones en la clase hija, con el mismo nombre que en la clase padre. En el ejemplo anterior, la función `mostrar_pagina()` de la clase `pagina_Web_formulario` está redefinida, ya que existe una función con

el mismo nombre en la clase pagina_Web. Al instanciar el objeto y ejecutar el método mostrar_pagina (), el método que se ha definido es la clase hija.

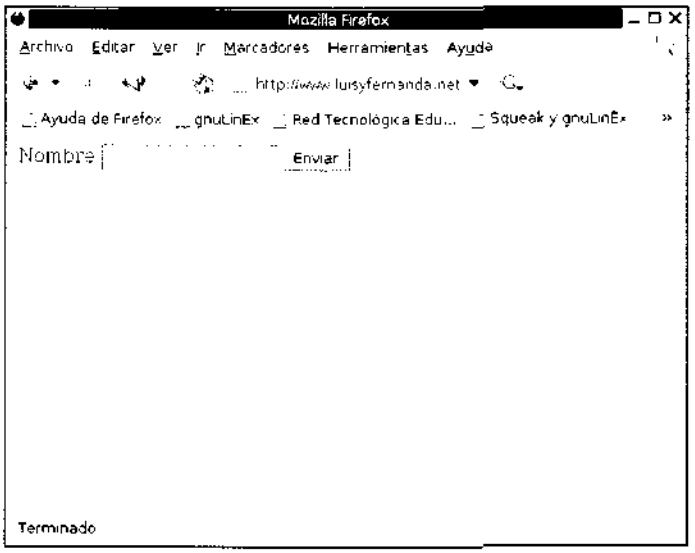



**Figura 9.1.** Creación de un formulario desde un objeto.

PHP 5, con su nuevo motor Zend Engine 2, introduce la palabra reservada final.

Si en la clase padre ponemos delante de cualquier función la palabra final, ésta función no podrá ser sobrecargada en las clases que la hereden.

También podemos declarar clases completas como final, lo que significará que no podrán ser heredadas.

```
final function mostrar_pagina()
{
  echo $this->cabecera();
  echo $this->cuerpo();
  echo $this->pie();
}
```

Si definimos la función anterior como final en la clase pagina_Web, la clase pagina_Web_formulario no podrá tener una función con este mismo nombre y mostrará un error en pantalla.

La figura 9.2 muestra el error que se produce al intentar sobrecargar un método que está marcado como final.

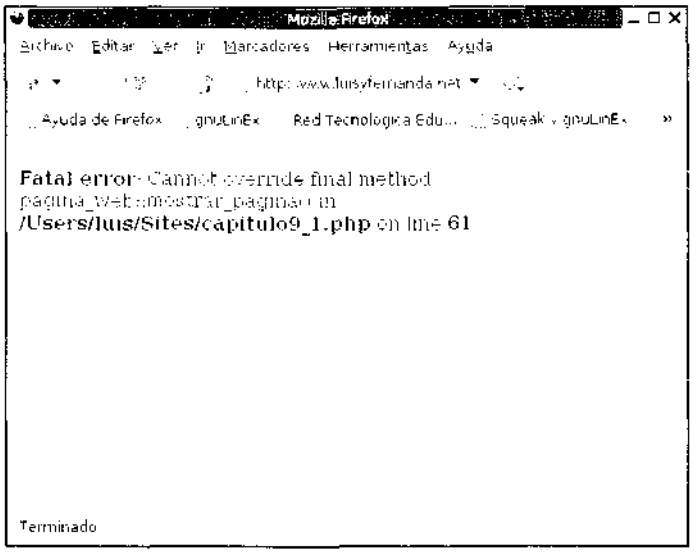



**Figura 9.2.** Error de sobrecarga de método.

## Herencia encadenada

Algunos lenguajes de programación permiten heredar de varias clases a la vez; esto es conocido como herencia múltiple. PHP 5 no permite herencia múltiple, pero sí herencia encadenada, es decir, permite heredar de varias clases padres correlativamente. Un ejemplo es:

```
class A

class B extends A

class C extends B

class D extends C
```

## Valores y alcance de variables

En versiones anteriores de PHP, los objetos eran pasados por valor a otras funciones, por lo tanto, el estado de los objetos no se conservaba. En PHP 5 los objetos son pasados por defecto por referencia, gracias al Zend Engine 2.

```
<?php
```

```
class Nombre
{
 var $nombre;
 function setNombre($nombre)
 {
  $this->nombre = $nombre;
 }
 function getNombre()
 {
  return $this->nombre;
 }
}
function cambiaNombre($objeto,$nombre)
{
 $objeto->setNombre($nombre);
}
$luis = new Nombre;
$luis->setNombre("Luis Miguel");
echo "El nombre del objeto es " . $luis->getNombre() . "<br>";
cambiaNombre($luis,"Pedro");
eche "El nombre del objeto es " . $luis->getNombre() . "<br>";
?>
```

El código muestra cómo cambiar las propiedades de un objeto desde una función con PHP 5.

El resultado en el navegador puede comprobarlo en la figura 9.3.


**Figura 9.3.** Los objetos se pasan por referencia.

Dentro de los objetos debe tener también cuidado con el alcance de las variables. Las variables definidas fuera del entorno del objeto no son accesibles desde los métodos de las clases, a menos que anteponga la palabra global.

## Miembros públicos, privados y protegidos

Mientras no se especifique otra cosa, los métodos y propiedades de una clase son siempre públicos, es decir, son accesibles desde fuera del objeto de la forma: objeto->función. Pero existe un camino para que las variables y los métodos no puedan ser accedidos desde fuera del objeto. Eche un vistazo ahora al objetopagina_Web. Su variable $titulo es público porque no lo hemos definido de alguna forma especial. Para cambiar el valor de esta variable podría utilizar la función asociada o directamente haciendo $objeto->titulo = "nuevo valor". Esta forma de actuar rompe con la filosofía de la programación orientada a objetos. Como norma general, las propiedades de los objetos deben estar ocultas al entorno exterior y tomar sus valores desde funciones definidas dentro del objeto.

## Métodos privados

Si declaramos un método o propiedad como privada, sólo se podrá acceder a él desde la clase que lo define. Las clases que hereden de una clase con métodos privados, no tendrán acceso a éstos métodos y tampoco será posible desde fuera del objeto hacer llamadas. Para hacer un método o propiedad privado, sólo hay que anteponer la palabra prívate delante de la declaración.

```
class pagina_Web
{
 prívate $titulo;
 function___construct($titulo)
 {
  $this->setTitulo($titulo);
 }
    prívate function setTitulo($titulo = "Titulo por defecto")
 {
  $this->titulo = $titulo;
 }
```

De esta manera, la clase pagina_Web_formulario, que hereda de pagina_Web, no puede acceder directamente a la función setTitulo ()

de la siguiente forma $this->setTitulo("Titulo nuevo") porque obtenemos un error de acceso ilegal. Después de esta modificación sólo es posible dar un título a la página Web a través del constructor que, como sí está dentro de la clase, tiene potestad para utilizarla.

## Métodos protegidos

Los métodos protegidos son menos restrictivos que los privados. Cuando declaramos un método como protegido, su visibilidad es en la clase que lo declara y en todas las clases que lo heredan. El ejemplo anterior puede modificarse de la siguiente forma:

```
class pagina_Web
{
protected  $titulo;
  function__construct($titulo)
  {
   $this->setTitulo($titulo);
  }
protected function setTitulo($titulo = "Titulo por defecto")
  {
   $this->titulo = $titulo;

  }
```

## Métodos públicos

Por defecto, todos los métodos y propiedades que se declaren en un objeto son públicos, a menos que lleven asociados la palabra protected o prívate. Como buena práctica de programación, las variables deben definirse como privadas o protegidas, para que no puedan modificarse desde fuera del objeto. Además, deben llevar dos métodos públicos asociados: setVariable () para dar un nuevo valor a la variable y getVariable () para recibir el valor actual de la propiedad. Veamos el ejemplo:

```
 class pagina_Web
{
 private  $titulo;
 function__construct($titulo)
 {
  $this->setTitulo($titulo);
 }
   public function setTitulo($titulo = "Titulo por defecto")
 {
  $this->titulo = $titulo;
 }
```

```
public function getTitulo()
{
 return $this->titulo;
}
```

# Interfaces

En proyectos profesionales a veces es necesario que un equipo de varias personas trabajen juntas. En este caso se hace imprescindible definir unas pautas generales de trabajo para que el resultado final sea el esperado. Si el desarrollo consiste en programar varios objetos, el analista de la aplicación puede definir la estructura básica en papel o crear una pequeña plantilla con métodos que el objeto final debería tener obligatoriamente. Esta plantilla es un interface y permite definir una clase con funciones definidas, pero sin desarrollar, que obliga a todas las clases que lo implementen a declarar estos métodos como mínimo.

```
interface Web
{
  public function setTitulo($titulo = "Titulo por defecto");
  public function getTitulo();
}
```

La interface anterior define la estructura básica que queremos para el objeto, declarando las funciones setTitulo y getTitulo. Para que una clase haga uso de la interface se declara de la siguiente forma:

```
class pagina_Web implements Web
```

# Clases abstractas

Un interface no permite crear el cuerpo de ninguna función, dejando esta tarea a las clases que la implementen. Las clases abstractas permiten definir funciones que deben implementarse obligatoriamente en los objetos que hereden y, además, permiten definir funciones completas que pueden heredarse. Las clases abstractas deben llevar la palabra reservadaabstract en la declaración de la clase y en todos los métodos que sólo definan su nombre.

```
abstract class Web
{
 protected $titulo;
```

```
    public function setTitulo($titulo = "Titulo por defecto")
    {
      $this->titulo = $titulo;
    }
    abstract public function getTitulo();
  }
```

En lugar de un `interface`, hemos optado por utilizar una clase abstracta, porque tenemos claro que el método `setTitulo()` tiene que funcionar así. También se define el método abstracto `getTitulo()`, que obligatoriamente debe ser declarado en la clase que herede de ésta. En realidad, un `interface` es una clase que tiene todos sus métodos abstractos.

## Clases con métodos estáticos

En PHP 5 puede declarar funciones dentro de una clase que no utilicen propiedades o métodos de la misma. Estos métodos pueden calcular valores numéricos, hacer una conexión a una base de datos o comprobar que un correo electrónico esté bien definido. Aunque no existe ninguna palabra reservada para definirlo, son conocidos como métodos estáticos y pueden ser utilizados sin necesidad de instanciar un objeto.

```
<?php
class Nombre
{
  protected $nombre;
  public function getNombre(}

    return $this->nombre;

  public function setNombre($nombre)

    $this->nombre = $nombre;

  public function NombreDefecto()

    return "Luis Miguel<br>";

}
$luis = new Nombre;
echo $luis->NombreDefecto();
//También se puede acceder al nombre por defecto
echo Nombre::NombreDefecto();
?>
```

Para acceder a un método estático desde el cuerpo del programa se escribe el nombre de la clase que lo implementa seguido del operador ( :: ) y el nombre del método. En el ejemplo vemos que se puede acceder al método NombreDefecto () creando un objeto o mediante la construcción Nombre: : NombreDefecto. Si intenta llamar a un método que no es estático de esta forma se imprimirá un error en pantalla. La figura 9.4 muestra el error que recibe si intenta llamar a un método no estático desde fuera de un objeto.

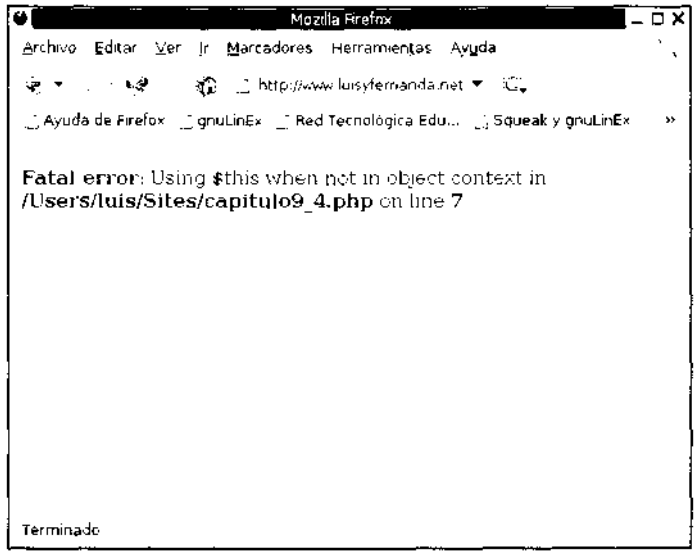



**Figura 9.4.** Error al utilizar métodos no estáticos.

## Llamadas a funciones padre

El operador ( :: ) se puede utilizar también dentro de una clase para hacer llamadas a funciones de la clase padre que estén sobrecargadas. Si define un constructor para la clase padre y otro para la clase hijo, cuando cree el objeto, el constructor que se ejecuta es el de nivel inferior, el constructor hijo. Para llamar al constructor padre debe utilizarse la nomenclatura Nombre_clase_padre : :__construct () .

```
<?php
class Nombre
{
 protected $nombre;
```

```
  f unet ion_construct ( $nombre)
  {
   $this->nombre = $nombre;
  }
  publie function getNombre()
  {
   return $this->nombre;
  }
  publie function setNombre($nombre)
  {
   $this->nombre = $nombre;
  }
  publie function NombreDefecto()
  {
   return "Luis Miguel<br>";
  }
}
class Apellido extends Nombre
{
  protected $apellidos;
  function___construct($nombre,$apellidos)
  {
   $this->apellidos = $apellidos;
   Nombre: :__construct(Snombre);
  }
  publie function getApellidos ()
  {
   return $this->apellidos;
  }
}
$luis = new Apellido("Luis Miguel","Cabezas Granado");
echo $luis->getNombre() . "<br>";
echo $luis->getApellidos();
?>
```

Puede ver que el constructor de la clase Apellido inicializa la variable $apellidos y, seguidamente, llama al constructor de la clase padre Nombre.

Esta forma de actuar puede resultar complicada si no conoce el nombre de la clase padre. La solución a este pequeño inconveniente la tenemos en la palabra parent, que hace alusión a la clase de la que heredamos. El código anterior puede quedar de esta forma:

```
class Apellido extends Nombre
{
  protected $apellidos;
  function___construct($nombre,$apellidos)
  {
   $this->apellidos = $apellidos;
```

```
    parent::__construct($nombre);
  }
  public function getApellidos()
  {
    return $this->apellidos;
  }
}
```

## Sobrecarga de métodos

Algunos lenguajes de programación orientados a objetos permiten declarar más de un método con el mismo nombre. La definición de estos métodos varía en el número de parámetros o en los tipos de datos de los argumentos. PHP 5 no permite sobrecarga de métodos, pero puede simular esta actuación empleando la técnica de las funciones con un número variable de parámetros. El constructor de la clase `Apellido` puede quedar de la siguiente forma:

```
class Apellido extends Nombre
{
  protected $apellidos;
  function __construct($nombre)
  {
    if (func_num_args()==2) {
      $apellidos = func_get_arg(1);
      $this->nombre = $nombre;
      $this->apellidos = $apellidos;
      parent::__construct($nombre);
    } else {
      $this->nombre = $nombre;
      $this->apellidos = $apellidos;
      parent::__construct($nombre);
    }
  }
  public function getApellidos()
  {
    return $this->apellidos;
  }
}
```

## Señalización

Los objetos son, en realidad, un conjunto de datos y funciones que guardan una serie de estados de ejecución. Este conjunto de *bits* se pueden al-

macenar, en un momento dado, y recuperar justo en el mismo estado en el que se encontraba cuando lo guardamos. Esta técnica se llama señalización y permite almacenar y recuperar el conjunto de *bits* que forman un objeto.

PHP 5 ofrece dos funciones, serialize() y unserialize(), que toman como parámetro un objeto y devuelven una cadena de caracteres. Los objetos serializados los podemos almacenar en base de datos, ficheros, etcétera.

```
$luis = new Apellido("Luis Miguel");
echo $luis->getNombre() . "<br>";
echo $luis->getApellidos(),-
$manuel = serialize($luis);
$pedro = unserialize($manuel);
echo $pedro->getNombre();
```

El código anterior crea el objeto $luis y utiliza los métodos propios de la clase y los heredados. Después, utilizamos la función serialize() para grabar el estado del objeto como un conjunto de caracteres en la variable $manuel. La variable $pedro contiene el resultado de aplicar unserialize() a $manuel.

De esta forma, conseguimos recuperar el objeto $luis con los datos que ya habíamos almacenado.

# Funciones de manejo de clases

Las funciones siguientes son útiles para recuperar información sobre la herencia de las clases, sobre métodos o variables miembro o llamadas a las funciones.

**Tabla 9.1.** Funciones de clases.

| Función | Descrición |
|---|---|
| get class() | Devuelve el nombre de la clase que ¡mplementa el objeto pasado como parámetro. |
| get_parent_class 0 | Devuelve el nombre de la clase padre. |
| class_exíst() | Si el parámetro es el nombre de una clase, devuelve un *true.* |
| get_declared_class() | Devuelve un *array* con el nombre de todas las clases definidas en el código actual. |

| Función | Descrición |
|---|---|
| is_subclass_of () | Toma dos parámetros. Si el primer parámetro es una subclase del segundo parámetro la función devuelve un *true.* |
| is_a()get_class_vars() | Devuelve un *array* con parejas variable / valor de una clase, que tengan valores por defecto. |
| get_object_vars () | Devuelve un *array* con parejas variable / valor de un objeto, que tengan valores por defecto. |
| method_exists () | Toma dos parámetros. Si el objeto pasado tiene un método con el nombre del segundo parámetro, la función devuelve *true.* |
| get_class_method () | Devuelve un *array de* métodos definidos en una clase. |

Puede ver el funcionamiento de la función get_class () en el ejemplo siguiente y el resultado en la figura 9.5.

```
$luis = new Nombre;
echo $luis->NombreDefecto();
echo 'La variable $luis es un objeto del tipo: '
get_class($luis);
```

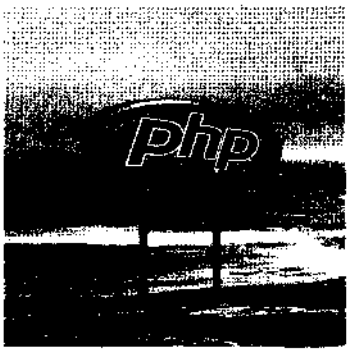

**Figura 9.5.** Nombre de la clase de un objeto.

## Resumen

Existe, en el mundo de los desabolladores de PHP, una antigua disputa sobre la mejor forma de afrontar un proyecto Web. Unos opinan que la forma más rápida de crear una Web es realizar funciones separadas en ficheros y, desde las páginas del proyecto, hacer las llamadas necesarias.

La otra vertiente opina que merece la pena dedicar un poco más de tiempo al desarrollo de objetos, porque después redundará en un gran beneficio en futuros programas. En realidad depende del proyecto. Si sólo necesita crear un formulario para recibir datos de un usuario, quizá no merezca la pena crear un objeto y sí algunas funciones de control. Si su proyecto va a ser muy extenso, la recomendación es que prepare un compendio de objetos que se interrelacionen para llevar a buen término el programa.

Puesto que el libro versa sobre PHP 5 y sus mejoras con respecto a las versiones anteriores, intentaremos en los capítulos siguientes realizar la mayoría de los ejemplos en metodología orientada a objetos.




Capítulo 11

# Ficheros y almacenamiento de datos

**En este capítulo aprenderá a:**

- Abrir y cerrar ficheros de lectura y escritura.
- Utilizar funciones para leer el contenido de un fichero.
- Realizar programas para la escritura de información en un fichero.
- Gestionar los ficheros y directorios de un sistema operativo.
- Crear ficheros de configuración.
- Generar cabeceras para la descarga de archivos.

# Introducción

Antes de que la proliferación de los gestores de bases de datos se hiciera efectiva, los lenguajes de programación utilizaban el acceso a ficheros para almacenar sus datos. Así, los lenguajes COBOL, Pascal o incluso C comenzaron a utilizar el manejo de archivos en aquellos programas que necesitaban guardar datos para recuperarlos posteriormente.

Actualmente, la información puede almacenarse en ficheros o en bases de datos. Los ficheros pueden almacenar textos que pueden ser leídos fácilmente por nosotros y por un ordenador. Además, existe una serie de utilidades que permiten al sistema operativo (gnuLinux o MacOSX) interactuar con ellos, hacer búsquedas, etcétera.

**Nota:**

*El sistema operativo Windows tiene algunas restricciones con respecto al manejo de ficheros y no se recomienda para gestionar los archivos.*

# Funciones de lectura y escritura de ficheros

Existen, en PHP 5, un conjunto muy numeroso de funciones que permiten manejar ficheros de diferentes formas. La idea básica para trabajar con ficheros es la siguiente:

- Abrir el fichero para su lectura / escritura.
- Leer el fichero.
- Cerrar el fichero (puede hacerse más tarde).
- Realizar las operaciones necesarias en el contenido.
- Escribir los datos.

# Abrir el fichero

La función `fopen ()` asigna a una variable un puntero (un descriptor) al fichero que quiera abrir. La variable puede utilizarse después para hacer cualquier tipo de operación. Si el fichero que intenta abrir no existe o no puede utilizarse en ese momento, `fopen ()` devolverá un valor *false.*

Los ficheros pueden abrirse en varios modos tal y como indica la tabla 10.1.

**Tabla 10.1.** Modos de apertura de los ficheros.

| Modo | Explicación |
|---|---|
| Sólo lectura ('r') | Sólo permite leer el fichero. Si intenta hacer otra cosa le saldrá un error. |
| Lectura y escritura si existe el fichero ('r+') | Permite leer y escribir un fichero que exista. El contenido del fichero no se borra, sino que se escribe a continuación. |
| Sólo escritura ('w') | Sólo se puede escribir. Sobrescribe cualquier fichero afectado. |
| Escritura y lectura ('w+') | Si el fichero no existe intenta crearlo. |
| Escritura al final ("a") | Escribe siempre al final del fichero tanto si existe como si no. |
| Lectura y escritura al final ("a+") | Escribe al final del fichero y permite leer. |

Desde la versión 4 de PHP puede abrir ficheros remotos que estén almacenados en servidores. Así, podrá utilizar el protocolo HTTP o FTP para recuperar datos remotamente. Puede ver como ejemplo varias formas de abrir un fichero con `fopen()`:

```
<?php
//Los ficheros HTTP solo pueden abrirse de lectura
$descriptor = fopen("http://www.luisyfernanda.net/
noticias.txt","r");
//FTP permite abrir ficheros de lectura y escritura
$descriptor2 = fopen("ftp://
usuario:contraseña@luisyfernanda.net/noticias.txt","r");
//Apertura normal de un fichero guardado en el ordenador local
$descriptor3 = fopen("../noticias.txt", "a+");
?>
```

## Lectura de ficheros

`fread()` tiene como parámetro un descriptor de fichero, devuelto por la función `fopen()` y el tamaño del bloque de datos que queremos leer.

```
$variable = fread($descriptor, 4000);
```

Normalmente no conocerá el tamaño del fichero y, por lo tanto, debe utilizar alguna técnica para leerlo correctamente. Lo más sencillo es apoyar-

se en la función filesize() que devuelve el tamaño en *bytes* del fichero que pase como argumento. Así, la lectura anterior quedaría:

```
$variable = fread($descriptor, filesize("noticias.txt"));
```

> **Advertencia:**
>
> *Tiene que notar que el lector* filesize() *no toma como parámetro el descriptor devuelto por la función* fopenO, *sino el nombre del fichero que estamos tratando.*

La función anterior devuelve un bloque completo de caracteres leídos desde un fichero y puede hacerse poco manejable cuando intente buscar algún dato específico. PHP 5 posee algunas funciones para leer línea a línea desde un fichero, fread() tiene como parámetros el descriptor del fichero y un bloque máximo de caracteres a leer. La utilización práctica es la siguiente:

```
<?php
$descriptor = fopen ("libro.txt","a +");
$linea_numero = 1;
while (!feof($descriptor)) {
        $linea = fgets($descriptor,4096);
        echo "línea número: $numero_linea es: $linea";
        $linea_numero++;
}
fclose  ($descriptor);
?>
```

El bucle actúa hasta que la función feof() encuentra el final del fichero apuntado por el descriptor. Mientras que no sea el fin del fichero, la función fgets() irá leyendo línea a línea todo el contenido e imprimiéndolo en pantalla.

La forma ideal de trabajar con ficheros es ir leyendo poco a poco su contenido, para poder buscar datos que nos interesen en cada línea o conjunto de caracteres. Existe una función que nos permite almacenar todo el contenido de un fichero en un *array* para su posterior proceso con las funciones propias de las colecciones de datos. La función explode() toma como parámetros un carácter por donde cortar la cadena y el nombre de la variable que contiene los datos. Podemos ver su funcionamiento en el ejemplo:

```
<?php
$descriptor = fopen ("libro.txt","a+");
$linea__numero = 1;
```

```
$archivo = "";
while ( !feof($descriptor) ) {
        $linea = fgets($descriptor,4096);
        $archivo = $archivo . "línea número: $numero__linea es:
$linea";
   $1inea_numero++;
}
fclose($descriptor);
$lista = explode(" ",$archivo);
?>
```

En este caso, `explode ()` busca en la variable `$archivo` todos los espacios en blanco y va almacenando en el *array* caracteres que estén entre dos ocurrencias (entre dos espacios). Así, obtenemos una gran lista de palabras almacenadas por separado, que hacen muy fácil buscar, ordenar o borrar cualquier contenido.

Por último, la función `fgete ()` permite leer un fichero carácter a carácter. Esto es muy útil cuando construimos un parse (un programa que interpreta el contenido de un fichero), y debemos ir evaluando las palabras letra por letra para actuar en consecuencia.

## Escritura de ficheros

Las funciones para escribir ficheros son tan sencillas de utilizar como las de lectura. La función principal `fwrite ()` escribe en un fichero, apuntado por un descriptor, una cadena de caracteres. Es imprescindible que el fichero esté abierto en uno de los modos de escritura para que no dé error.

Veamos ahora un ejemplo completo. El código siguiente es un pequeño libro de visitas, donde todo aquel que navegue por nuestra Web puede dejar mensajes de agradecimiento y ánimo. Aunque no es el mejor libro de visitas escrito nunca, sí es un buen ejemplo de cómo utilizar la lectura v escritura de archivos dentro de un entorno orientado a objetos.

```
<?php
class libro_visitas
{
 prívate $tamano;
 prívate $nombre;
 prívate $descriptor;
 prívate $contenido;
 function___construct($nombre)
 {
          $this->nombre = $nombre;
 \
```

```
 public function leer_libro{)
 {
          $this->descriptor = fopen ($this->nombre,"a+");
          $this->tamano=filesize($this->nombre);
          if ($this->tamano > 0) {
                   $this->contenido = fread($this-
                   >descriptor, $this->tamano);
          }
          fclose ($this->descriptor);
          return $this->contenido;
 }
 public function escribir__libro ($riombre, $texto)
 {
          $this->descriptor = fopen ($this->nombre,"a+");
          $nombre = "<trxtd bgcolor=\"#CCCCCC\">$nombre</td></tr>";
          $texto = "<trxtd bgcolor=\"#FFFFFF\">$texto</tdx/tr>" ;
          fwrite($this->descriptor,$nombre);
          fwrite($this->descriptor,$texto);
          fclose ($this->descriptor) ;
 }
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loóse.dtd">
<htral>
<head>
<title>Libro de visitas</title>
<meta http-eguiv="Content-Type" content="text/html; charset=iso-8859-l">
</head>
<body>
<p align="center"xstrong>Libro de visitas
</strongx/p>
<table width="100%"  border="0" cellspacing="0" cellpadding="0">


<?php
if (isset ($_POST["nombre"] )  |] isset($_POST["texto"])) {
          $libro = new libro_visitas("libro.txt");
          $libro>escribir_libro($_POST["nombre"],$_POST["texto"]);
}
$libro_ver = new libro__visitas("libro.txt");
echo $libro_ver->leer_libro();
?>
</table>
<form action="libro.php" raethod="POST" >
Nombre:
 <input name="nombre"  tYpe="text"  id="nombre">
</p>
<p>Texto:
```

```
  <textarea name="texto" id="texto"></textarea>
</p>
<p>
  <input type="submit" name="Submit" value="Enviar" >
</p>
</form>
</body>
</html>
```

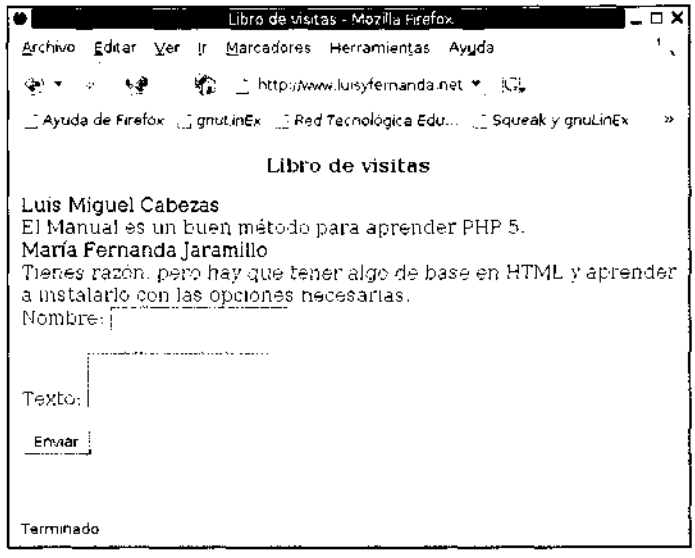



**Figura 10.1.** Libro de visitas con ficheros.

La figura 10.1 muestra el resultado del libro de visitas. Como puede ver, el código tiene dos partes diferenciadas. La primera es la definición de una clase con tres métodos, que es la encargada de almacenar y recuperar los datos. La segunda parte es el código HTML que nos permite visualizar los comentarios de las personas que nos visitan. Lo más recomendable es separar ambas partes en dos ficheros distintos y hacer uso de la función `require_once()` para utilizar el código PHP.

Si se fija en el código HTML, hay una parte central escrita en PHP, que comprueba si existen datos pasados mediante el método POST, es decir, si existen datos para escribir en el fichero. Si es así, creamos el objeto $1 `libro` y llamamos al `método escribir_libro()`, que se encarga de guardar los datos correctamente. Si no hay datos que guardar, simplemente se crea un objeto y se lee el archivo donde están los comentarios.

La clase libro_visitas escribe en el fichero filas de una tabla, que después se interpretan en el HTML colocando todo correctamente.

# Sistema de ficheros y directorios

Existe un conjunto de funciones que permiten manejar ficheros y directorios del sistema. Estas funciones son muy parecidas a comandos de Unix / gnuLinux y seguramente le resulten familiar al lector.

## Copiar, borrar y renombrar

Las operaciones básicas de manejo de ficheros y que se aprenden cuando comenzamos a utilizar un sistema operativo concreto son las de copia, borrado y modificación.

Para copiar un fichero, se utiliza la función copy (), que toma como parámetros el fichero que quiere copiar y el destino donde quiere guardar la copia.

Puede añadir un método nuevo a la clase libro_visitas para que realice funciones de copia de seguridad:

```
public function copia_seguridad()
{
  copy ($this->nombre, "copia.txt");
}
```

La siguiente función, unlink (), toma como parámetro el nombre de un fichero y lo borra. Puede hacer uso de esta función para ampliar su clase y permitir que se puedan borrar los ficheros que almacenan los libros de visitas:

```
public function borrar_libro()
{
  unlink($this->nombre);
}
```

Como el método copia_seguridad () sólo permite almacenar un fichero, puede ampliar su gama de métodos y añadir uno que le dé la posibilidad de renombrar el fichero copia.txt a otro nombre.

```
public function renombrar_copia($nuevo_nombre)
{
  rename($this->nombre,$nuevo__nombre);
}
```

## Funciones de comprobación

Es necesario, antes de borrar, copiar o renombrar un fichero, hacer una comprobación mínima para saber si el fichero existe.

De esta forma, evitará errores innecesarios y tendrá más control en el flujo de su programa.

La función `file_exists ()` comprueba si un fichero existe, dando el valor *tnte* en caso afirmativo *yfalse* en caso contrario.

El método `borrar_libro ()` puede quedar de la siguiente forma:

```
public function borrar_libro()
{
   if (file_exists($this->nombre)) {
            unlink($this->nombre);
   }
}
```

En los sistemas operativos UNIX / gnuLinux todos los recursos hardware se corresponden con un fichero. Puede que alguna vez necesite saber el tipo de fichero con el que va a tratar (fichero, directorio, recurso). La función `filetype ()` devuelve un valor que indica el tipo de fichero que corresponde al valor pasado como parámetro.

**Tabla 10.2.** Tipos de ficheros.

| Valor | Descripción |
|---|---|
| fifo | Es de tipo FIFO |
| char | Dispositivo de carácter |
| dir | Directorio |
| block | Dispositivo especial de bloques |
| link | Enlace |
| file | Fichero |
| unknown | Tipo desconocido |

Además de la función anterior, existe un conjunto de funciones que preguntan directamente al archivo por su tipo y el resultado es *true ofalse* en función de la veracidad de la pregunta. Las funciones `is_dir ()`, `is__executable ()`, `is_file ()` y `is_link ()`, preguntan a su argumento si es del tipo directorio, ejecutable, fichero o enlace respectivamente.

## Directorios

Es muy útil disponer de un gestor de ficheros Web que nos permita navegar entre los archivos y directorios. Para hacer esto en PHP, existen dos métodos. El primero utiliza las funciones opendir (), readdir () y closedir (), muy parecidas en su funcionamiento a fopen () , fread () y fcióse (). Lo mejor es ver cómo trabajan juntas con un ejemplo:

```
<?php
$directorio = "./";
$descriptor = opendir($directorio);
while ($entrada = readdir($descriptor) ) {
        if (is_dir($directorio.$entrada) ) {
                    echo "[Directorio] " . $entrada . "<br>"•
        } elseif ( is_file ($directorio . $entrada) ) {
                    echo " [Fichero] " . $entrada . "<br>";
        }
}
closedir($descriptor);
?>
```

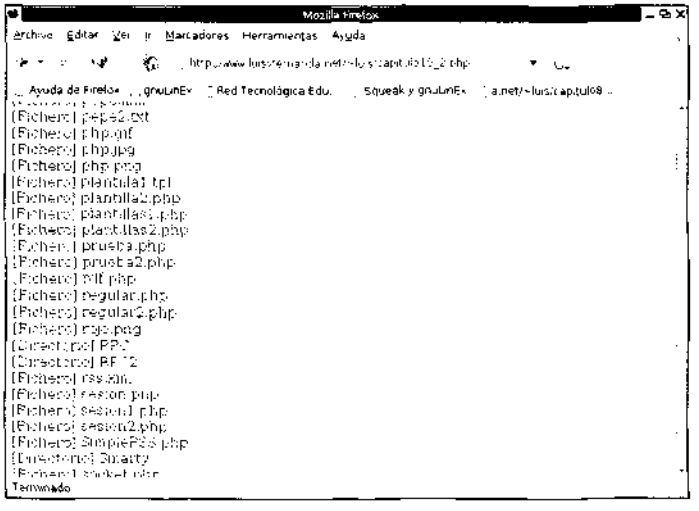



**Figura 10.2.** Listado de ficheros y directorios.

Como puede intuir, no existe ninguna diferencia significativa entre abrir un fichero o abrir un directorio. Lo primero es crear un puntero hacia el directorio que quiera observar y guardarlo en $descriptor. El bucle va leyendo entrada por entrada con la función readdir () y, dependiendo

del resultado de las funciones is_dir() o is_file() se imprimirá el directorio o el fichero pertinente.

Existe una segunda forma de visualizar lo mismo, pero utilizando programación orientada a objetos. PHP 5 provee un objeto que implementa toda la lógica vista anteriormente. La función dir() devuelve un objeto que permite examinar el directorio pasado como parámetro. El objeto posee un método llamado read() que va leyendo todas las entradas y un método cióse(), que cierra el objeto. El ejemplo anterior puede verse así:

```
<?php
$directorio = "./";
$fichero = dir({directorio);
while ($entrada = $fichero->read()) {
        if (is_dir({directorio.{entrada)) {
                echo "[Directorio] " . $entrada . "<br>";
        } elseif (is_file($directorio.$entrada)) {
                echo "[Fichero] " . $entrada . "<br>";
        }
}
$fichero->close();
?>
```

## Ficheros de configuración

Cuando sea un excelente programador en PHP 5, necesitará conocer técnicas para almacenar datos de configuración en un fichero. Una técnica habitual es crear un fichero con variables de PHP que toman los valores adecuados y, en el transcurso del programa, utilizar una llamada a la función requirejonee() para poder utilizarlas. De esta forma podríamos utilizar el siguiente ejemplo como configuración para conectar a una base de datos:

```
<?php
{servidor = "localhost";
{usuario = "luis";
$password = "secreta";
$base_datos = "anaya";
?>
```

Otra alternativa es utilizar un tipo de fichero de configuración que utiliza la misma estructura que el fichero php.ini.

La estructura es la siguiente:

```
;Los comentarios van después de un punto y coma
```

```
[Base__datos]
servidor=localhost
usuario=luis
password=secreta
base_datos=anaya
;Creamos un nuevo apartado
[Preferencias]
color=rojo
tamanio=medio
fuente=verdana
```

Este archivo almacena las variables que necesite en el transcurso del programa.

Para leerlo existe una sentencia especial que almacena el resultado en un *array*.

Para acceder a cada uno de los campos hay que utilizar la notación de un *array* de dos dimensiones como muestra el ejemplo.

La figura 10.3 muestra el resultado de leer un archivo de configuración.

```
<?php
$configuracion = parse_ini_file("configuración.ini",TRUE);
//Ejemplos de variables
echo  $configuracion["Base__datos"] ["servidor"];
echo  $configuracion["Preferencias"] ["fuente"] ;
?>
```

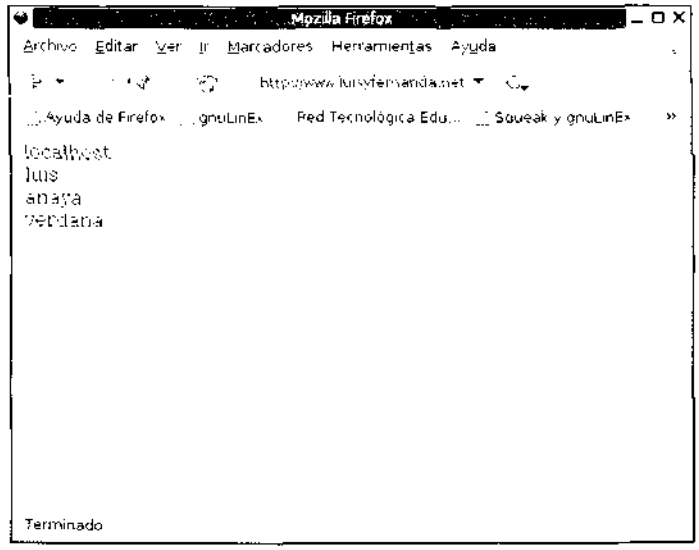



**Figura 10.3.** Lectura de un fichero de configuración.

## Manejo de ficheros en el servidor

Conociendo todas las posibilidades que ofrece PHP 5 a la hora de controlar archivos, llega la hora de aplicar a la práctica casos reales de funcionamiento. Los dos usos más comunes de manejo de ficheros en la Web tienen que ver con la subida de los mismos a un servidor, o con la descarga desde una página Web.

## Subida de ficheros

Cuando utilice un formulario Web, puede introducir los datos solicitados a través de cajas de texto, listas desplegables o botones de selección. Además existe una etiqueta HTML que permite elegir un fichero de nuestro ordenador local y subirlo al servidor. Puede crear un formulario mínimo que contenga la etiqueta en el siguiente ejemplo:

```
<HTML>
<BODY>
<form action="subir.php" method="POST" enctype="multipart/form-data">
Introduce el fichero: <input type="file" name="fichero" ><br>
<input type="hidden" ñame"TAMANIO" valué = "1000">
<input type="subrait">
</form>
</BODY>
</HTML>
```

Como puede ver, el atributo enctype debe tener el valor multipart= form-data para que el formulario trabaje correctamente. Además, hemos añadido una etiqueta llamada TAMANIO, que guarda el valor máximo en *bytes* que nuestro script va a permitir subir. Esto debe controlarse en la Web que reciba los datos mediante PHP. Puede ver el aspecto gráfico en la figura 10.4.

Cuando el formulario se envía, PHP detecta automáticamente que hemos enviado un fichero y lo almacena en un directorio temporal del servidor. La Web que recibe los datos introduce dentro de la variable súper-global 3_FILES [ "nombre_fichero" ] un conjunto de propiedades tales como el nombre del fichero subido, el nombre temporal del recurso, el tamaño o si ha tenido algún error la subida. Vamos a ampliar el formulario con código para el manejo de ficheros:

```
<HTML>
<BODY>
<form action="subir.php" method="POST" enctype="multipart/form-data">
```

```
Introduce el fichero: <input type="file" name="fichero"><br>
 <input type="submit">
 </form>
 <?PHP
$fichero = $_FILES["fichero"];
foreach ($_FILES["fichero"] as $indice => $valor) {
         echo "$indice: $valor<br>";
}
if ($_POST["TAMANIO"]<$_FILES["fichero"] ["size"]) {
         echo "<br>El fichero tiene un tamaño adecuado";
         copy ($_FILES["fichero"] ["tmp_name"],"archivo.txt");
} else {
         echo "<br>El fichero es demasiado grande";
}
?>
</BODY>
</HTML>
```

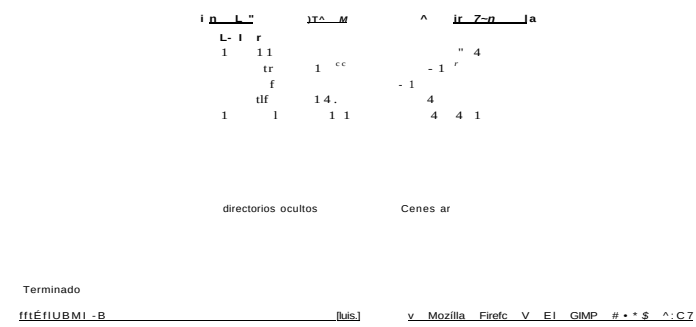



**Figura 10.4.** Formulario de entrada de ficheros.

El resultado muestra un formulario donde debe introducir un fichero, que puede ver en la figura 10.5.

Cuando activa el botón de subida, la página vuelve a cargar, pero esta vez muestra información sobre el fichero y si el tamaño es adecuado o no para su almacenamiento.

Nos hemos apoyado en la construcción `foreach` para averiguar las propiedades del fichero subido.

Si el tamaño en bytes que tenemos puesto como límite en el formulario es menor que el tamaño del archivo, se utiliza la función copy () para copiar el fichero y guardarlo en el servidor.


**Figura 10.5.** Formulario de entrada de ficheros.

## Descarga de ficheros

Normalmente dejamos al lenguaje HTML el trabajo de enlazar los archivos que deseamos que los usuarios de nuestra Web puedan descargar. Esto se puede hacer simplemente con un enlace del tipo:

```
<html>
<BODY>
<a href="ejemplo.zip">Pincha aquí para descargar</a>
</BODY>
</HTML>
```

Cuando hacemos clic en el enlace, normalmente el navegador averigua el tipo de fichero que vamos a descargar (ZIP, PDF, imagen) y nos muestra una pequeña ventana con información sobre el fichero.

Imagine ahora que los ficheros que queremos poner en descarga se generan en ese mismo instante, como los archivos PDF. Tenemos que seguir una pauta distinta para permitir la descarga desde el sitio Web, utilizando cabeceras HTTP. La figura 10.6 muestra la ventana del navegador que recibe la cabecera MIME.

```
<?php
$fichero = "ejemplo.zip";
$mimeType = "applícation/zip";
//Si el navegador es Explorer u Opera cambiamos el tipo de
cabecera
if (strpos($_SERVER['HTTP_USER_AGENT'],'MSIE 5') ||[
    strpos($_SERVER['HTTP_USER_AGENT'],'Opera 7')) {
           $mimeType = "application/x-download";
}
//Se genera la cabecera
header("content-disposition: attachment;
filename=ejemplo__personal.zip");
header("content-type: " . $mimeType);
header("content-length: " . filesize($fichero));
//Se envía el archivo al navegador
readfile($fichero);
?>
```

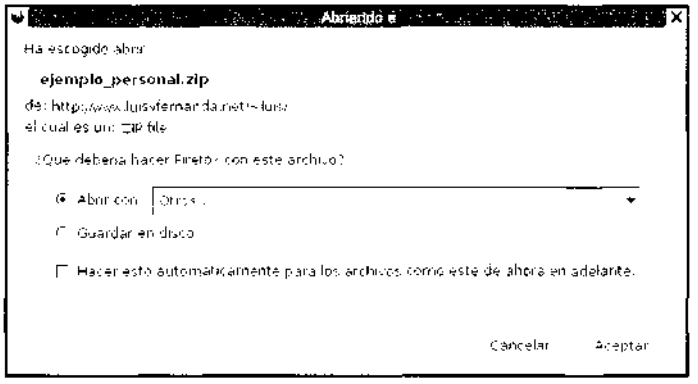



**Figura 10.6.** Descarga de un fichero ZIP.

Los tipo MIME identifican el tipo de fichero y lo asocian a una aplicación determinada en un sistema operativo determinado. Si utiliza el tipo MIME de PDF en un sistema Windows, la aplicación asociada puede ser Acrobat Reader de Adobe. Casi todos los archivos llevan asociado un tipo MIME y sirve, entre otras cosas, para que el navegador reconozca el tipo de archivo que está descargando y optar por mostrarlo directamente en pantalla o guardarlo en alguna ubicación del disco duro. Si comprueba antes el nombre del navegador que ha solicitado la descarga, puede cambiar el tipo para que sea compatible con distintas versiones. Las cabeceras HTTP deben generarse con la función `header()` y debe ser lo primero que se envía al navegador. Si envía algún dato o etiqueta antes que la cabecera obtendrá un error.

La cabecera que utilizamos en este caso es la de envío de un fichero del tipo MIME indicado y tamaño calculado con la función `filesize()`. No es necesario que el archivo que vaya a enviar tenga el mismo nombre en la descripción de la cabecera que en el servidor.

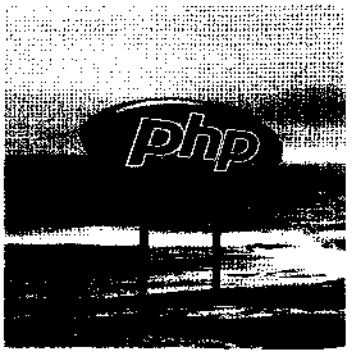
Finalmente, la función `readfile()` lee el fichero completamente y lo envía al navegador.

## Resumen

Situados ya en el ecuador del libro, podemos asegurar que está preparado para enfrentarse a programas serios. Los conceptos básicos los aprendió en los capítulos 1, 2, 3, 4, 5, 6 y 7. El capítulo 8 le mostró la forma de utilizar las variables entre varias páginas Web y el 9 la verdadera potencia de uso de PHP 5.

Este capítulo le ha mostrado la forma de guardar datos para que permanezcan almacenados indefinidamente y métodos para abrir, cerrar, copiar o borrar los ficheros que utilice. Aunque no es el mejor método para guardar los datos, muchas aplicaciones existentes se sirven de las funciones de archivos en vez de la utilización de bases de datos.

Los dos capítulos siguientes le guiarán por el apasionante mundo de los gestores de bases de datos. La potencia de estos gestores le permitirá erradicar de sus *scripts* la gestión de datos mediante ficheros. A partir de entonces la gestión de archivos quedará relegada al manejo de ficheros gráficos, copias o gestión de la configuración.




# Capítulo 11

# Bases de datos con SQL y SQLite

**En este capítulo aprenderá a:**

- Crear sus propias bases de datos con el lenguaje SQL.
- Realizar consultas con SQL para insertar, modificar o borrar registros.
- Conocer las funciones básicas de SQLite.
- Recuperar los datos de una tabla y el número de registros.
- Moverse entre los registros de una base de datos.

# Introducción

Este capítulo pretende ser una introducción al lenguaje estructurado de consulta (SQL), que es primordial conocer a la hora de utilizar alguna base de datos relacional. Puesto que no es un libro dedicado a SQL, sólo veremos las estructuras básicas del lenguaje para poder realizar nuestros ejemplos, por eso, recomiendo leer algún libro deciicado a consultas SQL para sacar el máximo partido a las bases de datos.

También conoceremos la nueva implementación de PHP 5 para bases de datos *SQLite.* Lejos de ser un gestor de bases de datos es, sencillamente, *una librería que nos permite tratar un fichero de ¡a misma forma que utilizamos* una base de datos. SQLite viene instalado en los binarios de PHP 5 y no utiliza ningún puerto de conexión externo, tan sólo funciona cuando se hace una llamada a las funciones de SQLite.

# SQL

La estructura básica de una base de datos SQL es muy sencilla. Los datos de una base de datos se almacenan en tablas, donde cada fila identifica unívocamente a un elemento distinto. Por ejemplo, la tabla de usuarios tendrá tantas filas como usuarios tenga la página Web. Cada tabla puede definir varias columnas. Las columnas hacen referencia a las propiedades de cada fila. Por ejemplo, la tabla de usuarios podría tener como columnas el código de usuario, el nombre, los apellidos y la dirección postal.

**Nota:**

*Hoy en día, tedas las aplicaciones profesionales escritas en PHP se basan en la interacción entre el código y los datos almacenados en alguna base de datos. Recomiendo al lector que no escatime esfuerzos en aprender el lenguaje SQL, pues le será muy útil para realizar después aplicaciones basadas en cualquier base de datos: MySQL, SQLite, SQL Server o, incluso, Oracle.*

El lenguaje SQL nos permite elegir varias filas atendiendo a un criterio, insertar nuevas filas, actualizar los datos o borrarlas, con las sentencias SELECT, INSERT, UPDATE y DELETE respectivamente. Este lenguaje per-

mite sólo manipular los datos contenidos en las tablas. Si quiere modificar la estructura tiene que apoyarse en el lenguaje de definición de datos que le permite crear nuevas tablas o bases de datos, insertar nuevas propiedades en forma de columnas o borrar tablas, con las sentencias CRÉATE, ALTER v DROP respectivamente. Para seguir los ejemplos, vamos a basarnos en una pequeña base de datos con tres tablas.

**Tabla** 11.1. Usuario.

| id_usuario | nombre | cuenta |
|---|---|---|
| 1 | Luis Miguel | 7011 |
| 2 | María Fernanda | 3454 |
| 3 | Pedro | 3445 |
| 4 | Javier | 1123 |

La tabla de Usuarios tiene 4 filas. Cada una de ellas es diferente a las demás y el parámetro que las identifica unívocamente es id_usuario. En España, todas las personas tienen un código numérico, el DNI, que las identifica y las hace distintas frente a otras personas que tengan el mismo nombre y apellidos. En las tablas ocurre lo mismo. Necesitamos un código numérico que nos identifique personas distintas. Las dos propiedades siguientes son el nombre y la cuenta del banco para hacer los pagos.

**Tabla 11.2.** Producto.

| id_producto | nombre | precio |
|---|---|---|
| 1 | ratón | 6 |
| 2 | portátil | 1000 |
| 3 | libro PHP5 | 20 |

La tabla de Productos es muy similar a la de Usuarios e identifica todos los productos que se pueden encontrar en nuestra Web de venta.

**Tabla 11.3.** Carrito.

| id_compra | id_usuario | Jd_producto |
|---|---|---|
| 1 | 1 | 1 |

| idcompra | ¡dusuario | id_producto |
|---|---|---|
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 3 | 3 |

La tabla Carrito de la compra mantiene un listado que relaciona los usuarios con los productos que han comprado. Si se fija en la primera fila, el id_compra es el número de factura, que debe ser distinto siempre en el año actual, el id_usuario es 1, lo que quiere decir que el usuario 1, Luis Miguel, ha comprado el producto 1, un ratón de PC. Si ha entendido esto ya podemos empezar a ver las sentencias SQL para manejar los datos.

## SELECT

Este es el comando principal que usará en todas sus aplicaciones de bases de datos. La sintaxis básica es la siguiente:

```
SELECT campol, campo2, campo3
FROM tabla'
WHERE condición;
```

La sentencia SELECT permite escoger los campos que especifiquemos de un conjunto de filas que cumplan una condición escrita después de la cláusula WHERE. Por ejemplo, para recuperar todos los nombres de los usuarios almacenados en la tabla Usuario tendríamos que hacer:

```
SELECT nombre
FROM Usuario;
```

El resultado es una columna con todos los nombres.

```
Luis Miguel
Maria Fernanda
Pedro
Javier
```

**Advertencia:**

*Si se fija, ¡a cláusula* WHERE *no aparece, porque estamos seleccionando todas ¡as filas. Si quisiéramos seleccionar todas las filas cuyo* id_usuario *sea mayor que 2 tendríamos que añadir* WHERE id_usuario > 2.

Si necesita utilizar todos los campos de una tabla, puede utilizar el símbolo * para seleccionar todos.

```
SELECT *
FROM Usuario;
```

Un último truco permite conocer el número de filas que cumplen la condición. Si después de la sentencia SELECT escribimos count(*), el gestor de bases de datos nos contará el número de filas implicadas en la selección.

```
SELECT count)*)
FROM Usuario;
```

El resultado devuelto es 4, el número de usuarios dados de alta.

### Uniones

Imagine ahora que queremos conocer el nombre de los usuarios que han comprado un producto determinado. Si nos fijamos en la tabla Carrito sólo podemos sacar el id_usuario con lo aprendido hasta ahora.

Existe la posibilidad de cruzar varias tablas. Cuando unimos dos tablas, lo que realmente pasa es que obtenemos como resultado una tabla mayor con la suma del número de columnas de las tablas y tantas filas como indique la multiplicación de las filas de las tablas implicadas.

Si hacemos algo así:

```
SELECT *
FROM Usuario, Carrito;
```

El resultado es la unión de las tablas Usuario y Carrito, cuyo resultado tiene la suma de las columnas (3 + 3) y la multiplicación de las filas (4 x 4):

**Tabla 11.4.** UsuarioxCarrito.

| idusuario | nombre | cuenta | id_carrito | id_usuario | idproducto |
|---|---|---|---|---|---|
| 1 | Luis Miguel | 7011 | 1 | 1 | 1 |
| 1 | Luis Miguel | 7011 | 2 | 1 | 2 |
| 1 | Luis Miguel | 7011 | 3 | 2 | 1 |
| 1 | Luis Miguel | 7011 | 4 | 3 | 3 |
| 2 | Mª Fernanda | 3454 | 1 | 1 | 1 |
| 2 | Mª Fernanda | 3454 | 2 | 1 | 2 |
| 2 | Mª Fernanda | 3454 | 3 | 2 | 1 |
| 2 | Mª Fernanda | 3454 | 4 | 3 | 3 |

| idusuario | nombre | cuenta | id_carrito | id_usuario | id_producto |
|---|---|---|---|---|---|
| 3 | Pedro | 3445 | 1 | 1 | 1 |
| 3 | Pedro | 3445 | 2 | 1 | 2 |
| 3 | Pedro | 3445 | 3 | 2 | 1 |
| 3 | Pedro | 3445 | 4 | 3 | 3 |
| 4 | Javier | 1123 | 1 | 1 | 1 |
| 4 | Javier | 1123 | 2 | 1 | 2 |
| 4 | Javier | 1123 | 3 | 2 | 1 |
| 4 | Javier | 1123 | 4 | 3 | 3 |

Aquí es donde realmente se necesita utilizar la cláusula WHERE. Lo que queremos conocer es el nombre de los usuarios que han comprado el producto número 1. Al hacer una unión, lo primero que tenemos que hacer es eliminar las filas repetidas mediante una sentencia que relacione las dos tablas. Si se fija bien, puede ver que la columna id_usuario aparece en las dos tablas, por lo tanto es una buena referencia. Observe la siguiente sentencia SQL:

```
SELECT *
FROM Usuario,  Carrito
WHERE Usuario . id_usuario = Carrito . id__usuario ;
```

Lo que hemos hecho es crear una unión de dos tablas, donde cada fila debe tener las dos columnas id_usuario iguales.

Como las dos tablas tienen el mismo nombre para la columna id_usuario, es necesario anteponer la tabla a la propiedad para que la consulta funcione, de la siguiente forma: Usuario.id_usuario o Carrito.id_usuario.

El resultado de la sentencia anterior es:

Tabla **11.5.** UsuarioxCarrito.

| idusuario | nombre | cuenta | id_carrito | idusuario | id_producto |
|---|---|---|---|---|---|
| 1 | Luis Miguel | 7011 | 1 | 1 | 1 |
| 1 | Luis Miguel | 7011 | 2 | 1 | 2 |
| 2 | M$^{a}$ Fernanda | 3454 | 3 | 2 | 1 |
| 3 | Pedro | 3445 | 4 | 3 | 3 |

Bien, ahora tenemos un conjunto de datos que nos da el nombre de los usuarios que han comprado algo en nuestra tienda. Si queremos averi-

guar quién ha comprado el producto número 1 tendremos que modificar la sentencia.

```
SELECT *
FROM Usuario, Carrito
WHERE Usuario.id_usuario - Carrito.id_usuario
AND id_producto = 1;
```

El resultado ahora se acerca más a la realidad de lo que necesitamos:

**Tabla 11.6.** UsuarioxCarrito.

| id_usuario | nombre | cuenta | id_carrito | id_usuario | id_producto |
|---|---|---|---|---|---|
| 1 | Luis Miguel | 7011 | 1 | 1 | 1 |
| 2 | Mª Fernanda | 3454 | 3 | 2 | 1 |

Acotando un poquito más la sentencia:

```
SELECT nombre
FROM Usuario, Carrito
WHERE Usuario.id_usuario = Carrito.id_usuario
AND id_producto = 1;
```

Ahora el resultado sí es el esperado:

**Tabla 11.7.** Nombre de los usuarios que han comprado el producto 1.

| nombre |
|---|
| Luis Miguel |
| Mª Fernanda |

Dominar la sentencia SELECT con todas sus variaciones es un trabajo duro de varios meses o años, pero la recompensa se recoge en los primeros pasos, cuando comenzamos a ver la potencia del lenguaje y el uso en los distintos gestores de bases de datos.

## INSERT

Para poder seleccionar los datos, primero hay que poder insertarlos en las tablas. La sentencia necesaria para introducir nuevos datos es:

```
INSERT INTO tabla (columnal, columna2, columna3...)
VALÚES (valorl, valor2, valor3...);
```

El resultado de esta operación es la inclusión de un registro nuevo en la tabla en las columnas indicadas y con los valores dados. Aquellas columnas que no aparezcan, no tomarán valor en ese registro, o tomarán el valor NULL. Para insertar un nuevo cliente en la tabla de Usuario puede utilizar la sentencia:

```
INSERT INTO Usuario (id_usuario, nombre, cuenta)
VALÚES (5, "Feo. José", 7898);
```

La tabla resultante quedará de esta forma:

**Tabla 11.8.** Usuario.

| idusuario | nombre | cuenta |
|---|---|---|
| 1 | Luis Miguel | 7011 |
| 2 | Mª Fernanda | 3454 |
| 3 | Pedro | 3445 |
| 4 | Javier | 1123 |
| 5 | Feo. José | 7898 |

## UPDATE

Se utiliza para editar información que ya está almacenada en la base de datos. En otras palabras, podemos cambiar selectivamente cierta información, sin tener que borrar un registro e insertarlo de nuevo.

La sintaxis básica es:

```
UPDATE tabla
SET carapol = valorl, campo2 = valor2, campo3 = valor3
WHERE condición;
```

La condición se utiliza para seleccionar el número de filas a las que afectará el cambio. Si no ponemos nada se supone que la actualización de datos se hace sobre todas las filas de una tabla, por eso es necesario poner alguna condición.

Podemos utilizar UPDATE para cambiar el número de cuenta de uno de los usuarios:

```
UPDATE Usuario
SET cuenta = 4455
WHERE id_usuario = 2;
```

Lo que hemos hecho es cambiar el número de cuenta de María Fernanda de 3454 a 4455.

### DELETE

És la sentencia más peligrosa de las vistas. Permite borrar de uno a varios registros de una tabla.
La sintaxis es:

```
DELETE FROM tabla
WHERE condición;
```

*Si* se omite la cláusula WHERE se borran todas las filas de una tabla.
Es aconsejable no precipitarse en el uso de esta opción y detenerse antes un momento a recapacitar si se necesita realmente dejar la tabla vacía.
La fila que contiene al usuario Javier puede borrarse de la siguiente forma:

```
DELETE FROM Usuario
WHERE nombre="Javier";
```

## Definición de tablas

El primer paso para poder interactuar con los datos de una tabla es su creación.

Para este fin se utiliza la sentencia CRÉATE, que define los campos y el tipo de datos que se va a almacenar en cada columna.

Ll formato simplificado es:

```
CRÉATE TABLE tabla (columnal tipo_dato, columna2 tipo_dato,
columna3 tipo_dato);
```

Los tipos de datos de SQL estándar pueden variar dependiendo del gestor de bases de datos que utilice, pero básicamente pueden ser:

**Tabla 11.9.** Tipos de datos en SQL 92.

| Especificación | Tipo de dato |
|---|---|
| INT, INTEGER | Entero grande |
| SMALLINT | Entero pequeño |
| REAL, FLOAT | Coma flotante |
| DEC, DECIMAL | Decimal |
| CHAR(n), CHARACTER(n) | Alfanumérico de longitud fija n |

| Especificación | Tipo de dato |
|---|---|
| VARCHAR(n) | Alfanumérico de longitud variable, como máximo n caracteres |
| DATE | Fecha |
| TIME | Hora |
| TIMESTAMP | Instante. Conjunto de Fecha y Hora de un momento determinado |

Con estos datos podemos ver la definición de la tabla de Usuario.

```
CRÉATE TABLE Usuario (id_usuario INTEGER, nombre CHAR(255),
cuenta INTEGER);
```

Si alguno de los campos es obligatorio, es decir, si siempre que realice una inserción de nuevos datos necesita que ciertos campos sean introducidos obligatoriamente, puede utilizar la sentencia NOT NULL después de la definición del campo.

El campo nombre es obligatorio en la tabla Usuario, ya que un registro de usuarios sin almacenar su nombre no tiene sentido.

```
CRÉATE TABLE Usuario (id_usuario INTEGER, nombre CHAR(255) NOT
NULL, cuenta INTEGER NOT NULL);
```

El id_usuario identifica un registro y sólo uno. No puede haber en la misma tabla dos números iguales en esta columna. La forma de encontrar rápidamente una fila determinada es preguntar por este dato.

La clave primaria de una tabla es un identificador único en cada tabla y que puede utilizarse para relacionar una tabla con otra. En nuestro ejemplo podemos ver claramente que id_usuario es la clave primaria de la tabla Usuario.

```
CRÉATE TABLE Usuario (id_usuario INTEGER PRIMARY KEY, nombre
CHAR(255) NOT NULL, cuenta INTEGER NOT NULL);
```

# SQLite

SQLite es una librería que implementa un conjunto de sentencias bastante amplio de SQL 92 estándar. La librería es muy pequeña y 2 ó 3 veces más rápida que los gestores de bases de datos MySQL o PostgreSQL. Necesita muy poco tiempo de ejecución y muy poca memoria para el proceso. No se necesita administrar la base de datos fuera del entorno de

programación, ya que no es un servidor. Existe un problema derivado de esto.

SQLite se basa en el almacenamiento de datos en un fichero, por lo tanto, cada vez que se almacenan datos, el fichero que contiene los registros se bloquea durante el tiempo de la actualización y no permite a otros usuarios interactuar. Si la base de datos se utiliza solamente para leer, la velocidad será muy elevada, pero si la utilizamos para actualizar constantemente los datos, SQLite perderá más tiempo negociando los bloqueos del fichero que dando servicio.

Entre las características de SQLite tenemos:

- Implementa muchas funcionalidades de SQL 92, incluido disparadores y transacciones.
- Protección de integridad de datos con cominit y `rollback`.
- Los ficheros de datos pueden moverse de servidor y seguirán funcionando.
- Soporta bases de datos de 2 *Terabytcs.*
- El código fuente es de dominio público.

## Creación de bases de datos

La *interfnce* de programación de la extensión SQLite es muy similar al gestor de bases de datos MySQL y PostgeSQL. La diferencia principal es que SQLite, aunque acepta la definición de datos con la sentencia CRÉATE, internamente sólo diferencia entre valores alfanuméricos o valores numéricos, es decir, que podemos omitir el tipo de los campos en la definición de las tablas.

La función `sqlite_open ()` recibe como parámetro el nombre de una base datos. Si la base de datos no existe la crea. Además de esta función, necesitaremos otra para poder enviar consultas a la base de datos. La función `sqlite _single_query ()` toma como parámetros la base de datos creada con `sqlite_open ()` y un literal con la sentencia SQL que queremos ejecutar.

El ejemplo siguiente crea la base de datos de usuario e inserta las filas que estamos utilizando para aprender el lenguaje SQL:

```
<?php
$base_datos = sqlite_single_open("Usuario.db");
$consulta =  "CRÉATE TABLE Usuario
          (id_usuario PRIMARY KEY,
```

```
          nombre CHARÍ255) NOT NULL,
          cuenta INTEGER NOT NULL)";
sqlite__query($base_datos,$consulta);
$consultal = "INSERT INTO Usuario (id_usuario, nombre, cuenta)
VALÚES (1, \"Luis Miguel\",7011)";
$consulta2 = "INSERT INTO Usuario (id_usuario, nombre, cuenta)
VALÚES (2, \"María Fernanda\",3454)";
$consulta3 = "INSERT INTO Usuario (id_usuario, nombre, cuenta)
VALÚES (3, \"Pedro\",3445)";
$consulta4 = "INSERT INTO Usuario (id_usuario, nombre, cuenta)
VALÚES (4, \"Javier\",1123)";
sqlite_single_query($base_datos,$consultal);
sqlite_ single_query($base_datos,$consulta2);
sqlite_ single_query($base_datos,$consulta3);
sqlite_ single_query($base_datos,$consulta4);
?>
```

De esta forma se ha creado la tabla Usuario, pero hemos necesitado hacer 4 consultas adicionales para añadir los datos. Esto se puede simplificar utilizando la función sqlite_query (), que acepta más de una línea de SQL.

Vamos a crear de esta forma la tabla Producto y Carrito:

```
<?php
$consulta = "CRÉATE TABLE Producto
(Íd_producto INTEGER PRIMARY KEY,
nombre CHAR(255) NOT NULL,
precio INTEGER NOT NULL);
INSERT INTO Producto (id_producto, nombre, precio)
VALÚES (1, \"Ratón\",6);
INSERT INTO Producto (idjroducto, nombre, precio)
VALÚES (2, \"Portátil\",1000);
INSERT INTO Producto (id_producto, nombre, precio)
VALÚES (3, \"Libro PHP5\",20)";
sqlite_query($base_datos,$consulta);
{consulta = "CRÉATE TABLE Carrito
(id^compra INTEGER PRIMARY KEY,
id^usuario INTEGER NOT NULL,
id^producto INTEGER NOT NULL);
INSERT INTO Carrito (id^compra, id_usuario, id_producto)
VALÚES (1, 1, 1);
INSERT INTO Carrito (id^compra, id_usuario, id_producto)
VALÚES (2, 1, 2);
INSERT INTO Carrito (id_compra, id_usuario, id_producto)
VALÚES (3, 2, 1);
INSERT INTO Carrito (id_compra, id_usuario, idjroducto)
VALÚES (4, 3, 3)";
sqlite_query($base_datos,$consulta);
?>
```

Algunos gestores de bases de datos necesitan añadir una sentencia específica para que uno de los campos sea auto incremental, es decir, que su valor aumente poco a poco sin necesidad de añadirlo nosotros; en realidad es gestionado por la propia base de datos.

Los campos enteros que llevan asociada la sentencia PRIMARY KEY automáticamente son tratados como campos auto increméntales y, en las órdenes INSERT se puede omitir la inclusión del valor, porque SQLite le dará el valor correlativo de la fila que le corresponda.

## Últimos cambios en una tabla

Hay veces que necesitamos saber el último valor auto incremental que SQLite ha generado para asociar una imagen, o un archivo, a un registro de una tabla. Para averiguar este dato podemos usar la función `sqlite_last_insert_rowid()`.

```
<?php
$base_datos = sqlite_open("Usuario.db");
sqlite_single_query($base_datos,"INSERT INTO Usuario (nombre,
cuenta) VALÚES (\"José A.\",2677)");
$ultimo_usuario = sqlite_last_insert_rowid($base_datos);
sqlite_single_query($base_datos,"INSERT INTO Producto (nombre,
precio) VALÚES (\"Ibook\",1200)");
$ultimo_producto = sqlite_last_insert__rowid($base_datos);
sqlite_single_query($base_datos,"INSERT INTO Carrito
(id_usuario, id_producto) VALÚES
(\"$id_usuario\",\"$id_producto\")");
?>
```

En el ejemplo puede ver la posible utilización de la función. Con ella extraerá el último usuario y el último producto introducido y aprovechará para insertarlos en la tabla de compras.

Otra función muy útil es `sqlite__changes ()`, que devuelve el número de filas afectadas por una consulta anterior.

Si borra varias filas, esta función le devolverá el número de filas que han sido borradas, como puede ver en la figura 11.1.

```
<?php
$base_datos = sqlite_open("Usuario.db");
sqlite_single_query($base_datos,"UPDATE Usuario set cuenta =
2222");
echo "Número de filas afectadas: " .
sqlite_changes($base_datos);
?>
```

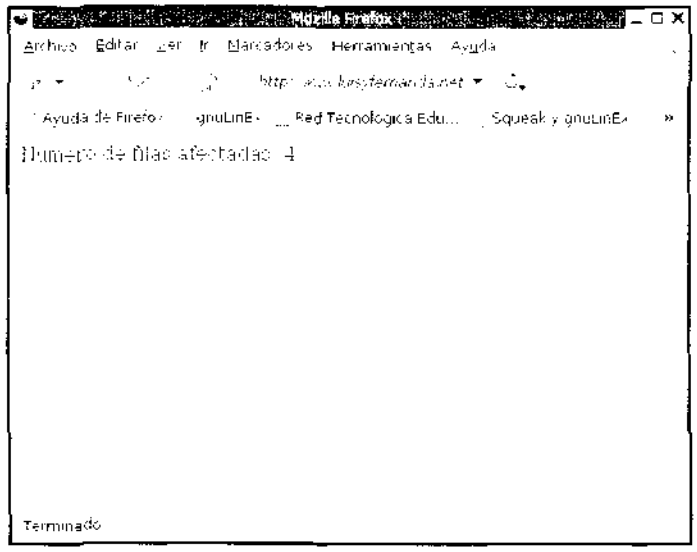



**Figura 11.1.** Número de filas afectadas por una consulta.

## Selección de datos

Como hemos visto antes, la función `sqlite_query()` da la posibilidad de hacer cualquier tipo de consulta vista al principio del capítulo.

Las sentencias CRÉATE, INSERT, DELETE o UPDATE no necesitan devolver ningún dato, pero la sentencia SELECT tiene que devolver los datos requeridos.

La variable que obtiene los datos se considera como un recurso y debe aplicarse una función para recuperar las filas una a una.

Lo podemos comprobar con un ejemplo, cuyo resultado se muestra en la figura 11.2.

```
<?php
$base_datos = sqlite_open("Usuario.db");
^resultado = sqlite_query($base_datos,"SELECT * FROM Usuario");
if ( ! $resultado) {
        echo "Parece que hay un error";
} else {
        while ($fila = sqlite_fetch__array($resultaáo)) {
                foreach ($fila as $indice => $valor) {
                        echo "$indice: $valor<br>";
                }
        }
}
?>
```

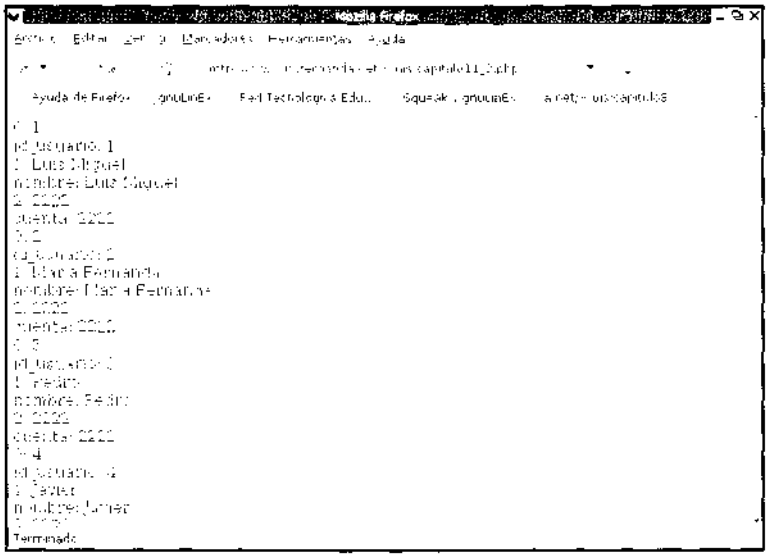


**Figura 11.2.** Recorrido de la tabla de Usuario con SQLite.

Después de hacer una consulta con la función `sqlite_query()`, la variable `$resultado` recibe un tipo de dato que no es manejable. La función `sqlite_fetch_array()` extrae de la variable `$resultado` todas las filas devueltas en forma de *array.* Para ello hemos construido un bucle `while` donde la variable `$fila` va recibiendo los registros.

Para acceder a cada columna de un registro simplemente tenemos que hacer uso de la variable como si fuera un *array* y como índice el nombre de la columna o el número de registro de esta forma: `$fila["nombre"]`, `$fila["cuenta"]`, `$fila[0]`; `$fila[l]`.

Dentro del bucle hemos utilizado otro bucle `foreach` para mostrar por pantalla todos los valores del *array.*

## SQLite orientado a objetos

Puesto que la potencia de PHP 5 radica en la fuerte orientación a objetos, es de agradecer que SQLite provea algún objeto para la interacción con bases de datos. SQLite tiene una potente *interface* orientada a objetos que puede ser usado para un eficaz manejo de los datos, librando al programador de crear su propio objeto con las funciones procedurales. El objeto se llama `SQLiteDatabase()` y el constructor recibe como parámetro la base de datos.

El ejemplo muestra cómo utilizarlo, aunque el lector ya conocerá el funcionamiento básico de los objetos:

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "CRÉATE TABLE Usuario
       (id^usuario INTEGER PRIMARY KEY,
       nombre CHAR(255) NOT NULL,
       cuenta INTEGER NOT NULL);
       INSERT INTO Usuario (id_usuario, nombre, cuenta)
       VALÚES (1, V'Luis Miguel\",7 011) ;
       INSERT INTO Usuario (id_usuario, nombre, cuenta)
       VALÚES (2, V'María Fernanda\",3454);
       INSERT INTO Usuario (id_usuario, nombre, cuenta)
       VALÚES (3, \"Pedro\",3445);
       INSERT INTO Usuario (id_usuario, nombre, cuenta)
       VALÚES (4, \"Javier\",1123);";
$base_datos->query($consulta);
?>
```

El método `query()` funciona exactamente igual que la función `sqlite_query()`, pero sólo recibe como parámetro la consulta, ya que la base de datos va implícita en el objeto.

## Selección de registros

Las consultas SELECT se tratan de manera parecida a lo visto anteriormente. Puesto que estamos utilizando un objeto, para recibir el resultado de una consulta tendremos que utilizar métodos que nos permitan manejarlos. En la figura 11.3 puede ver el resultado.

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "SELECT * FROM Usuario";
$resultado = $base_datos->query($consulta);
while ($resultado->valid()) {
       $fila = $resultado->current();
       echo $fila[0] . "<br>";
       echo $fila[l] . "<br>";
       echo $fila[2] . "<br>";
       $resultado->next(),•
}
?>
```

El método `valid()` da un resultado *true* si existen valores que no se han mostrado; si devuelve *false* es que ya hemos extraído todas las filas. El método `current()` nos da la fila actual déla consulta. La variable `$fila`

obtiene un *array* con los datos del registro actual. Para avanzar de registro en la consulta se utiliza el método next ().

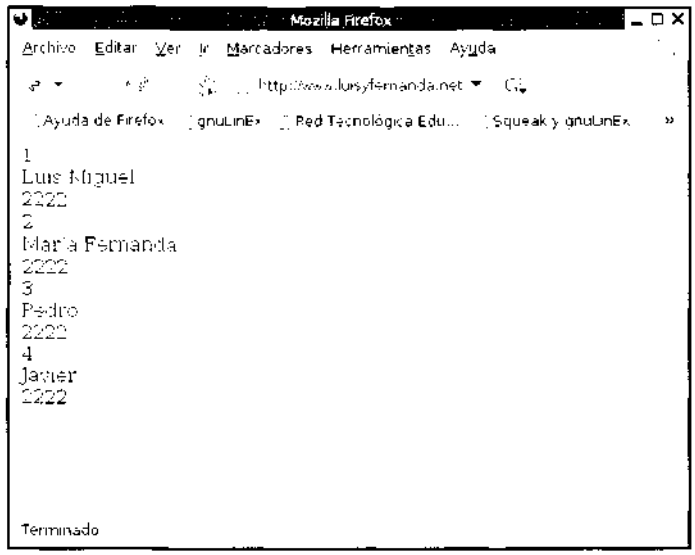



**Figura 11.3.** Recorrido de la tabla Usuario con next().

## Funciones de Array para recuperar datos

SQLite ofrece también un número de nuevas funciones que simplifican y aceleran la recogida de información de la base de datos. Por ejemplo, el método ArrayQuery () devuelve un array de dos dimensiones, donde la primera dimensión indica el número de la fila y la segunda dimensión el número de columna de esa fila. Si la variable $resultado recoge el resultado de ArrayQuery (), puede acceder a los registros de la siguiente forma: $resultado[0] [0], $resultado[l] [0], $resultado [2] [3] . Puede utilizar algún bucle *foreach* para sacar todos los datos.

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "SELECT * FROM Usuario";
$resultado = $base_datos->ArrayQuery($consulta);
$filas = 0;
while (isset ($resultado [$filas] ) ) {
        foreach ($resultado[$filas] as $indice => $valor) {
                echo "$indice: $valor<br>";
        }
        $filas++;
```

```
}
?>
```

Además de esta forma de trabajar, puede hacer que el resultado de la consulta se devuelva en forma de objeto. PHP 5 permite utilizar la construcción *foreach* para interactuar con objetos v recuperar los datos, de la siguiente forma:

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "SELECT * FROM Usuario";
{•resultado = $base_datos->unbufferedQuerv($consulta) ;
foreach ($resultado as $valor) {
         echo "Svalor[0]<br>";
         echo "$vaior[1]<br>" ;
         echo "$valor[2]<br>";
}
?>
```

La construcción unbufferedQuery () devuelve las filas en forma de objeto y, además, no almacena el resultado en buffer por lo que es muy rápida.

## Número de filas

La extensión SQLite posee un método para determinar el número de campos devueltos en una consulta en un resultado. El método numFields () se encarga de devolver este resultado. Para poder utilizarlo debe recuperar los datos con la función unbufferedQuery () y el objeto devuelto será el que le indique el número de campos que almacena.

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "SELECT * FROM Usuario";
$resultado = $base_datos->unbufferedQuery($consulta);
$numero_filas = $resultado->numFields();
echo "El número de filas es: $numero_filas" ;
foreach ($resultado as $valor) {
         echo "$valorf0]<br>" ;
         echo "$valor[1]<br>";
         echo "$valor[2]<br>";
   }
?>
```

## Moverse entre registros

En muchos casos querrá utilizar la menor memoria posible, y se hará indispensable utilizar el método unbufferedQuery (), aunque esto hace perder funcionalidad y puede no resultar la mejor elección.

Por ejemplo, imagine que quiere saber cuántas filas devuelve un resultado.

Usando unbufferedQuery() tenemos que hacer un bucle para contar una a una las filas del resultado. Con la función query() tan sólo hay que llamar al método numRows() para conocer el número de registros implicados en la consulta.

Las consultas con query() permiten, además, recuperar la información en el orden que queramos y no secuencialmente.

El ejemplo siguiente muestra cómo utilizar una consulta para ver los registros empezando desde el último hasta el primero:

```
<?php
$base_datos = new SQLiteDatabase("Usuario.db");
$consulta = "SELECT * FROM Usuario";
$resultado = $base_datos->query($consulta);
$numero_filas = $resultado->numRows();
$resultado->seek($numero_filas-1);
do {
        $fila = $resultado->current();
        foreach ($fila as $valor) {
                 echo "$valor<br>";
        }
} while ($resultado->hasPrev() && $resultado->prev());
?>
```

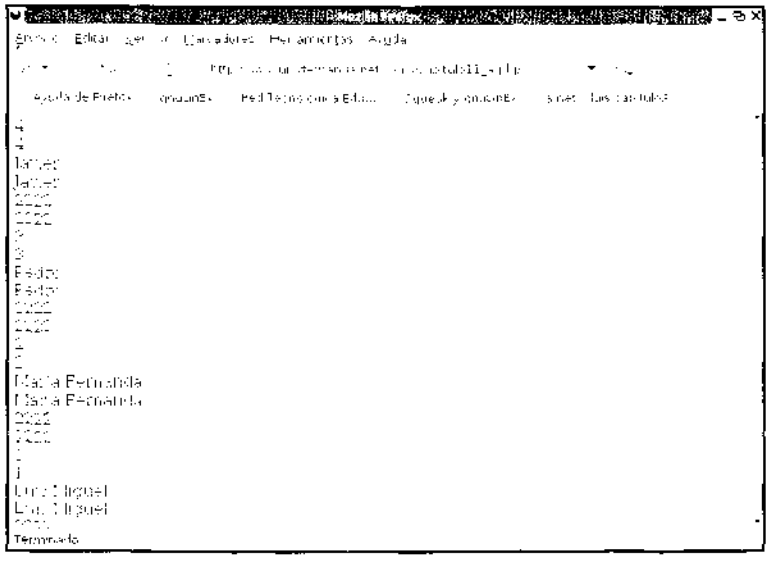


**Figura 11.4.** Recorrido de la tabla Usuario al revés.

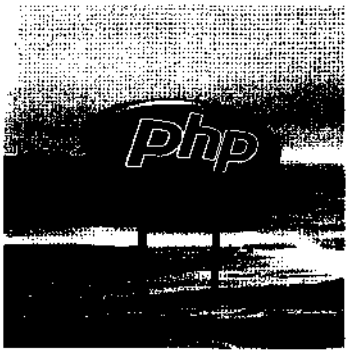
El método `seek()` se mueve a través del resultado y permite posicionarnos en el registro que pasemos como parámetro. El método `hasPrev()` es *true* si existe un registro previo y `prev()` es *true* si ha sido posible moverse a un registro anterior. La figura 11.4 es un fiel reflejo del funcionamiento.

## Resumen

SQLite es una de las características estrella de PHP 5. Se ha incluido para que no necesite instalar un gestor de bases de datos para sus proyectos. Esto hace más sencillo distribuir aplicaciones, porque no dependen de un software adicional, tan sólo de PHP 5.

SQLite es a la vez un servidor de bases de datos y un cliente. La librería permite operar con la mayor parte de las instrucciones de SQL; posee un mayor juego de instrucciones que MySQL 4.

Pero, como todo lo bueno, tiene una pequeña pega. Como SQLite trata las bases de datos como ficheros del sistema operativo, cuando un usuario hace una conexión para escribir datos, el fichero queda bloqueado para inclusiones de otros usuarios. Cuando termina la inserción del primer usuario, el segundo tendrá acceso a escribir los datos. Por lo tanto, SQLite es muy recomendable cuando realice proyectos donde la mayor parte de las veces se hacen lecturas de la base de datos, como un periódico digital o un WebLog. En otro caso es mejor utilizar un gestor de bases de datos como MySQL, que verá en el siguiente capítulo.




# Capítulo 12

# PHP 5 y MySQL

**En este capítulo aprenderá a:**

- Conocer la forma de conectarse a un servidor de bases de datos.
- Realizar consultas y recuperar los datos de distintas formas.
- Recuperar el número de filas que intervienen en una consulta.
- Crear nuevas bases de datos y tablas.

## Introducción

PHP 5 soporta muchos de los gestores de bases de datos relaciónales existentes en el mercado. Las dos alternativas más comunes son ProstgreSQL y MySQL.

Aunque PostgreSQL es mucho mejor en cuanto a características y funciones soportadas del SQL 92 estándar, MySQL se ha hecho más popular en el ambiente de los servidores Web. Cuando los servidores Web ofrecen su servicio como LAMP se refieren a Linux + Apache + MySQL + PHP.

Este capítulo cubre las operaciones más comunes que los desarrolladores de PHP pueden hacer con MySQL, desde recuperar o modificar datos, buscar textos o hacer una copia de seguridad de la base de datos.

## Administración de usuarios

Una gran parte del uso de la seguridad y efectividad de MySQL tiene que ver con comprender el sistema de privilegios. MySQL permite dar permisos al detalle basándose en grupos, usuarios, conexiones y comandos a utilizar. En teoría, se puede dar privilegios a ciertos usuarios para que sólo puedan escribir en algunas columnas de varias tablas. Es una buena idea dar a cada usuario los permisos mínimos necesarios para que pueda interactuar con su base de datos. En realidad, en un mundo ideal, no debería preocuparse por dar permisos a los usuarios o crear bases de datos. Si su trabajo profesional se basa en el desarrollo de aplicaciones, su administrador de bases de datos debería poner a punto el servidor de MySQL para facilitar su trabajo. Si se está interesado en conocer los entresijos del servidor de bases de datos, recomiendo al lector que lea la documentación o;/-*line* de la Web www.mysql.com o consiga uno de los muchos libros que tratan el tema.

En el Apéndice A puede ver cómo instalar MySQL y dar los permisos necesarios para que pueda probar los ejemplos.

## Conexión a MySQL

La conexión no puede ser más sencilla. Es un proceso de dos pasos:

- Se conecta con el servidor de MySQL.

- Se solicita la conexión a una base de datos específica.

Es importante recordar que MySQL es un servidor que puede estar alojado en el mismo ordenador que PHP 5, o en otro diferente. Por eso, la conexión se hace de forma distinta a SQLite.

Para conectar a MySQL es necesario enviar como parámetros la dirección del servidor, el usuario y la contraseña.

```
<?php
$servidor = "localhost";
$usuario = "luis";
$pass = "secreto";
$base_datos = "Compras";
//Conexión al servidor de bases de datos
$descriptor = mysql_connect($servidor,$usuario, $pass) ;
//Se selecciona la base de datos
mysql_select_db($base_datos,$descriptor);
//Se cierra la conexión cuando terminemos
mysql__close( $descriptor);
?>
```

La función mysql_connect () permite conectar a un servidor. En este caso, hemos puesto como servidor a localhost porque estamos utilizando nuestro ordenador local para hacer las pruebas.

Otra cuestión importante es que el servidor MySQL puede almacenar varias bases de datos de un mismo usuario, por eso tenemos que utilizar la funciónmysql_select_db (), para seleccionar la que queremos utilizar.

Por último, tenemos la función mysql_close () que se encarga de cerrar una conexión con el servidor.

El ejemplo siguiente muestra cómo se puede crear una clase reutilizable en todos los proyectos que sirva para conectarse a una base de datos:

```
<?php
class  Servidor_Base_Datos
{
  private  $servidor;
  private  $usuario;
  private  $pass;
  private  $base_datos;
  private  $descriptor;
  function____construct($servidor,$usuario,$pass,$base_datos)
  {
       $this->servidor = $servidor;
       $this->usuario = $usuario;
       $this->pass = $pass;
```

```
        $this->base_datos = $base_datos;
        $this->conectar_base_datos();
    }
    prívate function conectar_base_datos()
    {
        $this->descriptor = mysql__connect($this->servidor,$this-
        >usuario,$this->pass);
        mysql_select_db($this->base_datos,$this->descriptor);
    }
}
$servidor = "localhost";
$usuario = "luis";
$pass = "secreto";
$base_datos = "Usuario";
$usuario = new
Servidor_Base_Datos($servidor,$usuario,$pass,$base_datos);
?>
```

**Nota:**

*No he añadido al objeto ningún tipo de control de errores, ya que veremos ese tema con profundidad en el capítulo 17.*

## Seleccionar datos

Existen diferentes formas de recuperar datos desde MySQL, pero los más usados son seguramente `mysql_fetch_query()` y `mysql_query()`. Si leyó el capítulo anterior sobre SQLite no tendrá ningún problema para asimilar el funcionamiento de estas funciones.

Vamos a añadir dos métodos nuevos a la definición de la clase anterior para que nos pueda servir en otras aplicaciones.

```
public function consulta($consulta)
{
        $this->resultado=mysql_query($consulta,$this->descriptor);
}
public function extraer_registro()
{
        if ($fila = mysql_fetch_array($this->resultado,MYSQL_ASSOC)) •
                return $fila;
        } else {
                return false;
        }
}
```

El método consulta () recibe una sentencia SQL que debe ejecutarse en la base de datos apuntada por el descriptor $thís->descriptor.

El método extraer_registro() comprueba con la función mysql_fetch_array() si existen ficheros en el resultado de la consulta.

En caso afirmativo, devuelve un *arraxj* con los datos de ese registro.

En el capítulo anterior, vimos que un registro pasado *comoarray* permitía acceder a las columnas mediante un índice numérico ($fila[0]) o mediante el nombre de la columna (($fila["id_usuario"])).

La directiva MYSQL_ASSOc elimina la posibilidad de extraer los datos con *un* Índice .numérico, *mejorando la recuperación en* Jos *bucles del foreach.*

El ejemplo siguiente muestra cómo utilizar los nuevos métodos:

```
<?php
//Primero se añade el fichero que contiene la clase
require_once("MySQL.php");
$servidor = "localhost";
$usuario = "luis";
$pass = "secreto";
$base_datos = "Compras";
$usuario = new
Servidor_Base_Datos($servidor,$usuario,$pass,$base_datos);
$usuario->consulta("seslect * from Usuario");
while ($fila = $usuario->extraer_registro()) {
       foreach ($fila as $indice => $valor) {
                echo "$indice: $valor<br>";
       }
}
?>
```

El código anterior recupera la clase del fichero MySQL . php e inicializa las variables necesarias para conectar con la base de datos.

También podemos utilizar un fichero de configuración, como el que vimos en el capítulo 10, para almacenar los datos de la conexión.

La variable $usuario recoge la creación del objeto de conexión y utiliza el método consulta () para recuperar el nombre de los usuarios de nuestra Web.

Para extraer todos los datos nos encontramos con un bucle que va recuperando un *arraxj* por registro, almacenándolo en $fila.

Utilizando foreach podemos extraer todas las columnas de la fila indicada.

La figura 12.1 presenta el resultado del *script.*

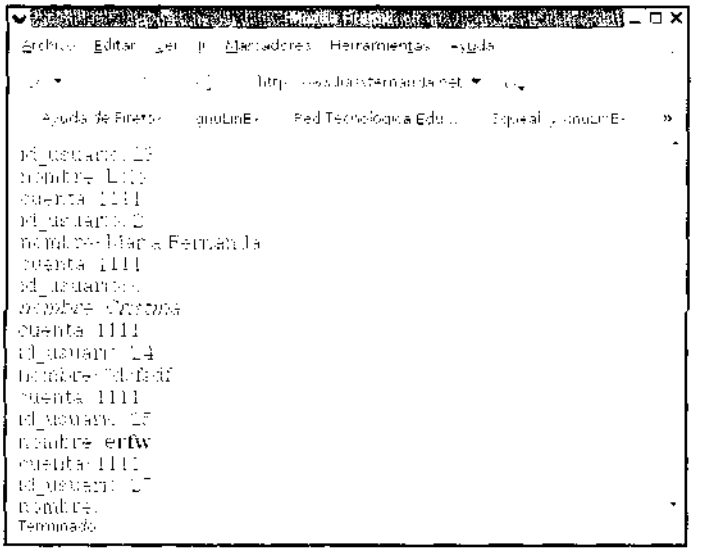



**Figura 12.1.** Número de filas afectadas por una consulta.

# Manipulación de datos

Seleccionar datos de una base de datos es sólo el principio; la potencia de SQL nos permite también insertar, actualizar y borrar filas de una tabla.

Con el objeto que hemos creado es muy sencillo hacer consultas del tipo INSERT, UPDATE o DELETE, llamando al método `consulta ()` .

## Insertar una fila

Para insertar una fila con nuestro objeto, tendremos que definir la consulta y ejecutar el método `consulta ()` .

```
<?php
require_once("MySQL.php");
$servidor = "localhost",-
$usuario = "root";
$pass = "";
$base_datos = "Compras";
$insertar = "INSERT INTO Usuario (nombre,cuenta) VALÚES
(\"Cristina\",2119)";
$usuario = new
Servidor_Base_Datos($servidor,$usuario,$pass,$base_datos);
^usuario->consuIta($insertar);
?>
```

## Actualizar una fila

Antes de actualizar una fila, necesita identificar la fila o filas de la tabla que quiere cambiar. En este ejemplo, se ha utilizado una sentencia SELECT para averiguar primero el valor id_usuario de Luis Miguel y después proceder a cambiar su cuenta bancaria.

```
<?php
require_once("MySQL.php");
$servídor = "localhost";
$usuario = "root";
$pass = "";
$base_datos = "Compras";
$consulta = "SELECT id_usuario FROM Usuario WHERE nombre='Luis
Miguel'";
$usuario = new
Servidor_Base_Datos($servidor,$usuario,$pass,$base_datos);
$usuario->consulta($consulta);
$fila = $usuario->extraer_registro() ;
$id_usuario = $fila["id_usuario"] ;
$actualizar = "UPDATE Usuario SET cuenta=9999 WHERE
id_usuario='$id_usuario'";
$usuario->consulta($actualizar);
?>
```

En un caso práctico, UPDATE puede utilizarse para modificar los datos introducidos en un formulario Web.

## Borrar una fila

Para borrar una fila puede seguir el mismo camino que se ha utilizado antes para actualizar. Primero averigüe el registro que quiere eliminar y después ejecute la consulta con la sentencia DELETE.

```
<?php
require_once("MySQL.php");
$servidor = "localhost";
{usuario = "luis";
$pass = "secreto";
$base_datos = "Compras";
$consulta = "SELECT id_usuario FROM Usuario WHERE nombre='Luis
Miguel'";
$usuario = new
Servidor_Base_Datos{$servidor,$usuario,$pass,$base_datos);
$usuario->consulta($consulta);
$fila = $usuario->extraer_registro() ;
$id_usuario = $fila["id_usuario"] ;
```

```
$borrar = "DELETE FROM Usuario WHERE id_usuario =
[1]$id_usuario'";
$usuario->consulta($borrar);
?>
```

## Errores con las comillas

Examinemos ahora el código siguiente:

```
<?PHP
require_once("MySQL.php");
if (isset($_POST["nombre"]) || isset($_POST["cuenta"])) {
          $servidor = "localhost";
          $usuario = "root";
          $pass = "";
          $base_datos = "Compras";
          $nombre = $_POST["nombre"];
          $cuenta = $_POST["cuenta"];
          $usuario = new Servidor_Base_Datos($servidor,$usuario,
          $pass,$base_datos);
          $insertar = "INSERT INTO Usuario(nombre,cuenta)
          VALÚES ('$nombre','$cuenta')";
          $usuario->consulta($insertar);
}
?>
<HTML>
<BODY>
<form action="formulario_Usuario.php" method="POST">
Nombre: <INPUT type="text" name="nombre"><br>
Cuenta: <INPUT type="text" name="cuenta"><br>
<input type="submit" name="Enviar">
</form>
</BODY>
<HTML>
```

El pequeño formulario que muestra la figura 12.2 permite insertar en la tabla Usuario nuevos clientes.

Los campos de texto que recogen el nombre y la cuenta, pueden recibir cualquier tipo de carácter; hay ciertos símbolos que pueden provocar que un programa falle y son utilizados malintencionadamente por *hackers* para corromper el sistema.

Si en la casilla destinada a escribir el nombre, escribe un nombre y en medio una comilla simple y otra doble tal y como muestra el ejemplo:

```
Luis Migue'1 Cabezas Gra"nado
```

El método consulta () intentará ejecutar:

```
mysql_query("INSERT INTO Usuario(nombre,cuenta) VALÚES (' Luis
Migue'1 Cabezas Gra"nado','')");
```

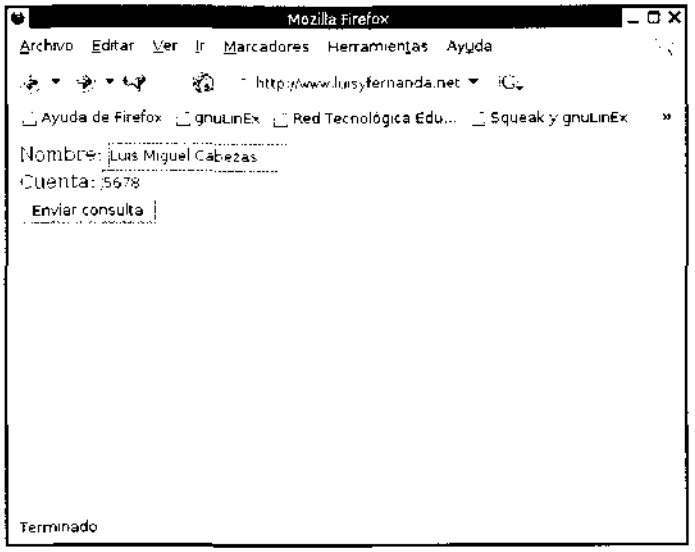



**Figura 12.2.** Formulario de entrada de datos.

¿Puede ver el problema? Cuando MySQL lee la sentencia cree que es:

```
mysql_query("INSERT INTO Usuario(nombre,cuenta) VALÚES (' Luis Migue'
```

El resto de la sentencia causará un error y fallará la consulta. La solución es añadir el símbolo de escape \ delante de la comilla simple para que MySQL tome el símbolo ' como un simple carácter.

```
mysql_query("INSERT INTO Usuario(nombre,cuenta) VALÚES (' Luis
Migue\'l Cabezas Gra\"nado','')");
```

Existen tres soluciones para añadir el símbolo de escape a todas las comillas que aparezcan en campo de texto:

- Lo primero que puede hacer es configurar el archivo php.ini, activando la opción magic_quotes_gpc. Con esta opción, a todos los datos pasados entre páginas Web por los métodos GET y POST, se les inserta automáticamente símbolos de escape en las comillas que aparezcan.
- Si magic_quotes no está activo en su servidor, PHP 5 tiene la función addslashes (), que tiene el mismo cometido que la opción anterior. Si magic_quotes está activo no podrá utilizar esta función, porque se añadirían dos símbolos de escape.

- El tercer camino, muy similar a addslashes, usa la función mysql_escape_string. Esta función añade símbolo de escape a las comillas simples, comillas dobles, caracteres de final de fichero y retorno de carro. Es un método implementado por MySQL, por eso es más recomendable utilizar addslashes.

Para saber si el fichero php.ini tiene activado magic_quotes existe la función get_magic_quotes_gpc ( ) . Puede añadir algunas líneas de código a su formulario para que controle este tipo de error de la siguiente forma:

```
<?php
require_once("MySQL.php");
if (isset($_POST["nombre"]) || isset($_POST["cuenta"])) {
          $servidor = "localhost";
          $usuario = "luis";
          $pass = "secreto";
          $base_datos = "Compras";
          if (get_magic_quotes_gpc()) {
                  $nombre = $_POST["nombre"];
                  $cuenta = $__POST["cuenta"] ;
          } else {
                  $nombre = addslashes($_POST["nombre"]);
                  $cuenta = addslashes($_POST["cuenta"]);
          }
          $usuario = new Servidor_Base_Datos($servidor,$usuario,
          $pass,$base_datos);
          $insertar = "INSERT INTO Usuario(nombre,cuenta) VALÚES
          ('$nombre','$cuenta')";
          $usuario->consulta($insertar);
}
?>
```

# Contando filas

A menudo es útil conocer el número de filas devueltas por una consulta antes de hacer cualquier cosa con el resultado. Puede averiguar el número de filas mediante el uso de alguna función de PHP o la utilización de MySQL para obtener el número de registros.

## Contar filas con PHP

La función mysql_num_rows () devuelve el número de filas seleccionadas.

Su uso puede ampliar el objeto de bases de datos:

```
public function numero_filas()
{
  return mysql_num_rows($this->resultado);
}
```

Además de esta función tenemos mysql_num_fields (), que devuelve el número de campos del una tabla; es necesaria cuando hacemos una consulta de datos y no sabemos cuántas columnas hemos seleccionado.

## Contar filas con MySQL

La alternativa es utilizar una consulta de SQL con la sentencia count ( * ) . Esto requiere hacer dos consultas, una para obtener el número de filas y otra para obtener el resultado.

Utilicemos el objeto para realizar la operación:

```
<?php
require_once("MySQL.php");
$servidor = "localhost";
$usuario = "root";
$pass = "";
$base_datos = "Compras";
$usuario = new
Servidor_Base_Datos($servidor,$usuario,$pass,$base_datos);
$consulta = "select count(*) as numero_filas from Usuario";
$usuario->consulta($consulta);
$fila = $usuario->extraer_registro() ;
$numero_filas = $fila["numero_filas"];
$consulta = "select * from Usuario";
$usuario->consulta($consulta);
echo "El número de filas es: $numero__f ilas<br>" ;
while ($fila = $usuario->extraer_registro()) {
       foreach ($fila as $indice => $valor) {
                echo "$indice: $valor<br>";
       }
}
?>
```

La primera sentencia SELECT cuenta el número de filas y al resultado le da el nombre numero__filas.

Para acceder al número se hace como siempre, leyendo el resultado del *.irray* $fila [ "numero_filas " ] .

La figura 12.3 muestra el resultado.

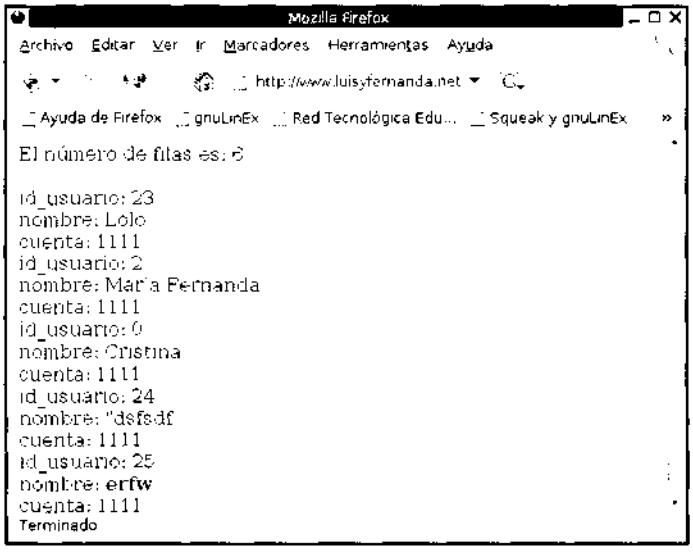



**Figura 12.3.** Número de filas del resultado.

## Contar filas afectadas

Es posible que sea necesario saber cuántas filas han sido afectadas después de añadir, actualizar o borrar algunas filas.

La función `mysql_affected_rows()` devuelve el número de registros afectados después de un cambio en las filas de una tabla. Actualizando nuestra clase, tenemos:

```
public function filas_afectadas()
{
  return  mysql_affected_rows($this->descriptor);
}
```

## Último número insertado

Cuando utilice campos auto numéricos en una tabla e inserte un nuevo registro, será útil conocer el número de la última fila insertada.

Para obtener este número tan solo tiene que utilizar la función `mysql_insert_id()`. Se puede utilizar de la siguiente forma:

```
public function ultima_fila()
```

```
{
  return mysql_insert_id($this->descriptor);
}
```

## Búsquedas dentro de una tabla

La forma de buscar ocurrencias en una columna es utilizar la sentencia LIKE de SQL.

```
SELECT * FROM Usuario WHERE nombre LIKE '%Luis%'
```

El símbolo % es un comodín, es decir, puede sustituir a cualquier carácter dentro de la columna donde busque.

## Definición de bases de datos

Como indicamos antes, lo ideal es que un administrador de MySQL le genere las bases de datos y las tablas. Después de esto, con su usuario y contraseña podrá hacer las consultas que tenga autorizadas.

En muchos casos, la figura del administrador de bases de datos no existe y es el desarrollador el que asume este papel. En este caso, le interesará conocer la forma de crear desde un *script* sus bases de datos.

## Creación de bases de datos

Para crear una base de datos tiene que utilizar la función mysql_create_db () . Recibe dos parámetros, el primero es el nombre de la base de datos que quiere crear y el segundo el descriptor devuelto por la función mysq_connect () . La función mysql_drop_db () borra la base de datos del gestor de MySQL.

```
<?php
$descriptor = mysq_connect($servidor, $usuario, $clave) ;
mysql_create_db("Nueva_Base" , $descriptor);
?>
```

## Creación de tablas

La creación de tablas debe hacerse a través del lenguaje de consultas SQL, como vimos en el capítulo anterior. Como ya sabe, la función mysql_

query () acepta consultas de distintos tipos. Aprovechando esto, puede incluir una consulta para crear una tabla nueva.

SQLite es una base de datos poco tipada, es decir, que sólo es capaz de almacenar dos tipos de datos: números o caracteres. En cambio, MySQL es fuertemente tipada, porque tiene un elevado conjunto de tipos para aplicar a sus columnas. La tabla 12.1 muestra algunos tipos de datos que puede aplicar en la sentencia CRÉATE.

**Tabla 12.1.** Tipos de datos de MySQL.

| **Nombre** | **Tamaño** | **Uso** |
|---|---|---|
| BIT, BOOLEAN | 1 *byte* | Almacena valores desde 0 a 255 sin signo o -128 a 127 con signo. |
| SMALLINT | 2 bytes | De 0 a 65535 sin signo y de -32768 a 32767 con signo. |
| MEDIUMINT | 3 bytes | De 0 a 16777215 sin signo o-8388608 a 8388607. |
| INT, INTEGER | 4 bytes | De0a4294967295ode-2147483648a 2147483647. |
| BIGINT | 8 bytes | De 0 a 18446744073709551615 o -9223372036854775808a 9223372036854775807. |
| FLOAT(precisión) | De 4 a 8 bytes | Número de coma flotante con precisión. |
| DATE | 3 bytes | Almacena una fecha. |
| DATETIME | 8 bytes | Almacena una fecha con la hora y minutos. |
| VARCHAR(número) | Número bytes | Almacena tantos caracteres como indica el número. El tamaño es menor que 255 caracteres. |
| TINYBLOB, TINYTEXT | Mayor que 255 bytes | Almacena caracteres. |
| BLOB, TEXT | Más de 64 kb | Almacena un número elevado de caracteres. |

Como ejemplo puede ver el siguiente código:

```
<?
$descriptor = mysq_connect($servidor, $usuario, $clave);
mysql_create_db("Concesionario", $descriptor);
mysql_select_db("Concesionario"),-
```

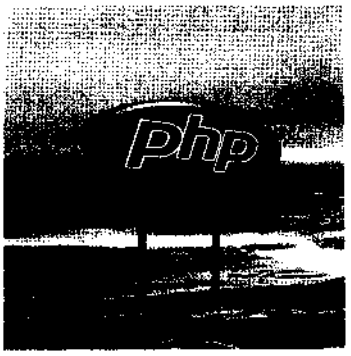
```
$consulta = "CRÉATE TAELE vehículos (
                    Íd_vehicul0 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                    color  VARCHAR(25),
                    fecha_compra DATETIME)";
$resultado = mysql_query($consulta);
$mensaje = mysql_drop_db("Concesionario");
?>
```

La columna id_vehiculo identifica a un único vehículo de la tabla. Como puede ver, en MySQL, tiene que añadir algunos modificadores a la columna para identificar sus características. Estos modificadores son obligatorios en MySQL. Por ejemplo, en SQLite no hace falta añadir ninguna propiedad para que el valor se incremente automáticamente. En MySQL es necesario especificar la propiedad AUTO_lNCREMENT. Es obligatorio además que todas las columnas tengan un tipo de dato definido.

# Resumen

MySQL es, sin duda, el sistema más utilizado a escala mundial. La mayor parte de las aplicaciones en PHP están escritas para MySQL, porque da una fiabilidad y velocidad absolutas. Hay que decir también que la versión 4 tiene bastantes restricciones con respecto a otros gestores de bases de datos como PostgreSQL, pero se irán solventando en las futuras versiones.

Si terminó el capítulo con éxito, todavía le queda un paso por andar. La extensión de MySQL que hemos utilizado es la que viene por defecto funcionando con PHP 5. Recientemente se ha creado desde cero una nueva forma de interactuar con MySQL, que permite hacer transacciones o consultas almacenadas (de la forma que lo hace Oracle). La extensión se llama MySQLi y por su leve dificultad, se escapa del objetivo de este libro.




Capítulo 13

# Sesiones y Cookies

**En este capítulo aprenderá a:**

- Diferenciar entre sesiones, cookies y cabeceras HTTP.
- Utilizar sesiones para ocultar variables entre usuarios y páginas Web.
- Almacenar la configuración de un usuario con cookies.
- Enviar información de cabecera vía Web.

## Introducción

Una sesión es un período de tiempo durante el cuál, una persona determinada ve un número de diferentes páginas Web de un determinado dominio.

El protocolo HTTP no tiene forma alguna de distinguir un usuario que se conecte desde España de otro que se conecte desde Japón. PHP 5 provee un mecanismo que permite añadir cierto nivel de diferenciación entre usuarios.

Miremos un ejemplo:

```
<HTML>
<BODY>
<?php
switch ($_GET["opción" ] ) {
        case 1 :
               echo "La opción es 1";
               break;
        case 2 :
               echo "La opción es 2" ;
               break;
        case 3 :
               echo "La opción es 3";
               break;
}
?>
</BODY>
</HTML>
```

Esta página elige una de las opciones en función del parámetro GET que obtenga. Como ya sabe, GET pasa todos los valores junto a la dirección Web y es visible para nosotros.

El método GET no es muy funcional cuando necesite pasar datos comprometidos a través de varias páginas Web. POST resuelve en parte este problema.

Este método permite pasar entre páginas las variables que quiera, quedando ocultas para el usuario. A priori parece una buena forma de ocultar datos, pero nos encontramos que es necesario un formulario en cada página para utilizarla.

Las sesiones y las variables de sesión permiten pasar datos entre páginas, sin necesidad de utilizar formularios y es la mejor forma para ocultar datos comprometidos.

# Sesiones en PHP 5

Las sesiones deben soportar:

- Detectar que durante el transcurso de la navegación por varias páginas Web, la sesión permanece invariable.
- Almacenar información que formará parte de una sesión en concreto.

PHP trabaja con una combinación de métodos que son capaces de ocultar variables y *cookies,* de las que hablaremos en un apartado posterior.

## Instanciando sesiones

El primer paso que nuestro *script* debe realizar es la llamada a la función `session_start()`. Esta función registra una sesión en el servidor y la identifica con una cadena de 32 caracteres. Cada persona que entre en ese mismo instante en la Web desde un ordenador diferente, obtendrá un identificador distinto.

La función `session_start()` debe incluirse en todas las páginas Web que formen parte de la aplicación, porque no sólo se encarga de inicializar la sesión, sino que también se encarga de mantener la misma sesión durante el proceso. La función actúa de la siguiente forma:

- Si no existe una sesión activa, crea una nueva con un identificador.
- Si existe una sesión que puede ser recuperada, se extrae la sesión y las variables asociadas.

```
<?php
session_start();
?>
<HTML>
<BODY>
<?php
echo "La variable de sesión es:" . SID;
?>
</BODY>
</HTML>
```

La figura 13.1 muestra la variable de sesión. En el momento en que *elscript* escribe cualquier cosa, el servidor termina de enviar las cabeceras HTTP y empieza a enviar el contenido. No se puede enviar una nueva cabecera cuando ya estamos enviando el cuerpo de la Web. Puesto que la función `session_start()` utiliza cabeceras para pasar la información de sesión,

su utilización debe estar siempre antes de enviar cualquier etiqueta de HTML, dato o, incluso, un espacio en blanco.

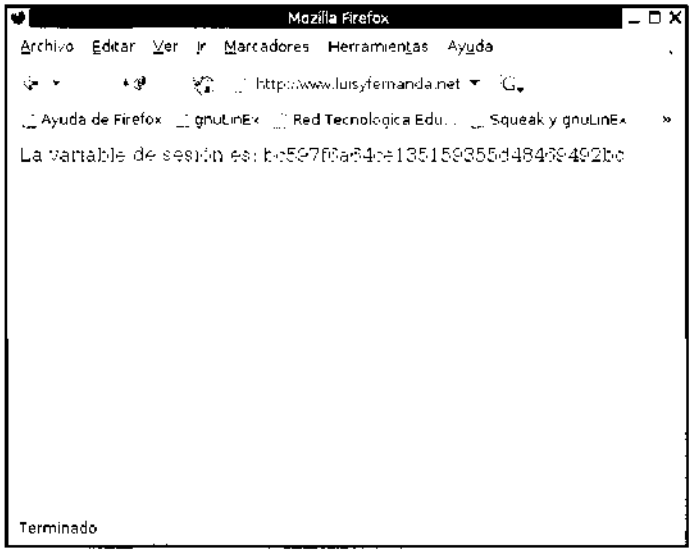



**Figura 13.1.** Variable de sesión.

Si utiliza la función de sesión o `header()`, después de obtener la cabecera HTTP obtendrá el error de la figura 13.2.

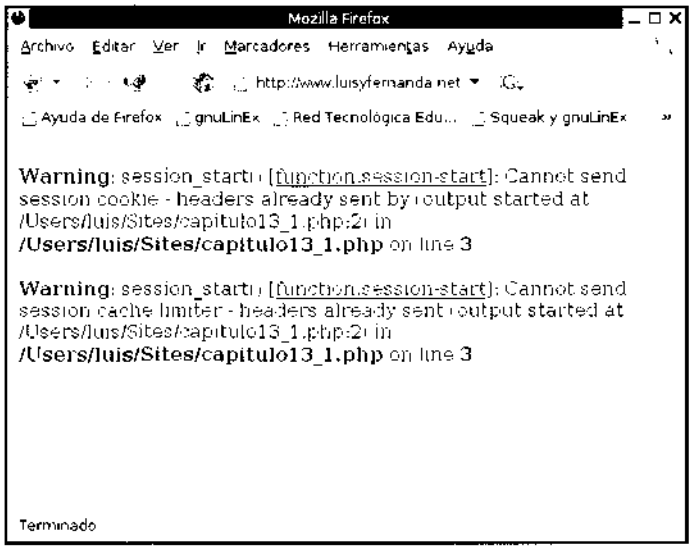



**Figura 13.2.** Error al enviar datos después de una cabecera.

La primera vez que instanciamos una sesión, *sé* crea una constante llamada S ID que contiene el identificador de 32 caracteres que nos identifica.

## Variables de sesión

Existe una variable súper-global, que se encarga de almacenar las variables que quiere pasar entre páginas de una sesión. El *array* $_SESSION permite almacenar datos de cualquier tipo para recuperarlos en otras páginas que tengan la misma sesión. El código para propagar variables de sesión puede ser tan sencillo como la página sesionl. php:

```
<?php
session_start();
?>
<HTML>
<BODY>
<?php
$_SESSION["nombre"] = "Luis Miguel Caberas";
$_SESSION["edad"] = 29;
echo "<a href = \ " sesion2.php\">Pasar variables</a>" ;
?>
</BODY>
</HTML>
```

La. va.tva.bie súpex-^tóbal almacena. valores de. d'-feteate tupo, y creamas. UTJ. enlace a la página sesion2 .php para poder recuperarlos:

```
<?php
session_start();
?>
<HTML>
<BODY>
<?php
echo "Mostrar las variables de sesión<bí>";
foreach ($_SESSION as $indice => $valor) {
        echo "$indice: $valor<br>";
}
echo "Mostrar las COOKIES<br>" ;
foreach ($_COOKIE as $indice => $valor) {
        echo "$indice: $valor<br>" ;
}
?>
</BODY>
</HTML>
```

Si se apoya en el bucle *foreach,* podrá extraer todos los valores pasados desde la Web sesionl .php. Como curiosidad, hemos introducido un bucle que extrae las variables almacenadas en cookies y puede ver, en

la figura 13.3, que el identificador de sesión se almacena en el ordenador local con el nombre de PHPSESSID.


**Figura 13.3.** Las sesiones se almacenan en cookies.

**Nota:**

*Como veremos más adelante, ¡as* cookies *son pequeños contenidos que se almacenan en el ordenador local. Las* cookies *son muy utilizadas por los servidores Web para guardar las preferencias de un usuario o algún dato interesante. En realidad son variables que se guardan en el ordenador local del usuario y pueden ser rescatadas por PHP en alguna otra ocasión en la que se necesite.*

## Problemas con los navegadores

Un problema muy común es que el usuario tenga, por razones de seguridad, desactivada la opción de permitir cookies en su navegador. El problema es que el ordenador local no va a aceptar nuestra variable PHPSESSID y todas las páginas creerán que no existe sesión activa y no podrán recuperar las variables. Se puede resolver con unas pocas líneas de código para la página sesionl.php:

```
<?php
```

```
session_start();
?>
<HTML>
<BODY>
<?php
$_SESSION["nombre"] = "Luis Miguel Cabezas";
$_SESSION["edad"] = 29;
$id_sesion = SID;
echo "<a href=\"sesion2.php?$id_sesion\">Pasar variables</a>" ;
?>
</BODY>
</HTML>
```

Esta forma de trabajar implica tener que pasar en todos los enlaces el identificador de sesión. La forma correcta de hacerlo es la que muestra el ejemplo, ya que produce una URL de este tipo: sesion2 . php?PHPSESSID = 9097c43aa86d3 45 9bl66d71c4ea4c930.

**Nota:**

*Si quiere que PHP sea transparente manejando el paso de variables a través de* cookies *o vía el método* GET, *necesita tener configurado PHP 5 con las directivas* -enable-trans-sid *y* -enable-track-vars. *Si estas opciones no están activadas, el único trámite para utilizar variables es pasar mediante GET o* POST *el identificado!' de sesión corno en el ejemplo anterior.*

## Funciones para el manejo de sesiones

La tabla 13.1 muestra algunas funciones interesantes con la descripción de uso.

**Tabla 13.1.** Funciones relacionadas con las sesiones.

| Función | Descripción |
|---|---|
| session_start() | Inicia una sesión y permite almacenar variables en la estructura $_SESSION. |
| session_destroy() | Elimina todas las variables de sesión. |
| session_name() | Devuelve el nombre de la sesión, normalmente PHPSESSID. |
| session_id() | Devuelve los 32 caracteres que forman el identificador de sesión. |

Con las funciones relacionadas en la tabla anterior, podemos construir un objeto que nos facilite el manejo de sesiones en nuestras aplicaciones.

```
<?php
class sesión
{
        function __constructor()
        {
                session_start();
        }
        public function set($nombre,$valor)
        {
                $_SESSION[$nombre] = $valor;
        }
        public function get($nombre)
        {
                if (isset($_SESSION[$nombre])) {
                        return $_SESSION[$nombre];
                } else {
                        return false;
                }
        }
        public function borrar_variable($nombre)
        {
                unset ($_SESSION[$nombre]);
        }
        public function  borrarsesíon()
        {
                $_SESSION = arrayO;
                session_destroy();
        }
}
?>
```

Las páginas sesionl .php y sesion2 . php pueden manejarse ahora con la clase que hemos creado de la siguiente forma:

```
<?php
require_once("clase__sesion.php");
$sesión = new sesión();
?>
<HTML>
<BODY>
<?php
$sesión->set("nombre","Luis Miguel Cabezas");
$sesión->set("edad",29);
$id_sesion = SID;
echo "<a href=\"sesion2.php?$id_sesion\">Pasar variables</a>";
?>
</BODY>
</HTML>
```

# Cookies

Una *cookie* es una pequeña parte de información que queda almacenada en el ordenador local de los usuarios de una página Web. Debe contener siempre un nombre y un valor. Es muy usual utilizar *cookies* para guardar preferencias de cada usuario en el uso de una Web, pero hay que tener cuidado con el número de variables que se pueden almacenar, pues normalmente hay un máximo de 20 por dominio, según el navegador.

En PHP 5 la función `setcookie()` se encarga de registrar las *cookies* durante el envío de las cabeceras. Para recuperar el valor, tiene que utilizar la variable súper-global `$_COOKIE`.

## setcookie()

La función `setcookie()` normalmente tiene 2 argumentos, el nombre de la variable y el valor. Además, podemos pasar varios parámetros que permiten dar un tiempo de vida a las variables, asociar un dominio o asegurar el envío mediante HTTPS.

La tabla 13.2 muestra en orden todos los argumentos que puede recibir una *cookie*.

**Tabla 13.2.** Argumentos de la función setcookie().

| Argumento | Tipo | Descripción |
|---|---|---|
| nombre | carácter | El nombre de la variable *cookie.* |
| valor | carácter | El valor que debe almacenar la variable. Si este valor no se indica, la *cookie* será automáticamente borrada. |
| tiempo de vida | entero | Indica el tiempo de vida de *unacookie.* Si el valor es 0, la variable nunca expirará. Si indica un número entero, tendrá el tiempo que la variable permanecerá activa. El valor hay que asignarlo en tiempo absoluto (como el devuelto por la función `mktime()`). |
| ruta | entero | Permite distinguir entre dos variables con el mismo nombre, pero llamadas desde distinta ruta en el servidor. La *cookie* "nombre" llamada desde la ruta `/foro` será distinta a la *cookie* "nombre" llamada desde la ruta `/usuario`. |

| Argumento | Tipo | Descripción |
|---|---|---|
| dominio | carácter | Asegura el dominio de servidor que puede leerla *cookie.* |
| Secure | entero (0 ó 1) | Si el argumento es 1, la *cookie* sólo será enviada a través de una conexión segura HTTPS. |

## Borrar una cookie

Borrar una *cookie* es fácil. Si llama a la función `setcookie ()` con los mismos argumentos que para habilitarla, excepto el valor, la variable se eliminará del sistema local del usuario. Si usa los parámetros `ruta` y `dominio`, deben aparecer también, y con los mismos valores.

Como de costumbre, vamos a implementar una pequeña clase para manejar las *cookies:*

```
<?php
class Cookie
{
   function___constructor()
   {
            //El constructor no hace nada
   }
   public  function  set($nombre,$valor,$expire=0,$ruta="
   ",$dominio="",$seguro=0)
   {
            setcookie($nombre,$valor,$expire,$ruta,$dominio,$seguro);
   }
   public function get($nombre)
   {
            if (isset ($_COOKIE [$nombre] )) {
             return $_COOKIE[$nombre];
            } else {
             return false;
            }
   }
   public function borrar_cookie($nombre)
   {
            setcookie($nombre,$valor,$expire,$ruta,$dominio,$seguro!;
   }
}
?>
```

El argumento "tiempo de vida", debe proporcionarse en valor de tiempo absoluto. Esto a veces puede resultar incómodo y sería más sencillo pro-

porcionar el tiempo de vida de la variable en minutos o segundos. La clase anterior solicita el tiempo en segundos y se encarga de añadir esos segundos al valor absoluto haciendo más simple la asignación.

Para ver cómo funciona, podemos utilizar la Web cookiel .php:

```
<?php
require_once("clase_cookie.php");
$cookie = new CookieO;
$cookie->set("nombre","Luis Miguel",10) ;
?>
<HTML>
<BODY>
<?php
echo "La COOKIE es: " . $cookie->get("nombre");
echo "<a href = \ "cookie2.php\">Pasar variables</a>" ;
?>
</BODY>
</HTML>
```

Como puede ver es muy sencilla la asignación de *cookies y* su recuperación.

## Cabeceras HTTP

Las funciones setcookíe () o session_start () introducen su contenido en una cabecera HTTP sin necesidad de especificación por el usuario. Es posible enviar cabeceras desde PHP con la utilización de la función header () . La función header () es muy simple; tan sólo toma como argumento un conjunto de caracteres que será la cabecera a enviar. Por ejemplo, una forma muy útil de redirigir una página a otra es utilizando la cabecera "Location:".

```
<?php
if (isset($_GET["usuario"] ) && $_GET["usuario"] == "Luis") {
          header("Location: http://www.luisyfernanda.net") ;
}
?>
<html>
<body>
Si ve esta página, su nombre no es Luis
</body>
</html>
```

Este *script* evalúa si en la variable súper-global $_GET existe un identificador llamado usuario y si su valor es Luis. En caso afirmativo se envía

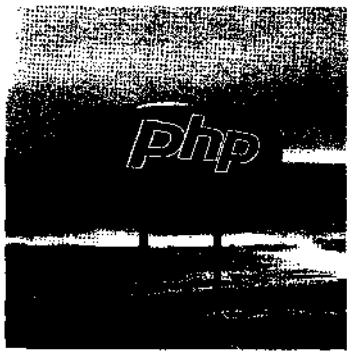
la cabecera al navegador, que invita a cargar la Web especificada. En caso negativo, se envía la página Web que hay después de la estructura *if*.

Este tipo de uso de las cabeceras puede servir para redireccionar a una página de error cuando un usuario intente entrar sin permiso o cometa un fallo en la entrada de un nombre o correo electrónico.

## Resumen

Las sesiones resuelven los problemas muy comunes de seguridad. El uso conjunto de sesiones *cookies* y cabeceras HTTP permiten un control absoluto sobre las variables de la aplicación. Casi todos los programas que necesitan autentificar un usuario (como una tienda *on-line)* se sirven de estas técnicas.

Pero el lector no debe quedarse aquí, hay muchas cabeceras distintas que pueden ser útiles en diferentes momentos, como la autenticación vía HTTP.




Capítulo 14

# Lectura y escritura de archivos XML

**En este capítulo aprenderá a:**

- Conocer el lenguaje XML.
- Diferenciar entre los métodos SAX, DOM y SimpleXML.
- Realizar un parse de XML con SAX.
- Leer y escribir archivos XML con DOM.
- Crear un parse muy potente y sencillo con SimpleXML.

# Introducción

XML (Lenguaje de Marcas eXtensible), forma parte de SGML (Lenguaje de Marcas Generalizado eStandar). Aún así, no es necesario saber nada de SGML para utilizar XML. El lenguaje XML define una estructura de documentos que pueden ser leídos por personas y por ordenadores.

El camino más sencillo para comprender XML es pensar en cómo se utilizan los documentos HTML. Estos documentos están estructurados y funcionan con etiquetas y atributos. Las etiquetas van encerradas entre símbolos de mayor y menor () y deben cerrarse de la misma forma añadiendo un símbolo de barra invertida (). Los documentos HTML tienen una ortografía específica, es decir, unas reglas que definen la forma correcta de estructurar un documento; por ejemplo, la etiqueta <BODY> debe ir siempre después de </HEAD> o, sino existe ésta, de <HTML>. Además, HTML contiene un diccionario cerrado de etiquetas que definen el lenguaje y no podemos utilizar otras que no estén especificadas.

En cambio, XML no tiene un diccionario de etiquetas. Las etiquetas que aparecen son las que nosotros mismos creamos. La única norma que tenemos que seguir es que toda etiqueta de inicio (<coche>) debe tener una etiqueta de fin (</coche>) .

El documento siguiente muestra un archivo XML bien formado:

```
<?xml version="1.0" ?>
<biblioteca>
  <tema  id="informática">
          <libro>
              <titulo>Manual Imprescindible de PHP 5</titulo>
              <autor>Luis Miguel Cabezas Granado</autor>
          </libro>
          <libro>
              <titulo>Delphi 7</titulo>
              <autor>Marco Cantu</autor>
          </libro>
          <libro>
              <titulo>Delphi 6 y Kylix</titulo>
              <autor>Francisco Charte Ojeda</autor>
          </libro>
  </tema>
</biblioteca>
```

Como puede ver, la estructura del documento es muy similar a la de una página Web. Está formado por etiquetas y atributos, cuya definición puede inventarse sobre la marcha, es decir, las etiquetas <libro>, <titu-

lo> o <seccion> podrían tener un nombre totalmente diferente, pero no así en HTML, que está obligado a mantener el lenguaje definido. Algunos navegadores, como Mozilla, interpretan los ficheros XML como en la figura 14.1.

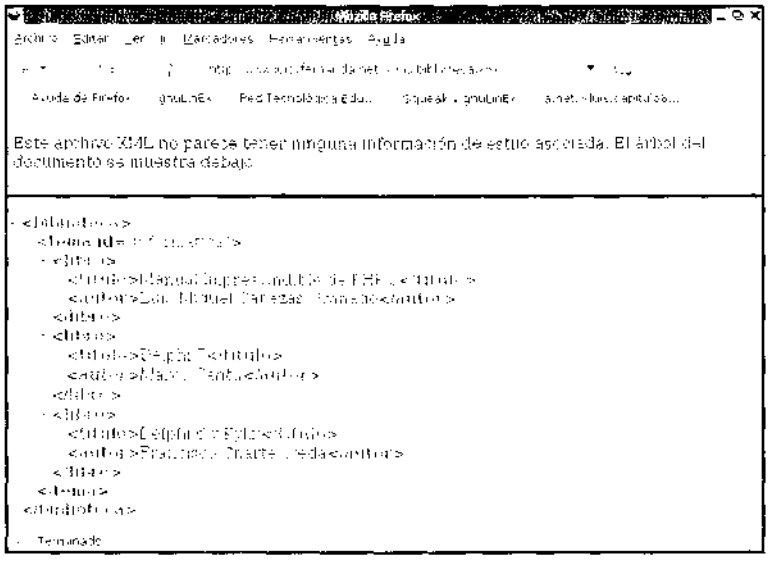



**Figura 14.1.** Archivo XML interpretado por Mozilla FireFox.

Un documento XML está obligado a cumplir ciertas normas, que permiten definir un texto bien formado:

- Debe tener un único elemento raíz. Sólo puede haber un par de etiquetas que diferencien el inicio y el final del documento, como en HTML, donde se utiliza <HTML> y </HTML>.
- Los elementos deben ser hereditarios. La estructura < A x B x / B > < / A> se permite, pero no la siguiente < A x B x / A x / B > . En la primera forma la etiqueta <A> envuelve a la etiqueta <B>, que es la forma correcta de escribir un documento XML. HTML, sin embargo, permite la segunda forma de componer un documento. <A HREF="index.php"> <Bx/Ax/B>.
- Todos los elementos tienen que tener una etiqueta de cierre. La pareja <titulo></titulo es correcta. HTML permite etiquetas sin cerrar como <B> o <LI>.
- Los elementos pueden tener entre las dos etiquetas cualquier tipo de contenido, como <titulo>Manual imprescindible de PHP 5</titulo>.

- Los caracteres &, <, >, las comillas simples y las comillas dobles están prohibidas como contenido y deben utilizarse símbolos de escape para utilizarlas.

## SAX, DOM y SimpleXML

Lo primero que tenemos que hacer antes de leer archivos XML es conocer las tres formas existentes, con sus ventajas e inconvenientes:

- **SAX:** Es más ligero y fácil de aprender; trata los archivos XML como un flujo de datos que se leen poco a poco.
- **DOM:** Lee el fichero completo y crea un objeto en memoria. Permite crear archivos desde cero.
- **SimpleXML:** Es el más sencillo de utilizar. Los elementos se pueden leer con un simple `foreach`.

## SAX

Se utiliza para parsear elementos XML. Está basado en eventos, lo que significa que el parse hace llamadas a determinadas funciones, dependiendo de los elementos que examina.

Los eventos de SAX son proporcionados por PHP en forma de función. Al parsear se reconocen piezas de XML, como etiquetas de inicio o fin, datos o entidades externas y cada una de estas piezas hace una llamada a un evento. El procedimiento para parsear un archivo XML con SAX debe seguir los siguientes pasos:

- Determinar qué clase de eventos queremos manejar.
- Escribir una función que maneje cada evento. Es muy común escribir una función para manejar los datos y las etiquetas de inicio y de fin.
- Crear el parse usando la función `xml_parser_create ()` y hacer la llamada con `xml_parse ()` .
- Liberar la memoria usada con la función `xml_parser_free ()`.

Hemos creado un objeto para leer archivos XML. En principio no hace gran cosa; tan solo examina las etiquetas de inicio y fin y las imprime en pantalla, pero con esto bastará para aprender la peculiar estructura de un parse SAX.

```
<?php
```

```
class xml
{
 function___construct($fichero_xml)
 {
          $this->fichero_xml = $fichero_xml;
          $this->xml_parser=xml_parser_create();
//Puesto que el parse lo creamos desde un objeto debemos
registrar la incidencia con xml_set_object
          xml_set_object($this~>xml_parser,$this);
//Funciones de callback para manejar XML mediante SAX
          xml_set_element_handler($this->xml_parser,"elemento_
          inicio","elemento_fin");
          if (!($fp = fopen($this->fichero_xml,"r")))
                    {
                    die ("Error de Entrada y Salida");
                    }
          while ($datos = fread($fp,filesize($this->fichero_xml)))
                    {
                    if   (!xml_parse($this->xml_parser,$datos,
                    feof($fp)))
                              {
                              die ("Fallo en el XML");
                              xml_error_string(xml_get_error_
                              code($this->xml_parser));
                              }
                    xml_parser_free($this->xml_parser);

                    }

 function  elemento_inicio  ($parser,$nombre,$atributos)

          echo  "$nombre<br>";

 function elemento_fin  ( $parser,$nombre)

          echo  "/$nombre<br>";

}
$biblioteca = new xml("biblioteca.xml");
?>
```

Si guarda la clase en un archivo llamado clase_xml. php podrá usarla en sus proyectos tan sólo con instanciar el objeto. Si comienza por el constructor, verá que el parámetro que recibe es el archivo que quiere parsear. Debe crear el parse con la funciónxml_parse_create () y después registrarlo en PHP como un objeto mediante la función xml_set_ object () . Si crea un parse fuera de la estructura de un objeto no hará falta hacer la llamada a la función anterior.

La función xml_set_handler () indica al parse que los eventos encargados de encontrar las etiquetas de inicio y de fin serán manejadas con las funciones que se pasan como parámetro. En este caso elemento_inicio ( ) se ejecutará cada vez que se encuentre una etiqueta de inicio y elemento_fin {) cuando se encuentre la etiqueta de cierre.

La última parte del constructor lee el archivo pasado como argumento y analiza los datos a través de la función xml_parse (), que inicia la secuencia de análisis. Si ejecuta una instancia de la clase, comprobará que el navegador muestra todas las etiquetas de inicio y de fin con una rigurosa herencia. La figura 14.2 muestra las etiquetas leídas del archivo XML.

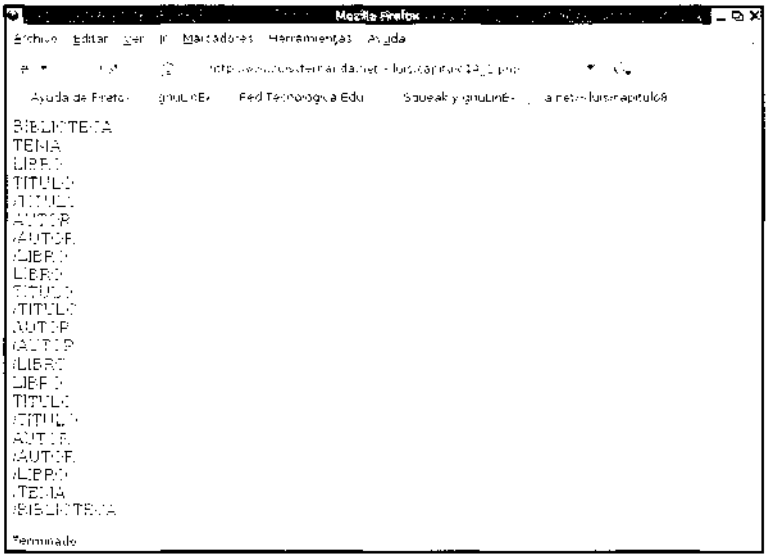



**Figura 14.2.** Etiquetas de un archivo XML con SAX.

Para poder ver los datos tiene que añadir una nueva función que maneje ese evento. La funciónxml_set_character_data__handler () permite manejar el evento que se dispara cada vez que se encuentra un dato entre dos etiquetas. Esta función tendrá que añadirla al constructor después del método que maneja los eventos de etiquetas:

```
xml_set_character_data_handler($this->xml_parser,"datos");
```

Además de esto, tendrá que incluir una nueva función, que se encargará de mostrar los datos:

```
function datos($parser,$valor)
  {
```

```
        echo "    $v9lor<br>";
    }
```

**Nota:**

*Los eventos añadidos deben llevar los argumentos necesarios para recuperar los datos. Cada vez que cree un par se del tipo SAX las funciones tienen que tener los parámetros que indicamos en nuestra clase de ejemplo.*

Después de comprobar el funcionamiento de la clase, puede modificarla para que haga algo realmente útil, como mostrar ordenadamente los datos del fichero. Las funciones de la clase pueden quedar de esta forma:

```
function elemento_inicio ($parser,$nombre,$atributos)
{
 switch($nombre) {
 case "TEMA":
      $tema = $atri.butos [" ID" ] •
      echo "Tema $t.ema<br>" ;
      break;
 case "TITULO":
      echo "<b>";
      break;
 case "AUTOR":
      echo " - " ;
      break;
 }
}
function elemento_fin ($parser,$nombre)
{
 switch($nombre) {
 case "TITULO":
      echo "</b>";
      break;
 case "AUTOR":
      echo "<br>";
      break;
 }
}
function datos($parser,$valor)
{
 echo "   $valor" ;
}
```

En este caso, el parser irá analizando las etiquetas que nos interesan y realizará una acción determinada. Por ejemplo, cuando se comprueba la exis-

tencia de una etiqueta TITULO escribimos en HTML la etiqueta de negrita <B>, para resaltar este dato, como muestra la figura 14.3.

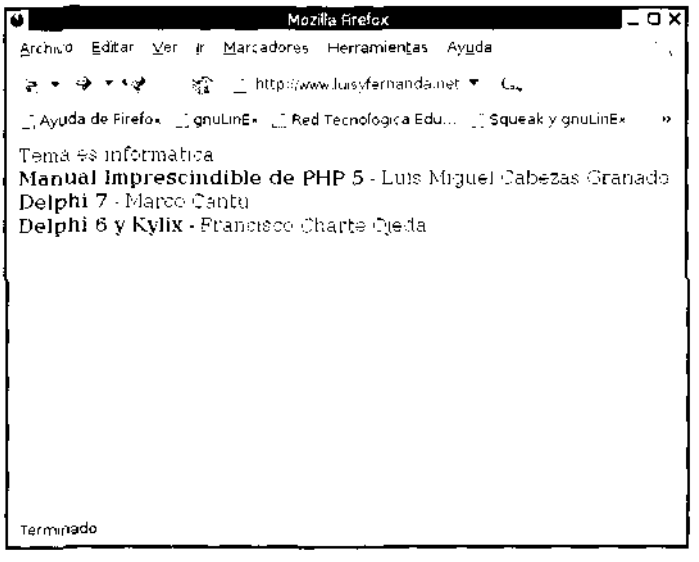



**Figura 14.3.** Archivo XML interpretado por Mozilla FireFox.

Elaborando un poco más la clase obtendrá un objeto de gran utilidad, que podrá recuperar información de bases de datos o archivos de configuración.

## DOM

Esta API define un completo camino para crear, definir y parsear archivos XML. DOM es una recomendación del consorcio *World Wide Web.*

La idea básica consiste en que todos los archivos XML pueden verse como un conjunto de nodos que forman parte de un árbol. Empezando desde el elemento raíz, del que todos los elementos nacen como hijos, cualquier programa debería ser capaz de crear una estructura lógica del documento.

La API puede utilizarse para leer archivos en memoria, modificarlos y volver a escribir el archivo con los nuevos datos.

DOM está escrito en metodología orientada a objetos, y todos los nodos son instancias de diferentes objetos.

## Usar DOM para leer archivos

El siguiente código muestra un archivo XML muy simple:

```
<?xml version="l.0" ?>
   <tema id="informática">
        <titulo>Manual Imprescindible de PHP 5</titulo>
   </tema>
```

Si parseamos este documento con DOM, la etiqueta titulo será considerada un nodo y además un hijo del nodo raíz, que en este caso es tema. Esta parte queda más o menos clara debido a la jerarquía de etiquetas que tiene XML. La controversia llega con el texto que hay en el nodo tema, que no se considera como una parte, sino como un nodo independiente cuyo valor es el texto Manual Imprescindible de PHP 5. Este nodo permite recuperar el valor con el método nodeValue.

Vamos a ver los pasos necesarios para leer el archivo biblioteca.xml, ya que difiere bastante la metodología con respecto a SAX.

### Todo es un objeto

PHP 5 implementa una clase DOM para representar los archivos XML, llamada domDocument. Como ya sabe, para instanciar la clase sólo tiene que hacer uso del operador de objetos:

```
$biblioteca = new domDocument;
```

La extensión DOM sigue las especificaciones de XML y trata los espacios en blanco como contenido. Esto significa que si entre dos etiquetas existe un espacio, el contenido será tratado como un objeto de texto. Para evitar fallos de este tipo puede ejecutar el siguiente método:

```
preserveWhiteSpace = false;
```

Lo siguiente que tendrá que hacer es leer el archivo biblioteca.xml con el método load() del objeto $biblioteca. Si el contenido XML se almacena en una variable, puede utilizar el método loadXML():

```
$biblioteca->load("biblioteca.xml");
```

A partir de aquí, ya puede recuperar el contenido del fichero, atendiendo a sus preferencias. En SAX, tenía que leer el documento completo e ir preguntando etiqueta por etiqueta para averiguar todos los autores que estaban dados de alta en el archivo XML. En DOM puede recuperar esta información usando getElementsByTagname(), que devuelve un conjunto de objetos que coinciden con la etiqueta que pase como parámetro.

```
<?php
//Instanciamos el objeto
$biblioteca = new domdocument;
//Anulamos la posibilidad de que un espacio en blanco sea un
objeto
$biblioteca->preserveWhiteSpace = false;
//Leemos el archivo XML
$biblioteca->load("biblioteca.xml");
//Buscamos todas las etiquetas libro
$libros = $biblioteca->getElementsByTagname("libro") ;
//Imprimimos todos los libros
foreach ($libros as $libro) {
   echo $libro->firstChild->nodeVaiue . "<br>";
}
?>
```

La variable `$libros` es en realidad un objeto de la clase `domnodelist`, y guarda una relación de nodos, cuya etiqueta es `libro`, es decir, hemos seleccionado todos los libros del archivo. Con un bucle `foreach` podemos sacar todos los valores. Antes vimos que el texto asociado a un nodo es un nodo hijo con un valor determinado, por eso debe llamar al método `firstChild`.

El hijo de la etiqueta `libro` es el texto asociado y su valor lo podemos recibir llamando a `nodeValue`.

El resultado es la lista de libros:

```
Manual Imprescindible de PHP 5
Delphi 7
Delphi 6 y Kylix
```

### Atributos

Para recuperar los atributos de los nodos está el método `getAttribute ()`, al que tiene que pasar como argumento el nombre del atributo. El archivo `biblioteca.xml` tiene una única etiqueta con el atributo `id`, pero nos sirve para mostrar cómo funciona el método:

```
<?php
$biblioteca = new domdocument,-
$biblioteca->preserveWhiteSpace = false;
$biblioteca->load("biblioteca.xml");
$temas = $biblioteca->getElementsByTagname("tema");
foreach ($temas as $tema) {
   echo $tema->getAttribute("id");
}
?>
```

El resultado es el nombre del tema o temas que estén almacenados.

### Búsquedas múltiples

Nuestro objetivo final es mostrar todos los títulos, con sus autores, ordenados por temática. Conociendo ya los métodos principales para recorrer los objetos DOM, es sencillo pensar en un código que implemente lo que pedimos:

```
<?php
$biblioteca = new domdocument;
Sbiblioteca->preserveWhiteSpace = false;
$biblioteca->load("biblioteca.xml");
$biblioteca->documentElement;
$temas = $biblioteca->getElementsByTagname("tema");•
foreach ($temas as $tema) {
   echo "Tema: " . $tema->getAttribute("id") . "<br>";
   $libros = $tema->getElementsByTagname("libro");
   foreach ($libros as $libro) {
        echo "<b>" . $libro->firstChild->nodeValue . "</b> - " ;
        $autores = $libro->getElementsByTagname("autor");
        foreach ($autores as $autor) {
             echo $autor->firstChild->nodeValue . "<br>";
        }
   }
}
?>
```

Aunque es un poco más complejo, el fundamento básico es ir utilizando bucles para leer los hijos de un nodo determinado hasta recuperar todos los datos necesarios.

## Escribir archivos XML con DOM

DOM permite hacer más cosas que la simple impresión de los datos en pantalla. Se puede utilizar para crear nuevos documentos con el nivel de jerarquía que quiera. Para hacer esto debe apoyarse en la creación de objetos de la clase domElement () .

Vamos a crear un *script* para crear el siguiente archivo XML:

```
<?xml version="1.0"?>
<biblioteca>
   <tema>Texto</tema>
</biblioteca>
```

Para crearlo necesita instanciar una serie de objetos que empieza por la creación de un documento DOM, mediante la utilización del objeto domdocument () .

```
<?php
```

```
$biblioteca = new doradocument("1.0");
$raiz = new domelement("biblioteca") ;
$raiz = $biblioteca->appendChild($raiz);
$tema = new domelement("tema","Texto");
$tema = $raiz->appendChild($tema);
$biblioteca->save("biblioteca.xml");
?>
```

La variable $raiz es el objeto que contendrá todo el documento. Para crear las etiquetas tiene que utilizar el método appendChild (). El objeto que llama al método contendrá como hijo al elemento que se pasa como parámetro.

Si se fija en el ejemplo, $tema es un nuevo elemento que tendrá la etiqueta tema.

Para que $tema sea un hijo de la raíz, debe hacer que el objeto padre, $raiz, llame al método appendChild () con el parámetro $tema.

Para grabar el resultado final, el objeto DOM tiene que llamar al método save (). El fichero biblioteca.xml lo puede crear con el código siguiente:

```
<?php
$biblioteca = new domdocument("1.0");
$raiz   = new domelement("biblioteca");
$raiz   = $biblioteca->appendChild($raiz);
$tema   = new domelement("tema");
$tema   = $raiz->appendChild($tema);
$tema->setAttribute("id","informática");
$libro  = new domelement("libro") ;
$libro  = $tema->appendChild($libro);
$titulo = new domelement("titulo","Manual Imprescindible de PHP 5" •
$titulo = $libro->appendChild($titulo);
$autor  = new domelement("autor", "Luis Miguel Cabezas Granado");
$autor  = $libro->appendChild($autor);
$libro  = new domelement("libro");
$libro  = $tema->appendChild($libro);
$titulo = new domelement("titulo","Delphi 7") ;
$titulo = $libro->appendChild($titulo);
$autor  = new domelement("autor","Marco Cantu");
$autor  = $libro->appendChild($autor);
$libro  = new domelement("libro");
$libro  = $tema->appendChild($libro);
$titulo = new domelement("titulo","Delphi 6 y Kylix");
$titulo = $libro->appendChild($titulo);
$autor  = new domelement("autor", "Francisco Charte Ojeda");
$autor  = $libro->appendChild($autor);
$biblioteca->save("biblioteca.xml");
?>
```

El conjunto de líneas de código anterior crea un archivo completo XML. Quizá sea mejor dividir el texto en funciones recursivas que permitan teclear menos a la hora de crear los documentos, pero eso ya es trabajo para el lector. El método setAttribute() añade un atributo al objeto que hace la llamada; el primer argumento será el nombre del atributo y el segundo su valor. La llamada a domelement() puede tener uno o dos valores. Si lleva un atributo significará que el nombre de la etiqueta será ese parámetro. Si se llama con dos parámetros, el primero será el nombre de la etiqueta y el segundo un objeto de texto que se añadirá como texto hijo del nodo.

## Modificar archivos XML

La verdadera potencia de DOM reside en el poder de gestionar los archivos para añadir, modificar o eliminar nodos. La mayoría de los algoritmos que realizan alguna de estas operaciones se basan en la búsqueda de los nodos para, posteriormente, añadir hijos nuevos o eliminar los encontrados. Para añadir nuevos elementos se hace como cuando creamos el fichero biblioteca.xml, con el método appendChiId(). Antes de añadir nuevos nodos tiene que localizar el lugar exacto donde quiere añadir las etiquetas. El ejemplo siguiente muestra cómo añadir un nuevo nodo tema:

```
<?php
$biblioteca = new domdocument("1.0");
$biblioteca->load("biblioteca.xml");
$raiz = $biblioteca->documentElement;
$tema_nuevo = new domelement("tema");
$tema_nuevo = $raiz->appendChild($tema_nuevo);
$tema_nuevo->setAttribute("id","ficción");
$biblioteca->save("biblioteca2.xml");
?>
```

Los pasos son los que ya ha aprendido. Cargue el documento y cree un objeto, llamado $raiz, que se convierta en la etiqueta raíz, mediante el método documentElement. Cree un nuevo elemento del tipo tema y añádalo, para después insertar el atributo que se necesita. Por último, guarde el archivo en el fichero biblioteca2.xml.

# SimpleXML

Probablemente, el punto más fuerte de PHP 4 fue la incorporación de herramientas para el soporte de XML. PHP 5 le da una vuelta más a la tuerca

implementando nuevas herramientas de lectura, elaboración y modificación de archivos XML basado en la librería libxml2 y una nueva API llamada SimpleXML.

Si XML es un lenguaje bien construido y legible por personas y ordenadores, los programas para manipular estos archivos deberían ser también sencillos de utilizar. SimpleXML nace con esta premisa permitiendo leer un fichero completo con una sola instrucción e, inmediatamente después, poder leer los datos como conjunto de variables PHP.

El conjunto de funciones que pertenece a la API SimpleXML es nuevo en PHP 5. Mantiene una absoluta flexibilidad a favor de la simplicidad y bajo nivel de memoria usado. SimpleXML utiliza el menor número de líneas de código para leer o escribir datos en un fichero XML.
El ejemplo siguiente muestra cómo podemos parsear el archivo biblioteca, xml:

```
<?php
$biblioteca=simplexTnl_load_fiie("biblioteca.xml");
foreach ($biblioteca->tema as $tema) {
        echo "Tema es " . $tema["id"] . "<br>";
        foreach ($tema-?libro as $libro) {
                 echo "<b> " . $libro->titulo ."</b> - ";
                 echo $libro->autor . "<br>";
        }
}
?>
```

Lo primero que llama la atención es que en apenas 10 líneas de código, se ha conseguido extraer todos los datos necesarios, a diferencia de SAX o DOM.

La función simplexml__load_file () crea un objeto con el archivo XML que pase como argumento.

A partir de aquí puede acceder a todos los datos, así como acceder a las variables de los objetos:

```
echo$biblioteca->tema->libro[0]->titulo;
echo  $biblioteca->tema->libro[1]->titulo;
```

Para extraer el contenido que necesita lo más sencillo es crear una estructura de bucles foreach para ir sacando los datos ordenados.

El nombre SimpleXML es totalmente descriptivo, aunque no hay que confundir la palabra simple con básico. La prueba de su potencia está en que los desarrolladores de PEAR van a utilizar SimpleXML para desarrollar su librería de SOAP.

## Resumen

XML se ha convertido en un estándar para el intercambio de información en Internet. Numerosas aplicaciones están construidas con algún tipo de tecnología que implementa XML, como XML-RPC, RSS o SOAP.

En este capítulo ha aprendido a leer y escribir documentos de tres formas posibles. La más completa sin duda es DOM, que permite, además de leer archivos, escribir y modificarlos a nuestro antojo. El problema es que puede resultar algo complicado de entender al principio y quizá no merezca la pena en aplicaciones sencillas.

En el capítulo siguiente conocerá la verdadera aplicación práctica. Basándonos en SimpleXML y en DOM crearemos algunas clases para leer archivos de noticias o ejecutar funciones definidas en un ordenador remoto.


Capítulo 15

# Aplicaciones prácticas de XML

**En este capítulo aprenderá a:**

- Entender el funcionamiento de RSS.
- Incluir en sus páginas noticias que provienen de un RSS.
- Crear archivos RSS desde DOM.
- Utilizar un cliente XML-RPC para ejecutar funciones remotas.
- Desarrollar un servidor XML-RPC.

# Introducción

Existen en el mercado muchas publicaciones que hablan sobre XML y sus tecnologías derivadas. En estos libros, que suelen ser de unas 1000 páginas, se hace un recorrido por todos los rincones de XML, DTD, XLST, etcétera. Si lee alguna publicación de este tipo puede quedar desencantado porque, entre tantos conceptos, no se ve una aplicación real de la tecnología.

Como este libro es eminentemente práctico, me gustaría que el lector aprendiera dos metodologías que se utilizan actualmente en la Web: RSS y XML-RPC. La primera describe una forma de exportar información, noticias o datos al exterior para que, desde cualquier página Web, puedan leerse. La segunda es una forma de trabajar que permite ejecutar funciones que están almacenadas en un ordenador remoto.

# Compartir información con RSS

RSS es un mecanismo para publicar información sobre el contenido de un sitio Web y es muy común utilizarlo para dar las últimas noticias. Esto permite introducir en su sitio Web un documento XML con noticias de otras páginas.

La versión original de RSS, la 0.90, fue creada por Netscape con el objetivo de crear portales servidores de noticias de distinta categoría, pero resultó algo complejo de utilizar. La versión 0.91, mucho más simple, la desarrolló la empresa UserLand Software, quien lo comercializó como parte de sus productos basados en página Web, como los *Weblogs*.

Un tercer grupo de desabolladores independientes, basándose en los comienzos de RSS, la versión 0.90, escribió una nueva revisión basada en RDF; esta versión recibe el nombre de RSS 1.0. Desde la aparición de la versión 0.1, UserLand Software ha seguido desarrollando las versiones 0.92, 0.93, 0.94 y la 2.0.

## Distintos formatos

Existen 7 formatos distintos y, como programador, debería ser lo suficiente habilidoso como para manejar cualquiera de ellos. Pero como el conté-

nido lo produce la Web que desea exportar su información, tendrá que tener en cuenta las características de cada formato:

Tabla 15.1. versiones de RSS.

| Versión | Desarrollo | Descripción | Status | Recomendación |
|---|---|---|---|---|
| 0.90 | Netscape | | Obsoleta por la aparición de 1.*0* | No usar |
| 0.91 | UserLand | Simple | Muy popular, pero obsoleta por la aparición de 2.0 | Sencilla y fácil de migrar a la versión 2.0 |
| 0.92, 0.93, 0.94 | UserLand | Permite texto enriquecido y metadatos | Obsoleto por la 2.0 | Mejor usar la 2.0 |
| 1.0 | Grupo de desarrollo RSS-DEV | Basado en RDF y extensible con módulos | Muy estable | Usar si necesitamos especificaciones avanzadas |
| 2.0 | UserLand | Exter\=Áble cor», módulos. Fácil de migrar desde cualquier versión 0.9 | EsteWe y ef. continuo desarrollo | Uso de carácter general con metadatos y texto enriquecido |

Como ejemplo, vamos a leer el RSS 1.0 de la Web del autor www.luisyfernanda.net, que contiene esta información:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22 -rdf-syntax-ns#"
xmlns="http://my.netscape.com/rdf/simple/0.9/">
      <channel>
            <title>www.luísyfernanda.net</title>
            <link>http://www.luisyfernanda.net/índex.php?
            blogId=2</link>
            <descriptionx/description>
      </channel>
      <item>
            <title>PHP 5 en la línea de salida</title>
            <descriptiorv>íüadi Gutw.&ia.s dice qu.e en. 24 horas estará
            lista la nueva versión de PHP5 . . • que nervios .... &lt;br
```

/>Para el que no pueda esperar, la Reléase Candidate 4 está en: <a href ="http : // snaps.php.net/~andi/">http://snaps.php.net/ ~andi/</a><br/><br/><br/></ description>
<link>http://www.luisyfernanda.netíndex.php?op=View Article&articleId=41&blogId=2</link>
</item>
<item>
<title>Galopín 1.0</title>
<description>
&lt ;a href="http://www.luisyfernanda.net/resserver.php? blogId=2& resource=DSCN0840 . JPG " > < img align=" right&quot ; style="margin: 5px;" border="O" alt="Rueda de prensa" src="http://www.luisyfernanda.net/resserver . php?blogId=2&resource=DSCN0 840.JPG&mode=preview" / ></a>
Hoy se ha presentado en la A.A.V.V. de La Antigua, en Mérida, la versión 1.0 de Galopín, una herramienta escrita en PHP por dos desarrolladores extremeños. Tras los forcejeos con la prensa y las salidas con quite de los implicados, nos quedamos con la miel en los labios, deseando probar la aplicación y, porqué no, destriparla para ver como funciona el chisme.<br />El que quiera descargarla puede hacerlo desde la Web de sinuh en <a href="http:// galopin.sinuh.org/">galopin.sinuh.org</a><br /
*>Sclt;'bx/S<sub>i</sub>g£;*
<br /></description>
<link>http://www.luisyfernanda.net/ index.php?op=ViewArticle&articleld=4 0&blogId=2</link>
</item>
<item>
<title>EL TIEMPO DE MERIDA</tÍtle>
<description>
Hace unos días empecé escribir alguna cosilla para averiguar la temperatura que hay en Mérida. Los datos los recojo de la Web del Instituto Nacional de Meteorología, y los plasmo en una tabla. Si quieres puedes utilizar los datos llamando a la Web: < a href = "http://www.luisyfernanda.net/ clase_tiempo.php?LOCALIDAD=MERIDA">http:// www.luisyfernanda.net/clase_tiempo.php?LOCALIDAD=MERIDA.</ a><br />Donde pone LOCALIDAD se puede poner alguna población distinta como, CORIA, CACERES, BADAJOZ,UNIVERSIDAD, etc...<br />El módulo está en fase alfa y el dibujito que aparece puede que no corresponda con la realidad, pero es un avance ;)<br />En unas semanas implementaré el servicio en SOAP, como lo hace weather.com.<br />Saludos.<br / ><br /><br /><br/>

```
    &lt;br  /&gt;</description>
    <link>http://www.luisyfernanda.net/
index.php?op=VíewArticle&amp;articleld=3 6&amp;blogId=2</link>
  </item>
  <item>
    <title>PHP 5 a punto de caramelo</title>
    <description>Hace un par de días en la lista interna de
desarrolladores de PHP, Andi Gutmans ha escrito que PHP5 ya
está horneado y le falta un poquito de chocolate para salir al
mercado. Están teniendo algún problemilla con librerías que
controlan el acceso a memoria, pero se prevé que la primera
versión estable salga esta semana. &lt;br /&gt;&lt;a
href=&quot;http://www.zend.com/lists/php-dev/2004  07/
msg00059.html&quot; &gt;Notica de Zend &lt;/a&gt;&lt;br/
&gt;&lt;br  /&gt;</description>
    <link>http://www.luisyfernanda.net/
index.php?op=ViewArticle&amp;articleld=3 3&amp;blogId=2</link>
  </item>
  <item>
    <title>OpenOffice2htm  en  Sourceforge</title>
    <description> Acaban de aceptar el Proyecto OpenOffice2htm
en sourceforge, ellos sabrán lo que hacen. Lo que pasa es que
tengo un pequeño problema; entre tanto menú de administración
de ficheros, documentos, etc... me pierdo y, por ahora, no he
podido subir ningún fichero de la clase. Espero aclararme estos
días y tener listo la primera Reléase en &lt;a
href=&quot;http://www.luisyfernanda.net/
www.sourceforge.com&quot;&gt;www.sourceforge.com.&lt;/
a&gt;&lt;br  /&gt;&lt;br/&gt;&lt;br  /&gt;</description>
  <link>http://www.luisyfernanda.net/
index.php?op=ViewArticle&amp;articleld=17&amp;blogId=2</link>
  </item>
 </rdf:RDF>
```

El archivo anterior muestra las últimas noticias de la Web www.luisyfernada.net sobre PHP y proyectos en los que el autor está involucrado.

Si se fija, no es más que un archivo XML, con un formato determinado.

Puede verlo en la figura 15.1.

Tiene dos partes diferenciadas:

- Una parte de descripción del documento entre las etiquetas <channel>. En esta zona puede encontrar el título de Web donde están las noticias, la URL del archivo RSS y una descripción de la temática que aborda.
- Otra zona, entre los nodos <item>, que contiene todas las noticias que quiere exportar.

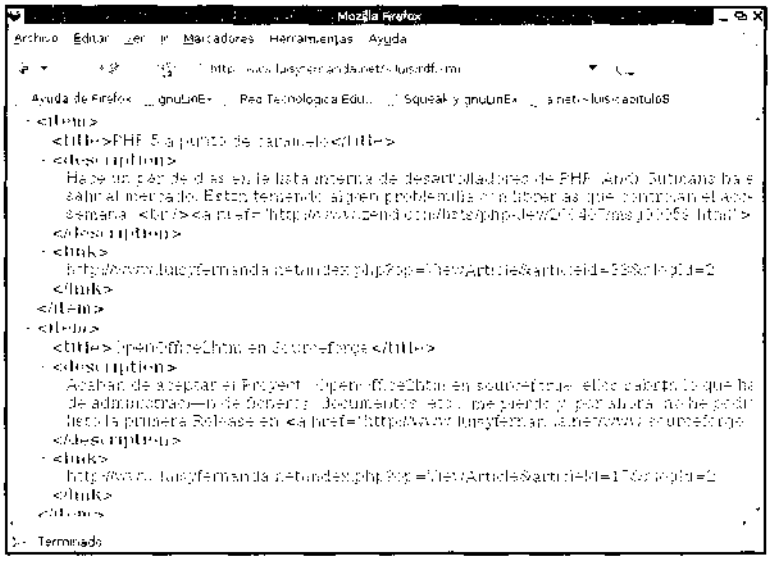



**Figura 15.1.** RDF de www.luisyfernanda.net.

## Leer un archivo RSS

En el capítulo anterior vimos cómo parsear un archivo XML creado por nosotros mismos. Esta vez, tenemos que extraer el contenido que nos interesa de un archivo con un formato determinado. Vamos a crear un pequeño objeto que sea capaz de insertar en nuestra Web las noticias de la página www.luisyfernanda.net. Por supuesto, vamos a utilizar SimpleXML por ser una de las herramientas estrella de PHP 5:

```
<?php
class SimpleRSS
{
 private $RSS;
 function __construct($fichero)
 {
  $this->RSS = simplexml_load_file($fichero);
 }
 public function parsea_item()
 {
  foreach ($this->RSS->item as $item)  {
            $this->parsea_titulo($item);
            $this->parsea_descripcion($item);
            $this->parsea_enlace($item);
  }
 }
 private function parsea_titulo($item)
```

```
    echo $item->title • "<br>";
    echo "<hr>";

  prívate function p,arsea_descripcion ($item)

    echo $item->descr.iption . "<br>";

  prívate function p#rsea_enlace($item)

    echo $item->link • "<br>";

  public function paísea_cabecera()

    &c^o "Titilo-. « . St^va,-^S.S,S->et^™^l->tlt"Le . "-JaxV-,
    echo "Web: " . $this->RSS->channel->link . "<br>";
    echo "<hr>";
   }
}
$RSS = new SimpleRSS("http://www.luisyfernanda.net/
rss.php?categoryId=3&blogId=2");
$RSS->parsea_cabecera();
$RSS->parsea_item();
?>
```

Si ejecuta el programa, tendrá como resultado las últimas noticias sobre PHP del portal www.luisyfernanda.net separadas por líneas horizontales. El método público `parsea_item()` es el encargado de extraer noticia a noticia y mandarlas al navegador. Este método hace 3 llamadas a 3 métodos distintos que se encargan de recuperar el título, el enlace a la noticia y la descripción.

Hemos separado los métodos para que cada lector personalice la forma de visualizar los distintos datos. Mi forma particular puede verla en la figura 15.2. Por ejemplo, el título podría mostrarlo en azul y el texto en gris, todo metido en una tabla-

## Escribir archivos RSS

Los archivos XML y en particular los archivos RSS pueden generarse de la misma forma que creamos un fichero de texto, añadiendo etiquetas con cierto criterio a un archivo de texto, pero la opción más correcta es utilizar DOM para generar nuestros propios documentos de noticias.

Como ya sabe, la extensión DOM permite añadir, borrar y modificar elementos a lo largo de un documento en cualquier orden.

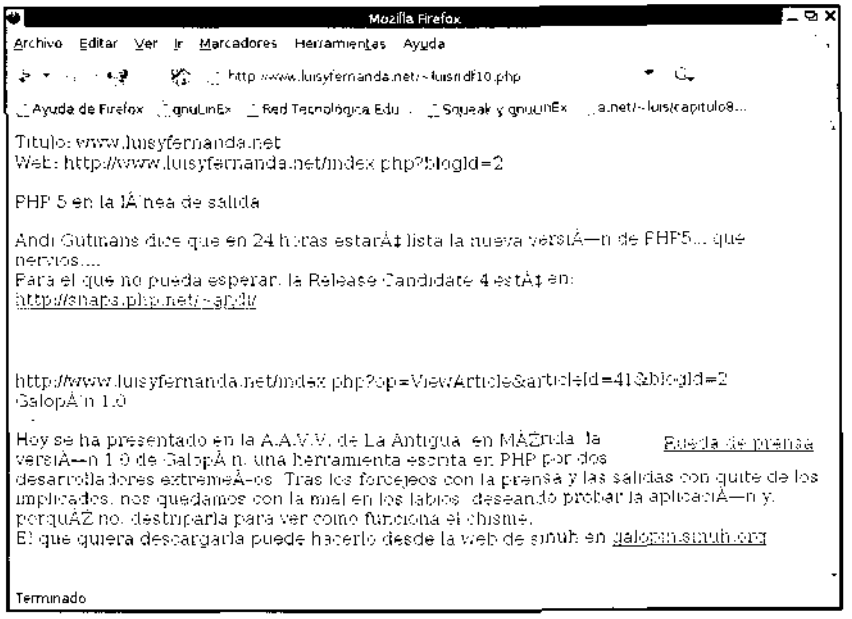



**Figura 15.2.** RDF parseado por SimpleRSS.

Para este cometido tenemos la siguiente clase, que nos va a generar documentos RSS a partir de cualquier fuente de datos.

```
<?php
class GeneraRSS
{
 prívate {documento;
 private $raiz;
 prívate $channel;
 private $titulo;
 private $enlace;
 private {descripción;
 private $item;
 function___construct()
 {
  $this->documento = new domdocument("1•0");
  $this->inicio();
 }
 private function inicio()
 {
  $this->raiz = $this->documento->createElement("rdf:RDF");
  $this->raiz->setAttribute('xmlns:rdf','http://www.w3.org/1999/
  02/22-rdf-syntax-ns#');
  $this->raiz->setAttribute('xmlns','http://purl.org/rss/1.0/');
  $this->raiz = $this->documento->appendChild($this->raiz);
```

```
}
public function canal($titulo,$enlace,$descripcion,$sobre)
{
  $this->channel = $this->documento->createElement("channel");
  $this->channel = $this->raiz->appendChild($this->channel);
  $this->titulo=$this->documento->createElement('title');
  $this->textotitulo=$this->documento->createTextNode($titulo) ;
  $this->titulo->appendChild($this->textotitulo);
  $this->channel->appendChild($this->titulo);
  $this->enlace=$this->documento->createElsment('link');
  $this->textoenlace=$this->documento->createTextNode($enlace);
  $this->enlace->appendChild($this->textoenlace);
  $this->channel->appendChild($this->enlace);
  $this->descripcion=$this->documento->createElement
  ('description') ;
  $this->textodescripcion=$this->documento->createTextNode
  ($descripcion);
  $this->descripcion->appendChild($this->textodescripcion;
  $this->channel->appendChild($this->descripcion);
}
public function item_nuevo($titulo,$enlace / $descripcion)
{
  $this->item = $this->documento->createEl<?ment('item');
  $this->item = $this->raiz->appendChild($this->item);
  $this->titulo=$this->documento->createElement('title');
  $this->textotitulo=$this->documento->createTextNode($titulo);
  $this->titulo->appendChild($this->textotitulo);
  $this->item->appendChild($this->titulo);
  $this->enlace=$this->documento->createElement('link');
  $this->textoenlace=$this->documento->createTextNode($enlace);
  $this->enlace->appendChild($this->textoenlace);
  $this->item->appendchild($this->enlace);
  $this->descripcion=$this->documento->createElement
  ('description') ;
  $this->textodescripcion=$this->documento->createTextNode
  ($descripcion);
  $this->descripcion->appendChild($this->textodescripcion);
  $this->item->appendChild($this->descripción);
  $this->items[]=$this->item;
}
public function grabar($fichero)
{
  $this->documento->save($fichero);
}
}
^documento = new GeneraRSS () ;
^documento->canal ("www.luisyfernanda.net" , "http://www.luisy
fernanda.net", "La Web del auto","");
$doc\imei\to->itev?1_YY\ievo VMaiwial Imprescindible, de PHP 5","http-.//
viww.luisyfernanda.net", "Ya a la venta");
```

```
$documento->item_nuevo("Nueva reléase de PHP 5","http://
www.iuisyfernanda.net","La nueva reléase corrige algunos fallos");
Sdocumento->grabar("miRSS.xml");
?>
```

La clase anterior se puede guardar en un archivo independiente y utilizar cuando sea necesario en el transcurso de un proyecto.

Aunque no implementa todas las especificaciones de RSS 1.0 puede utilizarse para generar noticias de nuestra base de datos.

**Nota:**

*Si desea modificar ia ciase para que contemple todas las opciones de Ja especificación RSS 1.0, puede visitar ia Web http:llpurt.orgl rss/1.0/.*

En el capítulo anterior, vimos cómo se puede generar un archivo XML con una sucesión de llamadas a métodos. Esta vez, hemos encapsulado todas las llamadas en varios métodos que se encargan de generar la parte de descripción del RSS y, por otro lado, las noticias.

El método `createElement ()` se encarga de crear nuevos nodos para después encadenarlos con `appendChild ()` a los nodos padre. Para crear un nodo de texto, sin embargo, tenemos que utilizar el método `createTextNode ()`; una vez creado el nodo de texto tenemos que asociarlo a un nodo-elemento (una etiqueta).

El último paso es hacer una llamada al método `save ()` con el nombre del fichero como parámetro para guardar el archivo RSS.

## Servicios Web XML-RPC

El trabajo como programador puede llevarle a desarrollar aplicaciones para distintos Sistemas Operativos y en distintos lenguajes de programación.

Si, en algún momento, necesita comunicar dos programas que están funcionando en máquinas distintas (un servidor Windows y otro gnuLinux) y escritos con lenguajes diferentes, puede que XML-RPC sea la solución perfecta.

El mecanismo permite que el ordenador cliente pueda acceder a los métodos del ordenador servidor y ejecutar rutinas almacenadas remotamente.

Este apartado muestra algún ejemplo de cómo implementar y explotar un servicio Web con la librería de clase IXR creada por Simón Willison bajo el auspicio de *Open Source.*

Una llamada XML-RPC consiste en dos partes: una pregunta y una respuesta.

Cada una de las partes es un archivo XML que contiene los parámetros solicitados al servidor o los datos devueltos.

Una pregunta al servidor puede ser como la que sigue:

```
<methodCalI>
   <methodName>conterit.getNotícias</methodName>
   <params>
        <param>
             <valuexint>23</int><value>
        </param>
   </params>
 </methodCall>
```

Como puede ver, es un archivo XML con una definición determinada. La etiqueta <methodName> contiene el nombre del método que debe llamarse en el servidor y <param> contiene un parámetro que se le pasará al método de la llamada.

El documento anterior será transmitido vía HTTP mediante POST al servidor de XML-RPC, que parseará el documento, ejecutará la función y devolverá un documento similar con los datos de la respuesta.

## Clase IXR

IXR es un conjunto de clases que implementa un servidor y un cliente de XML-RPC.

> **Nota:**
>
> *Puede encontrar la librería en la Web http://scripts.iricutio.com/xmlrpc/ o en la página del libro en http://unuzu.nnai/amulti-media.com.*

Lo mejor de la programación orientada a objetos es que no necesita saber cómo funcionan las clases que utilice, sólo conocer el uso correcto de los métodos. Así, puede utilizar la clase IXR de la manera que vamos a ver, sin conocer siquiera el formato de los archivos XML-RPC.

## Cliente XML-RPC

Los servicios XML-RPC tienen una dirección Web con una página Web escrita en PHP. Si escribe la dirección completa en el navegador se producirá un error, porque la Web esperará que mande mediante POST un archivo XML. Para utilizar sus servicios Web, o los servicios Web de alguna empresa como UserLand tiene que utilizar un cliente XML-RPC escrito, en este caso, en PHP 5. En este caso vamos a utilizar la clase IXR__Client de la librería IXR. El ejemplo muestra cómo utilizar el cliente en varios pasos:

- Crear un objeto cliente con la URL pasada como parámetro.
- Usar el método query () para hacer la consulta al servicio Web.
- Extrae el resultado con el método getResponse () .

Lo primero que vamos a hacer es implementar un pequeño objeto para recuperar la fecha de un servidor remoto:

```
<?php
require_once('IXR_Library.inc.php');
class XMLRPCFechacliente {
   private $cliente;
   function___construct($url,$debug=false) {
     $this->cliente= new IXR__Client ($url) ;
     $this->clíente->debug=$debug;
   }
   function getFecha($argumentol)
   {
     if (!$this->cliente->query('fecha.getFecha',$argumentol)) {
     trigger_error($this->cliente->getErrorCode() .
     ' : '.$this->cliente->getErrorMessage());
     return false;
     }
   return $this->cliente->getResponse () ;
   }
}
?>
```

La clase XMLRPCFechacliente implementa dos métodos. El constructor crea un objeto cliente IXR que recibe el servidor de servicios Web. El método getFecha () tramita la petición de la fecha con el servidor y obtiene una respuesta que devuelve a la página Web. Si todo va bien, el servidor enviará la fecha del lugar donde esté.

La llamada a query () tiene como parámetro primero el nombre del método que debe ejecutar el servidor y, como segundo parámetro, un argu-

mentó que puede pasarse. La Web que quiera mostrar esta información debe crear el objeto anterior más o menos como se muestra en el siguiente ejemplo:

```
<?php
require_once('XMLRPCFechacliente.php');
$servidor='http://www.luisyfernanda.net/Fecha.php';
$fechaRemota= new XMLRPCFechacliente($servidor) ;
echo $fechaRemota->getFecha("Hoy");
?>
```

Si evalúa el código en tan sólo 4 líneas, puede recibir cualquier tipo de información que el servidor tenga. Para ello debe crear una variable que contenga el nombre del servidor y la página PHP que contiene el programa de respuesta. La instanciación del objeto XMLRPCFechacliente y la llamada al método getFecha () es suficiente para desencadenar la conexión con el servidor.

## Servidor XML-RPC

En el apartado anterior ha visto que puede utilizar servicios Web ofrecidos por otras empresas para averiguar la hora de un País remoto, el cambio de moneda, el tiempo o, incluso, el catálogo de libros de una conocida tienda *on-line*. Pero lo realmente interesante es poder crear sus propios servicios Web.

La clase IXR_Server hace la mayoría del trabajo por usted, que puede emplear el tiempo en inventar nuevos servicios Web.

Hemos creado una clase que implementa el servicio getFecha (), que el objeto cliente utiliza.

```
<?php
require_once('IXR_Library.inc.php');
class XMLRPCFechaservidor extends IXR_Server
{
   function __construct()
   {
      $this->IXR_Server(array('fecha.getFecha' => 'this:getFecha'));
   }
   function getFecha($argumentol)
   {
      if ($argumentol=="Hoy") {
         $fecha = "La fecha remota es:" . date("d-m-Y");
      } else {
         $fecha = "Especifique un dia";
      }
```

```
        return $fecha;
        }
    }
?>
```

La clave para entender el funcionamiento del servidor está en el constructor. Aquí se crea un *array* con el nombre de los métodos que se pueden utilizar desde fuera.

El primer valor, `fecha.getFecha`, es el nombre que el cliente debe emplear para recuperar la Fecha del Servidor y el segundo valor, `this.getFecha`, es el método perteneciente al servidor que se ejecutará cuando reciba una pregunta a esa función.

La figura 15.3 muestra el dato que obtiene el cliente.


**Figura 15.3.** Fecha obtenida mediante XML-RPC.

La definición del método que envía la fecha se hace como ya estamos acostumbrados.

Lo que hemos visto es sólo la definición de la clase, pero necesitamos una página Web de apoyo que instancie el objeto y lo ejecute. Puede ser tan sencillo como:

```
<?php
require_once('XMLRPCFechaservidor.php');
$servidor= new XMLRPCFechaservidor();
?>
```

## Usos de XML-RPC

Los ejemplos que hemos visto no son especialmente excitantes, pero dan una idea de lo sencillo que es exportar cualquier tipo de información. La verdadera potencia de los servicios Web es que permiten disfrutar de nuevas funcionalidades de las que no disponemos. Podríamos crear un servicio Web que permitiera a usuarios de PHP 4 utilizar nuestro objeto de SimpleXML, pero esto lo dejo en manos del lector.

# Resumen

XML se ha convertido en los últimos años en un estándar libre para el intercambio de información. Incluso Microsoft, poco amiga de los archivos abiertos, está implementando servicios en XML con su plataforma .NET.

Después de leer este capítulo, podrá crear elegantes páginas que contengan noticias relevantes sobre cualquier tema y exportarlas a otras páginas amigas gracias a la tecnología RSS.

En cuanto a la creación de servicios Web, existe una controversia entre dos desarrollos principales: XML-RPC y SOAP de Microsoft. SOAP *(Simple Object Access Protocol),* es una implementación muy extensa de Microsoft para la creación de servicios Web. Pese a que su definición aboga por la simpleza (Protocolo Simple), la paradoja es que es mucho más complicada de utilizar que XML-RPC y por eso se escapa de los objetivos de este libro.

Una de las nuevas características de PHP 5 es la inclusión de una extensión para la generación de servidores y clientes SOAP, pero todavía está en fase experimental.

El futuro cercano puede estar en SOAP, así que recomiendo al lector visitar algunas páginas Web sobre el tema para comenzar a entender su funcionamiento.


Capítulo 11

# Generación de gráficos con PHP 5

**En este capítulo aprenderá a:**

- Crear gráficos estadísticos desde PHP 5.
- Crear imágenes en miniatura.
- Añadir marcas de agua personalizadas a sus imágenes.
- Crear gráficos estadísticos profesionales con la librería JpGraph.

## Introducción

La construcción de un sitio Web no sólo debe contar con textos, bases de datos y funciones escritas en PHP para manejar los datos. Existe un gran abanico de posibilidades que pueden dotar a la página de contenido multimedia (imágenes v sonidos) y que, además, pueden ser procesados de alguna forma por PHP.

Entre las funciones más básicas de interactuación con las imágenes, PHP puede realizar gráficos de barras para mostrar resultados estadísticos, añadir una marca de agua a todas nuestras fotografías o mostrar una réplica de las imágenes en un formato más reducido para poder seleccionar entre un conjunto de ellas.

La mayoría de los ejemplos utilizan la librería gráfica GD para PHP, que puede instalarse fácilmente en los tres Sistemas Operativos más comunes (gnuLinux, MacOSX y Windows).

## Gráficos HTML

Antes de entrar de lleno en las maravillas que ofrece la librería gráfica GD, vamos a realizar un pequeño ejemplo para aprovechar las posibilidades del lenguaje HTML para la creación de estadísticas gráficas. Esta posibilidad nos permite realizar estudios gráficos básicos sin tener que instalar ningún paquete adicional a PHP.

Puede decantarse por este método o por utilizar la librería, dependiendo de la complejidad de su proyecto.

Vamos a crear como ejemplo un sistema de votación electrónico que cuente los votos de cada participante y los almacene en una base de datos. Vamos a utilizar el gestor SQLite, visto en el capítulo 11. Para tal fin hemos creado una clase de conexión a SQLite muy parecida a la clase de conexión de MySQL.

```
<?php
class Servidor_Base_Datos
{
  prívate $base_datos';
  prívate $descríptor;
  prívate ^resultado;
  function___construct($base_datos)
```

```
  {
       $this->servidor = $servidor;
       $this->usuario = $usuario;
       $this->pass = $pass;
       $this->base_datos = $base_datos;
       $this->conectar_base_datos();
  }
  prívate function conectar_base_datos( )
  {
       $thís->descriptor = sqlite_open($this->base_datos) ;
  }
  public function consulta($consulta)
  {
       $this->resultado = sqlite_query($consulta, $this-
>descriptor);
  }
  public function extraer__registro ()
  {
       if ($fila = sqlite_fetch_array($this-
>resultado,SQLITE_ASSOC)) {
            return $fila;
       } else {
            return false;
       }
  }
}
?>
```

Los métodos son prácticamente iguales y la forma de utilizar el objeto es exactamente igual que en los ejemplos del capítulo 12.

Los datos que vamos a manejar en la encuesta son parte de una pequeña tabla que almacenará el nombre del Lenguaje de Programación que preferimos y el número de votos que le hemos dado.

Podemos inicializar la base de datos con el siguiente *script:*

```
<?php
requirejonee("clase_SQLite.php");
$base_datos = new Servidor_Base_Datos("votaciones.db");
$base_datos->consulta("créate table lenguaje (id_lenguaje INT
PRIMARY KEY, lenguaje CHAR(255), votos INT)");

$base_datos->consulta("insert into lenguaje (lenguaje,votos) valúes
('PHP',0)");
$base_datos->cónsulta("insert into lenguaj e (lenguaj e,votos) valúes
('JSP',€)");
$base_datos->consulta("insert into lenguaje (lenguaje,votos) valúes
('ASP',0)");
```

```
$base_datos->consulta("insert into lenguaje (lenguaje,votos) valúes
('COLD FUSIÓN',0);");
$base_datos->consulta("select * from lenguaje");

while ($fila = $base_datos->extraer_registro()) {
   echo $fila["lenguaje"] . ": " . $fila["votos"] . "<br>";
}
?>
```

Todo debería serle familiar. Lo que hacemos es crear una tabla nueva en la base de datos e insertar los lenguajes objetivo de la encuesta. Por último, hacemos una comprobación para ver si los datos están almacenados en la base de datos.

Lo siguiente que vamos a ver es una Web con un formulario para elegir el lenguaje que más nos agrada. En la zona inferior de la página tenemos un contador que nos indica el número de votos.

```
<?php

//Primero comprobamos si tenemos un voto

$base_datos = new Servidor_Base_Datos("votaciones2.db");

//Si hemos votado sumamos el valor al número de votos almacenado

if (isset($_POST["voto"])) {
          $voto = $_POST["voto"];
          $base_datos->consulta("select votos from lenguaje where
          id_lenguaje='$voto'");
          $numero_votos = $base_datos->extraer_registro();
          $numero_votos = $numero_votos["votos"];
          $numero_votos++;
          $base_datos->consulta("update lenguaje set votos=
          '$numero_votos' where id_lenguaje = '$voto'");
}
?>
<html>
<body>
¿Qué lenguaje de programación prefieres?

//Formulario que permite votar

<form action="voto.php" method="post">
<select name="voto">
<?php
$base_datos->consulta("select * from lenguaje");
while ($fila = $base_datos->extraer_registro()) {
       $id_lenguaje = $fila["id_lenguaje"];
       $lenguaje = $fila ["lenguaje"];
```

```
        echo "<option value=\"$id_lenguaje\">$lenguaje</option>";
}
?>
</select>
<input type="submit" name="Enviar" value="Enviar" >
</form>
<table>
<?php
$base_datos->consulta("select * from lenguaje");
while ($fila = $base_datos->extraer_registro() ) {
        $lenguaje = $fila ["lenguaje"];
        $votos = $fila["votos"] ;
        echo "<trxtd>$lenguaje </tdxtd> $votos votos</td></
tr>" ;
}
?>
</table>
</body>
</html>
```

Como puede ver, es un mecanismo muy sencillo de entender. La parte principal es el formulario que permite elegir entre los 4 lenguajes de *script.*

Si enviamos un voto, la misma página recibirá el dato y actualizará la base de datos, mostrando posteriormente los votos que tienen los lenguajes. En la figura 16.1 puede ver el resultado.

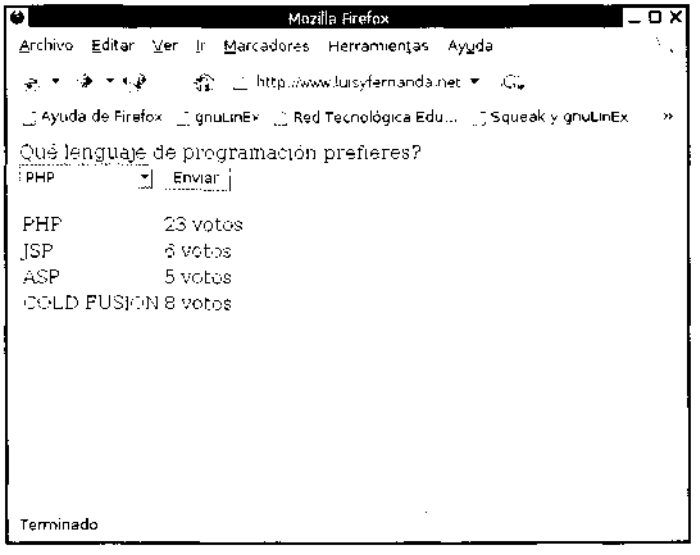



**Figura 16.1.** Votación electrónica.

## Gráficos de barras

Después de todo este código, estará preguntándose por las imágenes generadas automáticamente. Todavía no hemos hecho nada de eso, pero en el código siguiente tiene la clave de este apartado. Básicamente jugamos con 4 imágenes: rojo.png, verde.png, azul.png y amarillo.png. Cada imagen tiene un tamaño de 1 píxel de ancho y alto. El truco está en hacer un cálculo porcentuado de los valores de votos con respecto al que mayor votos tiene.

De ahí sacamos el ancho de la imagen. Una vez encontrado este ancho se le pasa a la propiedad width de la etiqueta <img> y obtenemos un gráfico con imágenes de distintos anchos, es decir, obtenemos un gráfico de barras estadístico.

El código puede ser como el siguiente:

```
<?php
$imagenes =
array("rojo.png","azul.png","verde.png","amarillo.png");
$base_datos->consulta("select * from lenguaje");
while ($fila = $base_datos->extraer_registro() ) {
        $id_lenguaje = $fila ["id_lenguaje"] ;
        $lenguaje = $fila ["lenguaje"] ;
        $votos = $fila["votos"];
        $ancho = dame_ancho($votos,$id_lenguaje);
        $imagen = $imagenes[$id_lenguaje-1];
        echo "<trxtd>$lenguaje - $votos votos</tdxtdximg
        src=\"$imagen\" width=\"$ancho\" height=\"20\"></tdx/tr>";
}
function dame_ancho($votos)
{
        $base_datos2 = new Servidor_Base_Datos("votaciones2.db");
        $base_datos2->consulta("select max(votos) as máximo from
        lenguaje");
        $numero_votos = $base_datos2->extraer_registro();
        $maximo_votos = $numero_votos["máximo"];
        return (intval($votos)*300)/intval($maximo_votos);
}
?>
</table>
```

Antes generábamos una tabla con los lenguajes y el número de votos. Ahora generamos los lenguajes y calculamos con la función dame_ancho () el porcentaje con respecto al mayor valor de cada uno de los votos de los lenguajes.

La figura 16.2 muestra los gráficos de barras de la votación electrónica.

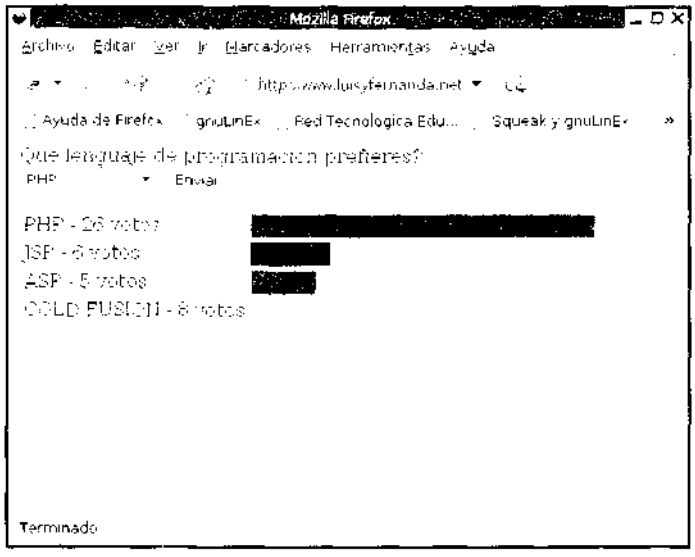



**Figura 16.2.** Votación electrónica con gráficos.

# Librería GD

La librería gráfica GD está escrita en lenguaje C y permite crear y manipular gráficos fácilmente. Permite importar y exportar gráficos de distinto tipo (GIF, JPG y PNG). La filosofía de los archivos GIF y PNG es casi la misma. Estos dos archivos guardan en memoria el conjunto completo de píxeles con su color concreto y comprimen el resultado para que el archivo final no sea muy pesado. Los archivos JPG también están comprimidos, pero se basan en algoritmos complejos que permiten un mayor rendimiento y la hacen muy propicia para usarse con fotografías.

## Tipos MIME

El estándar MIME se concibió originariamente para identificar los diferentes contenidos que podemos enviar por correo electrónico. Actualmente se utiliza para describir el tipo de archivo que enviamos a través de Internet. Es importante conocer los tipos MIME para trabajar con las librerías GD.

En general, antes de recibir la respuesta del servidor, éste debe enviarnos una cabecera anunciando el tipo de contenido que vamos a recibir usando una cabecera especial: `Content-Type`. PHP, por defecto, envía el tipo

MIME text/html (documento HTML). Cuando utilice la librería GD necesitará enviar una cabecera con el tipo de la imagen:

```
<?php
header("Content-Type: image/png");
?>
```

**Tabla 16.1.** Tipos MIME.

| Formato de imagen | MIMÉ |
|---|---|
| JPG | ¡mage/jpeg |
| PNG | image/png |
| GIF | image/gif |
| BMP | image/bmp |
| SVG | image/xml+svg |

## Mostrar una imagen en pantalla

Normalmente, utilizamos la etiqueta <img> de HTML para mostrar una imagen en un lugar determinado de nuestra página Web.
La propiedad src debe indicar una imagen determinada con formato conocido. La línea siguiente mostrará en pantalla el logotipo de php.

```
<img src="php.png">
```

La librería gráfica GD también permite enviar imágenes con el uso de unas funciones determinadas. Como ejemplo vamos a ver cómo mostrar el mismo logotipo, esta vez utilizando GD.

```
<?php
$imagen = imagecreatefrompng("php.png");
header("Content-Type:  image/png");
imagepng($imagen);
?>
```

**Advertencia:**

*La etiquete* <img> *de nuestra página Web debe apuntar ahora al archivo PHP que genera la imagen:* <img src="imagen.php">.

La función imagecreatefrompng () recupera una imagen de un archivo y crea un descriptor en la variable $imagen. Para mostrar la imagen,

tiene que enviar la cabecera del tipo de archivo gráfico que quiere mostrar y después enviar la imagen con imagepng () .

A simple vista parece demasiado esfuerzo para algo que resuelve perfectamente la etiqueta <img>, pero examine el siguiente código:

```
<?php
$imagenes = array("php.png","zend.png","paris.png");
//Semilla aleatoria
srand( (double)microtime 0*1000000) ;
$aleatorio = rand(0,2);
$fichero = $imagenes[$aleatorio];
$imagen = imagecreate'f rompng ($f ichero) ;
header("Content-Type: image/png");
imagepng($imagen);
?>
```

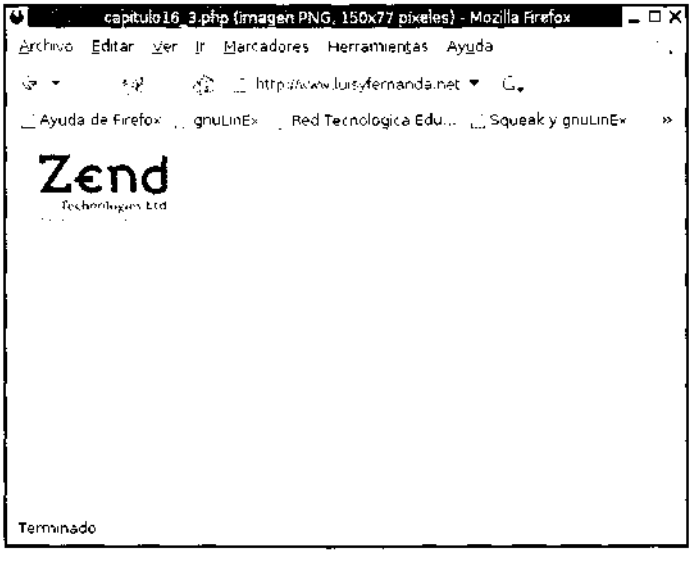



**Figura 16.3.** Elección aleatoria de imágenes.

El *array* $imágenes contiene varios archivos gráficos en PNG. Generamos un número aleatorio y recuperamos la imagen que ocupa la posición del número.

Los siguientes pasos son exactamente iguales a los vistos anteriormente. El *script* muestra una imagen distinta cada vez que lo ejecutamos y puede servir para crear una Web con distintos banners aleatorios.

Puede ver el resultado en la figura 16.3.

## Crear imágenes en miniatura

Si su página Web permite enviar fotografías, quizá necesite mostrar un gran número de ellas en una sola página. HTML permite poner un ancho determinado a la imagen, aunque el tamaño permanecerá invariable. Lo ideal es crear dos versiones de cada fotografía. Una en miniatura con poco peso en *bi/tes y* otra, la original, que puede ser mostrada cuando se haga clic en la imagen pequeña.

Gracias a la librería GD podrá modificar el tamaño de las fotografías que quiera. Vamos a crear una clase para generar fotografías con la mitad de tamaño:

```
<?php
class imagen
{
  prívate    $archivo;
  prívate    $imagen;
  prívate    $dimensiones;
  function____construct($archivo)
  {
            $this->archivo = $archivo;
            $this->imagen = imagecreatefromjpeg($archivo);
            $this->dimensiones = getimagesize ($archivo) ;
  /
  public function imagen_original()
  {
            header("Content-Type: image/jpeg");
            imagejpeg{$this->imagen);
  }
  public function imagen_miníatura()
  {
            $dimensionx = $thís->dimensiones[0] /2;
            $dimensiony = $this->dimensiones[1] /2;
            $this->miniatura = imagecreatetruecolor($dimensionx,
            $dimensiony);
            imagecopyresampled($this->miniatura, $this->imagen,
            0, 0, 0, 0, $dimensionx, $dimensiony, $this-
            >dimensiones[0], $this->dimensiones[1]);header
            ("Content-Type: image/jpeg") ;imagejpeg($tnis-
            >miniatura);
  }
}
$fotografia = new imagen($_GET["fotografía"]);
if ($_GET["modo"]=="original")
{
  $fotografia->imagen_original();
```

```
} else {
  $fotografia->imagen_miniatura();
}
?>
```

La clase está especializada en construir imágenes con la mitad de tamaño. Para que funcione tiene que hacer una llamada a la url clase_imagen.php, pasando como parámetros GET el nombre de la fotografía y el modo (tamaño original o miniatura).

El constructor de la clase rescata el archivo gráfico y averigua las dimensiones X e Y con la función getimagesize().

El método imagen_original() funciona exactamente igual que el ejemplo anterior. El método imagen_minuatura() divide entre 2 las dimensiones capturadas de la imagen y crea un entorno de trabajo con esas dimensiones con la función imagecreatetruecolor(). En este momento, la variable $miniatura del objeto tiene un espacio de memoria asignado para almacenar un gráfico, pero está vacío. Lo que hacen las líneas siguientes es redimensionar y copiar la imagen original a nuestro entorno de trabajo. La función imagecopyresampled() copia la imagen guardada en el segundo parámetro en el espacio de trabajo del primer parámetro.

Los argumentos 3 y 4 son las coordenadas X e Y donde empezará a pegarse la imagen copiada y el 5 y 6 las coordenadas desde las que se empezará a copiar la imagen original. Los siguientes parámetros son el ancho y alto de las imágenes destino y origen.

Una vez copiada la imagen al espacio de trabajo, ya podemos mostrarla enviando la cabecera y la imagen con la función imagejpg().

Vamos a crear una Web que haga uso de esta clase:

```
<html>
<body>
<form action="imagen.php" method="POST" enctype="multipart/
form-data">
Elige un archivo de imagen jpg: <input type="file"
name="fichero">
<input type="submit" name="Enviar" value="Enviar">
</form>
<?php
if (isset($_FILES["fichero"])) {
          echo "Imagen Original:<br>";
          $fotografia = $_FILES["fichero"]["tmp_name"];
          copy($fotografia,$_FILES["fichero"]["ñame"]);
          $foto_copia = $_FILES["fichero"]["ñame"] . "&modo=
          original";
```

```
            $url = "clase_imagen.php?fotografia=$foto_copia";
            echo "<img src=\"$url\">";
            echo "Imagen Miniatura:<br>";
            $foto_copia = $_FILES["fichero"] ["ñame"] . "&modo=
            miniatura";
            $url = "clase_imagen.php?fotografia=$foto_copia";
            echo "<img src=\"$url\">";
}
?>
</body>
</html>
```

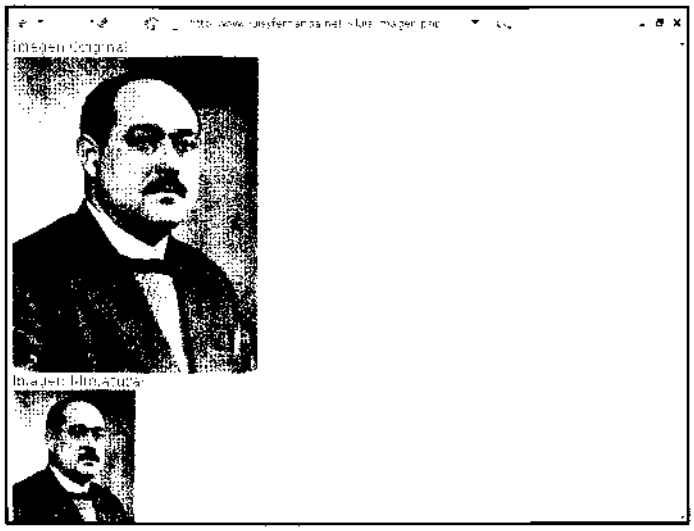


**Figura 16.4.** Cambio de tamaño de imágenes.

La página Web muestra un formulario que nos da la posibilidad de enviar al servidor una fotografía de nuestro ordenador.

Si le damos a enviar la página nos mostrará la fotografía en su versión original y en su versión reducida, como puede comprobar en la figura 16.4.

## Generar una marca de agua

Otro problema común sencillo de resolver es la creación de marcas de agua sobre las fotografías que ofrecemos en nuestra Web. La marca de agua puede ser un gráfico o un texto que insertamos encima de la fotografía original.

Una forma sencilla de implementar marcas de agua es añadiendo un método nuevo a la clase:

```
prívate function marca_agua()
{
   $this->marca_agua = imagecreatefrompng{"php.png");
   $dimensiones = getimagesize("php.png");
   $posicionx = $this->dimensiones[0] - $dimensiones[0] ;
   $posiciony = $this->dimensiones[1] - $dimensiones[1] ;
   imagecopy($this->imagen, $this->marca_agua, $posicionx,
   $posiciony, 0, 0, $dimensiones[0] , $dimensiones[1] ) ;
}
```

Podemos llamar al método desde el constructor para que la imagen guarde la información de la fotografía grabada e inserte la marca en alguna parte de la misma. También podemos generar un texto como marca de agua.

Vamos a añadir un nuevo método a la clase para poder marcar las fotografías:

```
prívate function marca_texto()
{
   $color = imagecolorallocate($this->iraagen, 255,0,0);
   ímagestríng($thís->imagen, 4, 0, 0, "Hecho con PHP5",
$color);
}
```

La figura 16.5 muestra la inclusión de una marca de agua gráfica y otra marca de agua de texto.

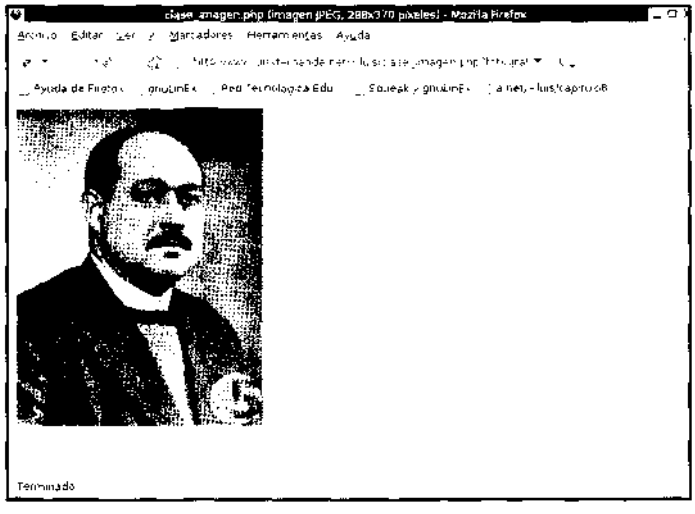



**Figura 16.5.** Marcas de agua.

La función imageallócate () crea un color con los componentes RGB como parámetros. La función devuelve un número, que identifica al color creado en la imagen. La función imagestring () imprime un texto de color determinado en la imagen. El segundo parámetro es el tipo de fuente que queremos introducir.

## Gráficos estadísticos profesionales con JpGraph

JpGraph es una librería escrita en PHP que permite crear todo tipo de gráficos estadísticos en dos y tres dimensiones. Hace uso de la librería gráfica GD en su versión 2. Se puede descargar de la página www.jpgraph.org.

JpGraph tiene dos licencias de uso. Una gratuita para proyectos no comerciales y con carácter educativo y otra de carácter comercial, de pago. Le recomiendo leer ambas licencias para comprobar que puede utilizarlas.

Para instalar la librería tiene que copiar el directorio con las clases en la ruta donde guarde su aplicación o en la ruta de búsqueda de PHP 5.

## Gráficos de barras

El ejemplo más sencillo es la creación de un gráfico con varios valores en el que podemos ver todos los valores interconectados entre sí por unas líneas.

```
<?php
require_once  ("jpgraph/jpgraph.php");
require_once  ("jpgraph /jpgraph_line.php");
$datosy = array(11,O,8,12,5,1,9,13,15,7) ;
// Creamos el gráfico
$grafico = new Graph(350,250);
$grafico->SetScale("textlin");
//Creamos una nueva linea de puntos
$puntos =new LinePlot($datosy);
$puntos->SetColor("blue");
//Añadimos los puntos al gráfico
$grafico->Add($puntos);
//Se muestra el gráfico
$grafico->Stroke();
?>
```

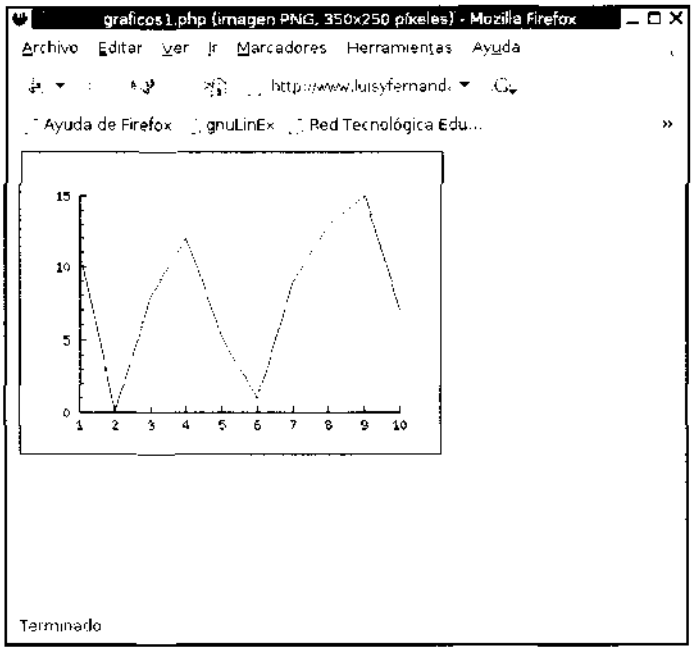



**Figura 16.6.** Gráfico generado con JpGraph.

Suponiendo que la librería esté guardada en la carpeta `jpgraph`, hacemos `unrequire_once ()` de la clase principal `jpgraph.php` y de la clase que utilizaremos para gráficos de línea `jpgraph_line.php`. La variable `$datosy` guarda diez valores que pueden identificar el número de visitas a nuestra Web o el número de ventas por meses de un artículo.

Para crear un gráfico tiene que crear el objeto Graph con el ancho y alto que quiere que tenga en la Web. Lo siguiente es configurar la escala en la que aparecerá el gráfico. Lo mejor es echar un vistazo al manual para elegir el que más le convenga en cada momento.

Seguidamente cree un conjunto de puntos con el método `Lineplot ()` con los valores del *array* `$datosy`. Puede elegir también el color de la línea resultado con el método `SetColor ()`.

Para finalizar, añada los valores al gráfico y muéstrelo con `Stroke ()`.

Con estas líneas hemos conseguido crear el gráfico de la figura 16.6.

Puede añadir algunas características más a sus gráficos como títulos en los ejes, colores o varias líneas de valores para comparar datos.

Para añadir títulos a los gráficos le pueden servir las líneas siguientes:

```
$grafico->title->set("Gráfico JpGraph"),-
```

```
$grafico->xaxis->title->set("dia");
$grafico->yaxis->title->set("visitas");
```

El método `title` genera un título asociado al objeto inmediatamente anterior. Si el objeto es `$grafico`, el título será el de la imagen completa. Si el objeto anterior es el eje x o el eje y (`xaxis` o `yaxis`), el título corresponderá con cada uno de los ejes. Puede ver el resultado en la figura 16.7.

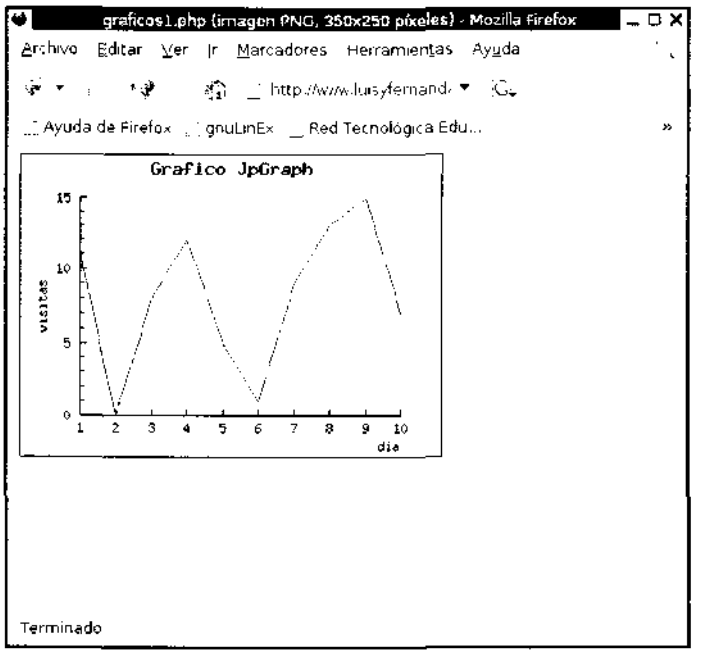



**Figura 16.7.** Gráfico generado con JpGraph con títulos.

Además, puede añadir tantos datos como quiera para hacer un estudio comparativo de varios datos. Para hacer esto debe repetir algunos pasos, pero con distintos valores:

```
$datosy2  =  array(1,2,3,7,8,5,4,3,12,7);
$puntos2  =  new LinePlot($datosy2);
$puntos2->SetColor("red");
$grafico->Add($puntos2);
$puntos->SetLegend("Europa");
Spuntos2->SetLegend("America");
$grafico->legend->Pos(0.05,0.5,"right","center");
```

Como vimos en el ejemplo anterior, podemos añadir otro *array* lleno de valores, crear una nueva línea de puntos con un color determinado y asociarlo, finalmente, con el gráfico que estamos creando, como puede ver en

la figura 16.8. Las últimas 3 líneas añaden una leyenda en la parte derecha del gráfico, es decir, un pequeño gráfico explicativo de las líneas de datos.

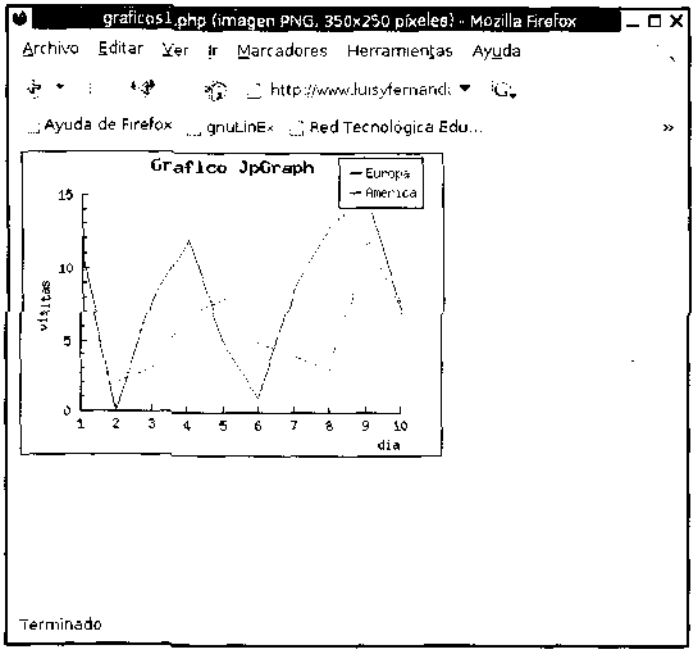



**Figura 16.8.** Comparación de varios gráficos.

## Gráficos en 3D

JpGraph realiza casi cualquier tipo de gráfico estadístico. Uno de los más utilizados es el gráfico de "quesito", donde podemos evaluar por porciones el dato de mayor amplitud sólo con un vistazo. El ejemplo siguiente muestra cómo hacer un gráfico de este tipo:

```
<?php
include ("jpgraph /jpgraph.php");
include ("jpgraph /jpgraph_pie.php");
include ("jpgraph /jpgraph_pie3d.php");

$datos = array(4 0,So,21, 33) ;

$grafico = new PieGraph(300,200) ;
$grafico->SetShadow()•

$grafico->title->Set("Gráfico de quesito");
$pl = new PiePlot30($datos);
$pl->SetAngle(20);
```

```
$pl->SetSize(0.5);
$pl->SetCenter(0.45);
$pl->SetLegends($gDateLocale->GetShortMonth());

$grafic0->Add($pl);
$grafic0->Stroke();
```

Para crearlo primero tiene que incluir los archivos jpgraph_pie.php y jpgraph^pie3d.php, que permitirán instanciar el tipo de gráfico PiePlot3D. *El* método SetShadowC) crea una sombra en eí contorno exterior del gráfico. La instanciación del gráfico circular se hace sobre la variable $pi, que empieza a tomar forma con las propiedades SetAngle() y SetCenter(), que permiten definir el ángulo del gráfico con respecto a la vista frontal y el centrado del gráfico con respecto a la caja que lo contiene. La fase final es añadir el "quesito" al gráfico creado y mostrarlo en pantalla. Una mejora del gráfico anterior puede ser resaltar la porción del gráfico que tiene mayor valor y separarlo notablemente. Esto puede hacerse añadiendo la línea siguiente:

```
$pl->ExplodeSlice(1);
```

El gráfico de quesito se muestra en la figura 16.9.

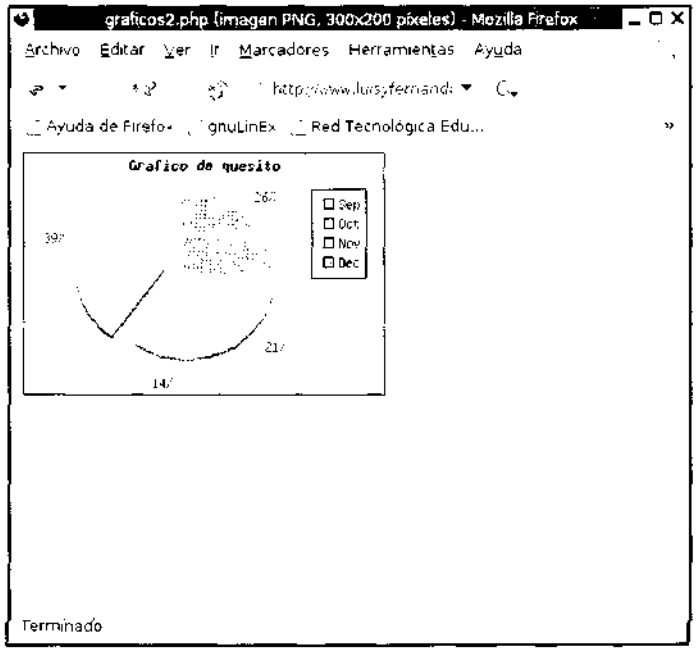
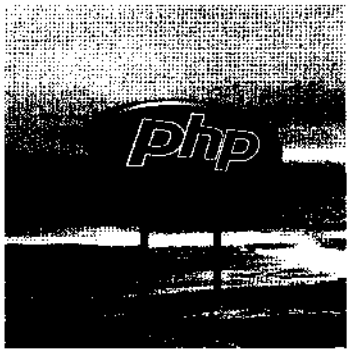



**Figura 16.9.** Gráfico de quesito.

## Resumen

El tratamiento de imágenes desde PHP 5 nos da la posibilidad de crear proyectos muy profesionales. Tanto la posibilidad de crear gráficos mas pequeños, marcas de agua o texto sobreimpresos, como la de generar estadísticas en tiempo real crean un potencial que no todas las páginas Web saben aprovechar.

Existen algunos proyectos, como K++, que permiten crear imágenes del tipo Flash y que merece la pena indagar en la búsqueda de contenido originales para nuestro portal.




Capítulo 17

# Gestión de errores en PHP 5

**En este capítulo aprenderá a:**

- Reconocer una excepción.
- Diferenciar entre los tres tipos de errores nativos de PHP 5.
- Crear excepciones dentro de un bloque Try / Catch.
- Resolver errores sin excepciones.
- Almacenar los errores producidos en un fichero.

# Introducción

Si está familiarizado con otros lenguajes de programación como C o Java, ya conocerá algunos mecanismos que permiten manejar errores y excepciones. Una de las nuevas características que PHP 5 incluye, es un objeto de manejo de excepciones con una sintaxis muy similar a Java.
El manejo de excepciones es una herramienta muy potente que podrá apreciar una vez que comprenda el concepto y haga buen uso del objeto en su código. Estas funciones permiten depuíar condiciones de error, recuperar situaciones inesperadas y presentar *interfaces* limpias de errores.

# Errores y Excepciones

Lo primero que tenemos que hacer es no pensar en una excepción como si fuera un error. Una excepción es una condición que se experimenta en un programa inesperadamente y no puede ser manejada por el código que hemos escrito. Generalmente no causan la parada del programa, pero sí es posible detectarla para que continúe normalmente.
Las excepciones, si son tratadas correctamente, pueden dar más fiabilidad a su aplicación y librarle de muchos dolores de cabeza. Aunque, si no son correctamente utilizadas, pueden crear más problemas que resolverlos. Debe tomar un tiempo para determinar si necesita crear sus propios métodos de manejo de errores y, si es así, verá que a la larga resulta más gratificante. Como ejemplo vamos a ver una pieza de código que simula la recepción de un identificador de usuario y su posible comprobación en una base de datos. Tal y como puede ver, se comprueba que los tres primeros caracteres sean "usr", que el tamaño en caracteres sea al menos de 9 y que exista en una base de datos:

```
<?php
//extraemos el usuario por GET
$usuario_id = $_GET['usuario_id ' ] ;
//Si existe el usuario escribimos el mensaje CORRECTO
if  (!usuario_valido($usuario_id))  {
   echo "ERROR:: El usuario $usuario_id no existe";
} else {
   echo "CORRECTO:: El usuario $usuario_id existe";
}

function usuario_valido($usuario_id) {
```

```
    $cadena = "usr";
    //Comprobaciones de usuario
    if   ( (strstr ($usuario_id,$cadena) == false) [[
          (strpos($usuario_id,$cadena) != 0)) {
               return false;
          }

          if (strlen ($usuario__id) < 9) {
               return false;
          }
          if (existe__base_datos ($usuario_id) ) {
               return true;
          } else {
               return false;
          }
}
?>
```

Como puede ver, cualquier condición que está dentro de la función que valida los usuarios puede dar un resultado negativo y sólo uno de ellos dará un resultado positivo; si el usuario existe en la base de datos.

Las excepciones permiten distinguir entre distintos tipos de errores y manejarlos dependiendo de su naturaleza.

## La clase Exception

La clase `Exception` se ha creado en PHP 5 para manejar las anomalías producidas. En vez de utilizar comprobaciones que evalúan si es falso o verdadero una determinada condición, se utiliza una instancia de la clase `Exception`, que captura los errores y permite mostrar un mensaje de error personalizado.

```
<?php
//extraemos el usuario por GET
$usuario__id = $_GET[ 'usuario_id'] ;
//Si existe el usuario escribimos el mensaje CORRECTO
try {
   if ( lusuario_valido($usuario_id)) {
        echo "ERROR:: El usuario $usuario_id no existe";
   } else {
        echo "CORRECTO:: El usuario $usuario_id existe";
   }
} catch(Exception $ex) {
   //Devuelve el mensaje de error
   $mensaje = ($ex->qetMessage() ) ;
   echo $mensaje;
```

```
}


function usuario_valido($usuario_id)  {
   $cadena = "usr";
   //Comprobaciones de usuario
   if   ( (strstr ($usuario_id,$cadena)  == false)  |
         (strpos($usuario_id, $cadena)  != 0))  {
              throw   new Exception  ("$usuario_id no contiene el
prefijo");
         }

         if  (strlen($usuario_id)  < 9)  {

              throw new Exception  ("El tamaño de $usuario_id es
menor de 9 caracteres");
         }
         if  (!existe_base_datos($usuario_id) )  {
              return  false;
         } else {
              return  false;
         }
}
```

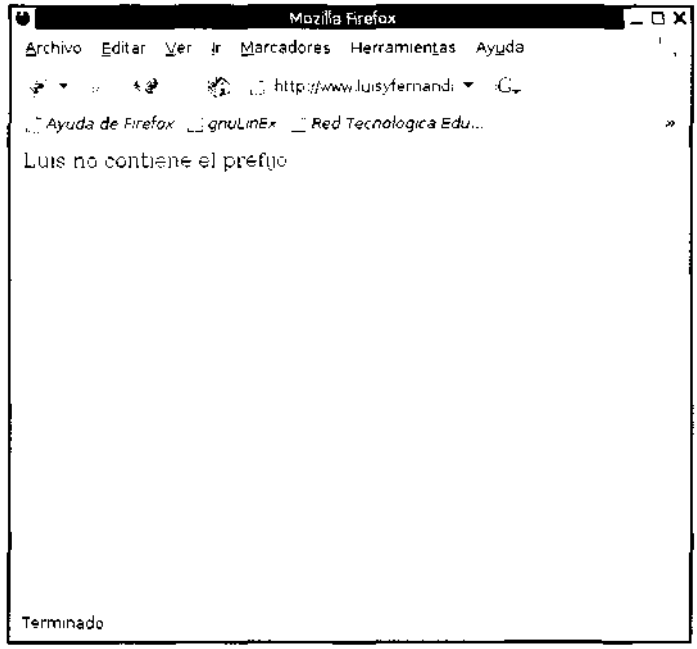



**Figura 17.1.** Gestión de errores con Try / Catch.

Como puede ver, las excepciones se crean de una manera algo diferente. En el momento en el que se escribe throw new Exception, los errores que no se controlen con el flujo normal del programa son utilizados como excepciones separadas del resto de la aplicación.

## Bloque Try / Catch

Las excepciones son controladas por el bloque try/catch. Todo el código que escriba y que pueda generar un error, se incluirá dentro de una estructura try. Si dentro del bloque try se encuentra un error, la ejecución del código se para y se pasa el control al bloque catch. Este bloque se consuita para encontrar *la* excepción que ha producido *el* error, de acuerdo con el código que hemos escrito.

Una de las cosas más convenientes de utilizar excepciones es que puede mostrar tanta información, o tan poca, como quiera. El objeto Exception tiene varios métodos para manejar sus propios errores.
El código siguiente muestra cómo utilizar throw e inmediatamente capturar la excepción. Podemos capturar también algunos parámetros útiles:

```
<?php
try {
   throw new Exception("Error de sintaxis");
} catch (Exception $ex){
   //Mensaje de error
   $mensaje = ($ex->getMessage()) ;
   //Código de error configurable
   $codigo = ($ex-igetCode());
   //Fichero del error
   $fichero = ($ex->getFile( ) ) ;
   //Linea donde se ha producido el error
   $linea = ($ex->getLine());

   echo "Error número $codigo: $mensaje en el fichero $fichero en
   la linea $linea";
}
?>
```

El mensaje de error que se muestra es muy típico en PHP. Como es totalmente configurable, puede generar su propio formato de Mensaje de Error, que puede ver en la figura 17.2.

Aunque en este ejemplo, el código que captura la excepción sólo tiene una línea, podrá incluir dentro del bloque varias líneas o funciones dentro que actuarán de la misma forma.

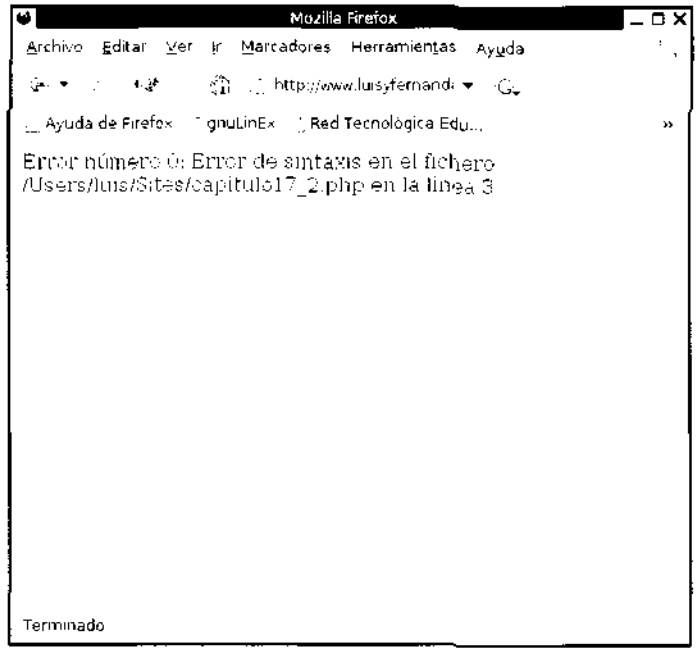



**Figura 17.2.** Información del mensaje de error.

## Heredar de (a clase Exception

PHP 5 permite definir sus propias clases que heredan de la clase `Exception`. Puede hacerlo de la siguiente forma:

```
<?php
class Excepción extends Exception
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
?>
```

El siguiente ejemplo muestra cómo crear clases propias que manejan errores a partir de la clase `Exception`:

```
<?php
$usuario_id = $_GET['usuario_id'];
try {

   if f J usuario_valido ($usuario__id) ) [
        echo "ERROR:: El usuario $usuario id no existe";
```

```
   } else {
        echo "CORRECTO:: El usuario $usuario_id existe";
   }
} catch(MiExcepcion $ex) {
   $cadena = "usr";
   if   ((strpos($usuario_id,$cadena) == false) |]
        (strpos($usuario_id,$cadena) != 0)) {
             $mensaje = $ex->getMessage();
   }
}
catch(OtraExcepcion $ex) {
   if (strlen($usuario_id) < 9) {
        $mensaje = $ex->getMessage();
   }
}
//Definición de las clases que heredan de Exception
class MiExcepcion extends Excepción
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
class OtraExcepcion extends Excepción
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
echo $mensaje;
function usuario_valido($usuario_id) {
   $cadena = "usr";
   if   ((strpos($usuario_id,$cadena) == false) ||
        (strpos($usuario_id,$cadena) != 0)) {
             throw new MiExcepcion("Usuario $usuario_id no
contiene el prefijo " ) ;
        }
   if (strlen($usuario_id) < 9) {
        throw new OtraExcepcion("Usuario tiene menos de 9
caracteres");
   }
}
?>
```

## Limitaciones de PHP 5

El objeto de excepciones es totalmente nuevo en PHP 5. Debido a esto, algunas funcionalidades creadas en otros lenguajes de programación, como

Java, no han sido todavía implementadas; por ejemplo el uso de los métodos finally() y throws(). Los errores comunes de PHP, que se muestran en el navegador, no son mapeados a excepciones automáticamente. Esta funcionalidad será implementada en versiones posteriores de PHP. Los errores que no pueden ser controlados por Exception pueden manejarse con otras técnicas de programación.

# Control de errores sin excepciones

Si ya conoce algunas versiones de PHP, sabrá que hay multitud de funciones capaces de controlar errores, incluidos los errores nativos de PHP, controladores personalizados de error y los errores de usuario.

## Errores nativos de PHP

PHP genera 3 tipos de error, dependiendo de su seriedad:

- **Notice:** No son errores severos y no crean problemas complejos. Por defecto, no se muestran en pantalla cuando suceden, a menos que digamos lo contrario en el archivo de configuración php.ini.
- **Warning:** Alguna parte del código ha causado un error, pero la ejecución continúa.
- **Fatal error:** Un problema serio que hace abortar la ejecución del programa.

Cada tipo de error es representado por una constante diferente: E_USER_NOTICE, E_USER_WARNING y E_USER_ERROR. Puede elegir el nivel de error que quiere mostrar en pantalla con la función error_reporting().

```
<?php
//Mostrar errores fatales
error_reporting(E_USER_ERROR);
//Mostrar advertencias y notificaciones
error_reporting(E_USER_WARNING | E_USER_NOTICE);
//Mostrar todos los errores
error_reporting(E_USER_ALL);
?>
```

El manejo de errores queda limitado la mayoría de veces a los del tipo warning. Los errores fatales se consideran tan críticos que no pueden controlarse y la ejecución del programa queda suspendida.

## Controladores de error

Cuando suceda un error que no tenga contemplado en su aplicación, pueden aparecer mensajes en la pantalla que no interesa mostrar. La solución es crear su propia función para controlar y mostrar errores.

```
<?php
function errores($error,{mensaje,$fichero,$linea) {
   echo "<b>: : ERROR : : </bxbr>" ;
   echo "Sentimos comunicarle que se ha producido un error";
   echo "Tipo de error: $error: $mensaje en $fichero en la linea
   $linea";
}
?>
```

Con estas líneas hemos creado una función que muestra el error producido, pero con un diseño acorde a nuestra página Web. Puede mostrar tanta información como quiera. El siguiente paso es llamar a la función `set_error_handler ()` para que PHP sepa que nuestra función `errores ()` se va a encargar a partir de ahora de gestionar los posibles fallos en el flujo del programa.

```
<?php
set_error_handler("errores");
?>
```

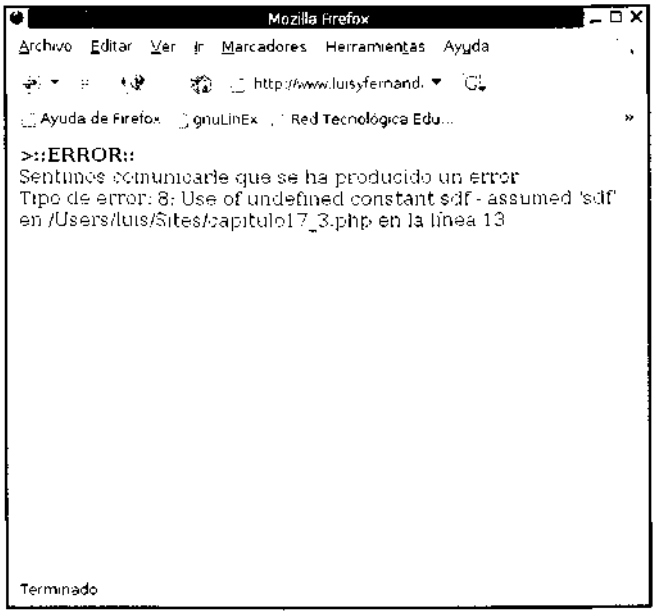



**Figura 17.3.** Mensaje de error propio.

Con el código anterior, todos los errores producidos del nivel que tenga permitido (fatal error, warning o notice) serán enviados a la función errores () para mostrar su mensaje particular.

## Errores de usuario con trigger_error()

Desde PHP 4, puede usarse una función para identificar los errores producidos por los usuarios, muy parecida a la función de PHP 5 throw () . La función trigger_error () genera un error del tipo que quiera (fatal error, warning o notice) y será gestionado por PHP o por su propio gestor de errores.

```
<?php
error_reporting(E_ALL);
function errores($error,$mensaje,$fichero,$linea) {
        echo "<b>: :ERROR: :</bxbr>";
        echo "Sentimos comunicarle que se ha producido un error";
        echo "Tipo de error: $error: $mensaje en $fichero en la
        línea $ linea";
}
set_error_handler("errores");

if (1 != 0 ) {
        trigger_error(" Esto es falso ",E_USER_NOTICE);
}
?>
```

Es mejor utilizar siempre la función trigger_error ( ) con set_error_handler () . El error que creemos puede ser de cualquiera de los 3 tipos existentes en PHP. En este caso, generamos un error notice para indicar que la condición es falsa.

## Depuración de errores

El manejo de errores puede ahorrar esfuerzos a la hora de depurar el funcionamiento óptimo de su código, pero también puede crear algunos dolores de cabeza si empieza a suprimir algunos fallos.

La funciónerror_log () crea un archivo donde se irán almacenando las ocurrencias de su código. Con el siguiente ejemplo podrá capturar todos los errores producidos en el programa, para después hacer un estudio de lo que ha pasado.

```
<?php
```

```
try {
   if ($usuario != "Luis") {
        throw new Exception("El usuario no existe");
   }
}
catch (Exception $ex) {
       $mensaje = ($ex->getMessage() ) ;
       $codigo = ($ex->getCode() ) ;
       $fichero = ($ex->getFile());
       $linea = ($ex->getLine());
       $log_mensaje =date("H:i d-m-Y") . " Error:: $codigo en
$fichero en la línea $linea. Error $mensaje";
   error_log($log_mensaje,3,"/Users/luis/Sites/error.log");
   echo $log_mensaje;
}
?>
```

Nota:

*Los ficheros* log *son muy utilizados en los Sistemas Operativos para mantener un registro de incidencias. Los ficheros* log *guardan desde el número de veces que ha entrado un usuario, hasta los errores que se han producido en el inicio del sistema.*

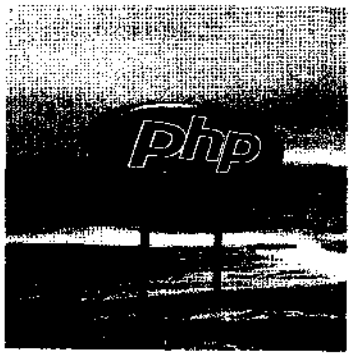
La función error_log() puede pasar como segundo parámetro un número entero que identifica el tipo de error y qué hacer con él.

- 0: Se utiliza el mecanismo del sistema operativo.
- 1: Envía el error a una dirección de correo especificada como cuarto parámetro. (Debe formar parte de una cabecera).
- 2: Envía el error a través de una sesión de depuración de PHP (Debe estar habilitado el modo depuración)
- 3: Envía el error a un fichero.

## Resumen

Como puede darse cuenta, la mayoría de los programas incluidos en el libro no tienen ningún tipo de control de errores. He preferido evitarlos para ganar en claridad y no tener que empezar explicando los bloques Try / Catch desde el principio. A partir de ahora siéntase libre para modificar todos los *scripts* a su antojo para que sean más fiables y seguros.

El desarrollo de las posteriores versiones de PHP permitirá manejar los errores más fácilmente. El futuro de las excepciones no ha hecho más que empezar con la versión 5 de PHP, y se esperan mejoras suculentas con respecto a las anteriores. Pese a esto, muchas librerías ya incluyen sus propios controles de error como la de MySQL o PEAR.




Capítulo 14

# Conexiones desde PHP 5

**En este capítulo aprenderá a:**

- Crear conexiones FTP con un servidor.
- Descargar y Enviar ficheros a un servidor mediante FTP.
- Elaborar mensajes de correo electrónico sencillos.
- Añadir adjuntos a un correo electrónico.
- Crear correos electrónicos complejos desde un formulario.

# Introducción

Transferir ficheros de un ordenador a otro es una tarea que sucede millones de veces a lo largo del día en Internet. Ya no es un problema compartir un fichero entre dos ordenadores ubicados entre distintos puntos del mundo. Los dos métodos más utilizados son el protocolo FTP y el correo electrónico.

Los ficheros pueden ser transferidos simplemente dejándolos en algún lugar público de nuestro servidor Web, o utilizando algún programa para enviarlos. El protocolo FTP y el correo electrónico son programas independientes que deben estar instalados en el servidor para poder usarse. Son dos programas del tipo cliente / servidor. Para poder utilizar cualquiera de los dos servicios anteriores debe conectarse a un servidor en un ordenador remoto y enviar los datos.

# FTP

Para utilizar el protocolo FTP tiene que tener habilitada la opción cuando instale PHP. Puede utilizar FTP para enviar ficheros en un script escrito en PHP desde alguna página Web.

Para conectar a un servidor tendrá que utilizar la siguiente función:

```
$conexión = f tp__connect ( "www. luisyf ernanda . net") ;
```

También puede utilizar una dirección IP para la conexión.

```
$conexión = ftp_connect ("127 . 0 . 0 . 1") ;
```

**Nota:**

*La dirección IP identifica un único ordenador dentro de Internet.*

*La IP 127.0.0.1 apunta a nuestro propio ordenador.*

El servidor FTP contiene un listado de usuarios a los que les permite la conexión. Dependiendo del usuario que se conecte, el servidor le permitirá descargar unos archivos u otros.

El siguiente paso es autentificarse como usuario del servidor de FTP con la función:

```
ftp_login($conexión, $usuario, $clave);
```

Como parámetros tiene que pasar la conexión establecida anteriormente, el usuario y la contraseña.

> Advertencia:
>
> *El protocolo FTP no es seguro. El usuario y la contraseña se envían en formato de texto y podrían ser capturados por algún programa* sniffer *que se encuentre entre su ordenador y el servidor de destino.*

Como de costumbre, puede encapsular todas las funciones para hacer una clase que permita conectar a un servidor.

```
<?php
class ftp
{
  prívate $servídor;
  prívate $usuario;
  prívate $clave;
  prívate $conexion;
  prívate $login;
  function __construct($servidor,$usuario,  $clave)
  {
          set_time_limit(0);
          $this->servidor = $servidor;
          $this->usuario = $usuario;
          $this->clave = $clave;
          $thís->conectar($this->servidor);
  }
  prívate function conectar($servidor)
  {
            $this->conexion = ftp_connect($servidor,21, 30) ;
  }
  prívate function loginFtp($usuario,$clave)
  {
            $this->login = ftp_login($this->conexion, {usuario,
            $clave);
  }
}
$ftp = new ftp("127.0.0.1","luis","secreto") ;
?>
```

Como puede ver, el método constructor utiliza la función set__time_limit () . Normalmente los *script* de PHP tienen un tiempo de vida determinado en el fichero de configuración php . ini. Este tiempo suele ser de 30 segundos y es una protección para que, si un programa se queda

bloqueado y su ejecución lleva más de ese tiempo, automáticamente quede cancelada su ejecución.

La función set_time_limit () le da la posibilidad de ampliar el tiempo de ejecución al número de segundos que indique como parámetro. Si el argumento es un 0, el tiempo de ejecución será infinito.

La función ftp_connect () también puede llevar como parámetros el puerto de conexión (por defecto es el 21) y el número de segundos que tardará PHP en volver a intentar la conexión si el intento anterior ha sido fallido.

## Mostrar los archivos remotos

Después de conectar con el servidor de FTP, lo más usual es recuperar un listado de ficheros que están en el servidor. La función ftp_nlist () extrae el nombre de los archivos del directorio que pasemos como parámetro.

El resultado es un *array*.

```
public function ver($directorío = "./")
{
   $this->archivos = ftp_nlist($this->conexion,$directorio);
   foreach ($this->archívos as $archivo) {
        echo "$archivo<br>";
   }
}
```

Puede añadir el método anterior a la clase ftp. La función ftp_nlíst () utiliza la conexión almacenada en el constructor de la clase y recibe un listado de ficheros que guarda como *array* en la variable `$this->archivos`.

Más adelante, utilizando un bucle foreach podrá extraer todos los nombres e imprimirlos en pantalla, como puede comprobar en la figura 18.1.

Si no es la primera vez que utiliza el protocolo FTP, sabrá que al conectar con un servidor lo más común es que conecte en el sistema raíz, donde encontrará los archivos y directorios que puede utilizar. La función ftp_chdir () permite cambiar de directorio dentro del sistema de ficheros.

Si en algún momento no sabe en qué directorio se encuentra, puede hacer uso de la función ftp_pwd (), que le dará la ruta completa de su situación.

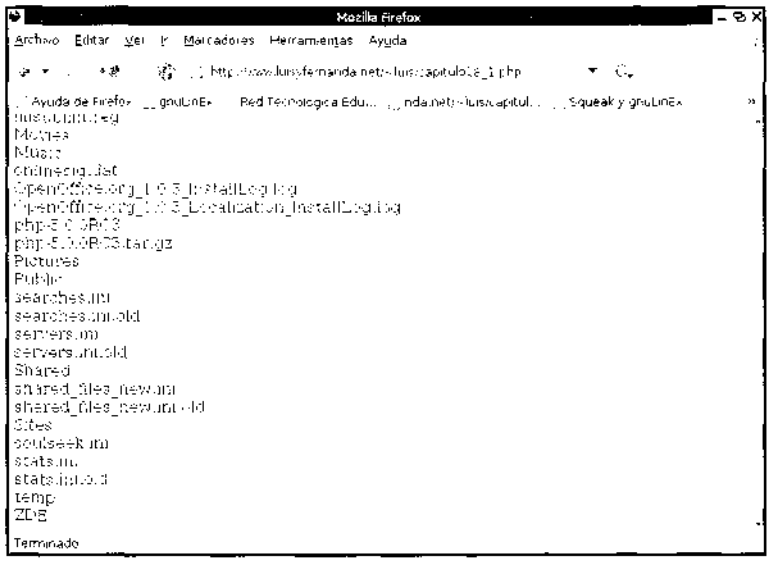



**Figura 18.1.** Conexión FTP mediante PHP.

## Descargar y Enviar ficheros

Puede descargar ficheros con la función `ftp_get()`. El siguiente ejemplo descarga un fichero después de hacer la conexión:

```
ftp_get($conexión, "nuevoFichero.txt","Fichero.txt",FTP_ASCII);
```

El nombre `nuevoFichero.txt` se corresponde con el nombre del fichero que tendrá el archivo después de ser descargado y `Fichero.txt` es el archivo remoto que queremos descargar.

El último parámetro, FTP_ASCII, le dice al FTP el tipo de archivo que va a ser descargado. Los modos que puede elegir son FTP_ASCII o FTP_BINARY. Los ficheros binarios son archivos escritos en el lenguaje máquina como los archivos gráficos.

Además, puede enviar archivos al servidor con la función `ftp_put()`. El siguiente ejemplo muestra cómo subir un archivo al servidor:

```
ftp_put($conexion, "nuevoFichero.txt","Fichero.txt",FTP_ASCII);
```

Como en el ejemplo anterior, `nuevoFichero.txt` será el nombre del archivo en el servidor remoto y `Fichero.txt` el archivo que vamos a enviar. El ejemplo siguiente descarga todos los archivos con extensión PHP en nuestro ordenador local.

```
public function descargaPHP($directorio = "./")
```

```
{
   $this->archivos = ftp_nlist($this->conexion,$directorio);
   foreach ($this->archivos as $archivo) {
        if (ereg(".php",$archivo)) {
             if ( ! f ile__exists ($archivo) ) {
                  ftp_get($this->conexion,$archivo,$directorio
                  $archivo,FTP_ASCII);
             } else {
                  echo "Fichero $archivo ya existe!!<br>";
             }
        } else {
             echo "El archivo $archivo no es de PHP<br>";
        }
   }
}
```

## Otras funciones de FTP

Además de las funciones descritas en este capítulo, existen otras muchas para el manejo del protocolo FTP.

**Tabla 18.1.** Funciones comunes al manejo de FTP.

| Funciones | Descripción |
|---|---|
| ftp_cdup($conexion) | Cambia al directorio inmediatamente superior al directorio actual. |
| ftp_chdir($conexión, "nombre directorio") | Cambia al directorio que pasemos como parámetro. |
| ftp_close($conexion) | Cierra una conexión FTP. |
| ftp:connect("servidor") | Establece una conexión con el servidor. |
| ftp_delete($conexión, "ruta") | Borra un fichero del servidor. |
| ftp_exec($conexión, "comando") | Ejecuta un comando del sistema remoto. |
| ftp_fget($conexión, $descriptor, "Fichero.txt", FTP_ASCII) | Descarga un fichero y vuelca los datos en un fichero abierto en el ordenador local. |
| ftp_fput($conexión, $descriptor, "Fichero.txt", FTP_ASCII) | Envía un archivo a un fichero abierto. |
| ftp_get($conexión, "nuevoFichero.txt", "Fichero.txt", FTP_ASCII) | Recupera un fichero de un servidor. |
| ftp_put($conexión, "nuevoFichero.txt", "Fichero.txt", FTP_ASCII) | Envía un fichero al servidor. |

| Funciones | Descripción |
|---|---|
| ftp_login($conex¡ón, $usuario, $clave) | Autentifica al usuario en el servidor. |
| ftp_mdtm($conexión,"Fichero.txt") | Recupera la fecha de la última actualización del fichero. |
| ftp_mkdir($conexión, "directorio") | Crea un directorio. |
| ftp_nlist($conexión, "directorio") | Recupera la lista de archivos de un directorio. |
| ftp_pwd($conexion) | Obtiene el nombre del directorio actual. |
| ftp_rename($conexión, "ArchivoAntiguo","ArchivoNuevo") | Cambia de nombre un archivo remoto. |
| ftp_size($conexion,"Fichero.txt") | Devuelve el tamaño de un fichero. |
| ftp_systype($conexion) | Permite obtener el tipo de sistema. |

# Correo electrónico

La aplicación más utilizada, sin duda, en Internet es el correo electrónico. La explosión de Internet y la expansión de servidores gratuitos que ceden cuentas de correo a todos los usuarios ha permitido que el crecimiento de los *e-tnail* se dispare en estos últimos años.

Para lograr la comunicación a través del correo electrónico han de intervenir numerosas piezas de un puzzle.

La mayor parte del modelo utiliza:

- **Servidor TCP / IP:** Se encarga de mantener activos para la conexión diversos servicios como FTP, SMTP o POP.
- **MTA *(Mail Transfer Agent):*** Acepta correo de otros servidores SMTP y busca una ruta para que los correos lleguen a su destino.
- **Cola de correo:** Recipiente donde se almacenan todos los correos entrantes dirigidos a una persona.
- **MUA *(Mail User Agent):*** Programa muy liviano para leer el correo electrónico, normalmente desde el servidor.
- **Servidor POP / IMAP:** Recibe peticiones de lecturas de correo. El servidor chequea si hay correos nuevos y los envía al cliente.
- **MLM (Mailing List Manager):** Este *software* ayuda a enviar grandes cantidades de correo.

## Enviar correo desde PHP

Sólo existe una función para enviar correo electrónico desde PHP 5. La función mail () devolverá un valor *true* si los datos son enviados correctamente. La forma más sencilla de utilizar esta función la puede ver en el ejemplo siguiente:

```
<?php
$resultado = mail(" luis'ánccextremadura.org", "Ejemplo de
Asunto", "Cuerpo del mensaje");
if ($resultado) {
   echo "Correo enviado correctamente";
} else {
   echo "El correo no ha podido enviarse" ;
}
?>
```

El primer parámetro recibe el correo de la persona a la que quiere enviar el mensaje. El segundo es un resumen del contenido de su carta electrónica y, por último, el tercer parámetro es el mensaje que quiere transmitir.

Puede dividir la función en variables para que sea más sencillo de utilizar:

```
<?php
$correo = "luis@nccextremadura.org";
$asunto = "Ejemplo de Asunto";
$cuerpo = "Cuerpo del mensaje";
$resultado = mail($correo, $asunto, $cuerpo);
if ($resultado) {
   echo "Correo enviado correctamente";
} else {
   echo "El correo no ha podido enviarse";
}
?>
```

Hay veces que necesitará algo más de control sobre lo que quiera enviar, o necesite enviar un correo a varias personas a la vez. Esto es posible añadiendo cabeceras adicionales a la función mail:

```
<?php
$correo = "luis@nccextremadura.org";
$asunto = "Ejemplo de Asunto";
$cuerpo = "Cuerpo del mensaje";
$cabecera = "From:
fernanda@\r\nbcc:pedro@nccextremadura.org\r\nContent-type:
text/plain\r\nX-mailer: PHP/" . phpversion();
$resultado = mail($correo, $asunto, $cuerpo, $cabecera);
if ($resultado) {
```

```
    echo "Correo enviado correctamente";
} else {
    echo "El correo no ha podido enviarse";
}
```

La cabecera muestra cómo enviar, junto al correo, la dirección de la persona que lo envía y una copia del mismo mensaje al correo pedro@nccextremadura.org. Es necesario añadir los separadores \r y \n para indicar un salto de línea entre los parámetros de la cabecera.

Las cabeceras adicionales pueden añadir muchos tipos de información, entre los que se encuentran:

- El nombre del emisor.
- Correo del emisor.
- Correo de respuesta.
- X-mailer y número de versión.
- versión MIME.
- Content-type.
- Codificación.
- Campo de copia y copia oculta.

La función mail () está preparada para emitir mensajes sencillos, pero hay muchas características que debe implementar con cabeceras. Por eso, han nacido una serie de librerías, escritas en PHP, que facilitan el envío de correos.

La librería PHPMailer añade a la función mail () muchas funcionalidades, como ficheros adjuntos o correos HTML.

## PHPMailer

PHPMailer está basado en programación orientada a objetos. Contiene varios ficheros que deben guardarse en el directorio de PHP dedicado a librerías o en nuestro espacio de trabajo.

La construcción más básica es la siguiente:

```
<?php
require_once("phpmailer/class.phpmailer.php");
$correo = new phpmailer() ;
//Datos personales del emisor
$mail->From = "luis@nccextremadura.org";
```

```
$mail->FromName = "Luis Miguel Cabezas";

$mail->Subject = "Mensaje de prueba";

$mail->Body = "Cuerpo del mensaje";
//Dirección de destino
$mail->AddAddress("zeevSphp.zend.com","Zeev Suraski");

if ( l$mail->Send()) {
   echo "Correo enviado correctamente";
} else {
   echo "El correo no ha podido enviarse";
}
?>
```

Como puede ver, las propiedades de la clase phpmailer son auto explicativas. El ejemplo va asociando cada propiedad a un dato necesario para el envío. El método AddAddress () añade una dirección de correo a la que enviar el mensaje y Send () lo envía a través de Internet.

## Añadir un fichero adjunto

Existen tres métodos para añadir un fichero adjunto a un mensaje: dos para archivos y otro más específico para imágenes:

- **AddAttachment(ruta, nombre):** Añade un fichero localizado en el servidor. Este archivo se puede subir antes al servidor mediante un formulario.
- **AddStringAttachment(caracteres,** fichero): Se puede utilizar para enviar ficheros almacenados en variables.
- **AddEmbeddedImage(ruta, cid, nombre):** Añade una imagen y permite añadirla al cuerpo de un correo HTML.

Las líneas siguientes se pueden añadir al ejemplo anterior para enviar un fotografía almacenada en el servidor.

```
if  (!$mail->AddAttachment("abuelo.jpg","abuelo.jpg"))  {
   echo "Falló al añadir un fichero adjunto";
}
```

Para demostrar la teoría anterior vamos a crear un pequeño formulario que permita enviar correo. Es un código muy sencillo, pero permite enviar mensajes escritos en HTML e incluir imágenes.

```
<?php
if (isset($_POST["nombre"])) {
          require_once("phpmailer/class.phpmailer.php");
```

```
            $correo = new phpmailer() ;
            //Datos personales del emisor
            $mail->From =  "luis@nccextremadura.org";
            $mail->Fxow£!aw>e = ""Luis Miguel Cabezas" ;
            $mail->Subject = $_POST [ "asunte?" ] ;
            $mail->Body =  $_POST["mensaje"];
            //Dirección de destino
            $mail->AddAddress($_POST["nombre"],$_POST["correo"]);
            if    (!$mail->AddEmbeddedImage('$_FILES["imagen"]
            ["tmp_name"' , '12345', 'imagen')) {
                    echo "Falló al añadir un fichero adjunto";
                    }
            if ( !$mail->Send() ) {
                    echo "Correo enviado correctamente";
            } else {
                    echo "El correo no ha podido enviarse";
            }
}
?>
<html>
<body>
<form  action="mail.php"  method="post"  enctype="multipart/form-
data"  name="forml">
  <table width="50%"  border="0" cellspacing="0"
cellpadding="O">
    <tr>
      <td  width="28%">Nombre:</td>
      <td width="72%"xinput name="nombre"  type="text"
id="nombre"></td>
    </tr>
    <tr>
      <td>Correo:</td>
      <tdxinput  name="correo"  type="text"  id="correo"></td>
    </tr>
    <tr>
      <td>Asunto:</td>
      <tdxinput  name="asunto"  type="text"  id= "asunto"></td>
    </tr>
    <tr>
      <td>Mensaje:</td>
      <tdxtextarea name= "mensa j e " rowS="5" id= "mensa j e " ></
      textareax/td>
    </tr>
    <tr>
      <td>Selecciona  imagen:  </td>
      <tdxinput  name="imagen"  type="file"  id=" imagen" x/td>
    </tr>
  </table>
  <p>
```

```
    <input type="submit" name="Submit" value="Enviar">
  </P>
</form>
</body>
</html>
```

La figura 18.2 muestra un formulario capaz de enviar contenido HTML e imágenes adjuntas.

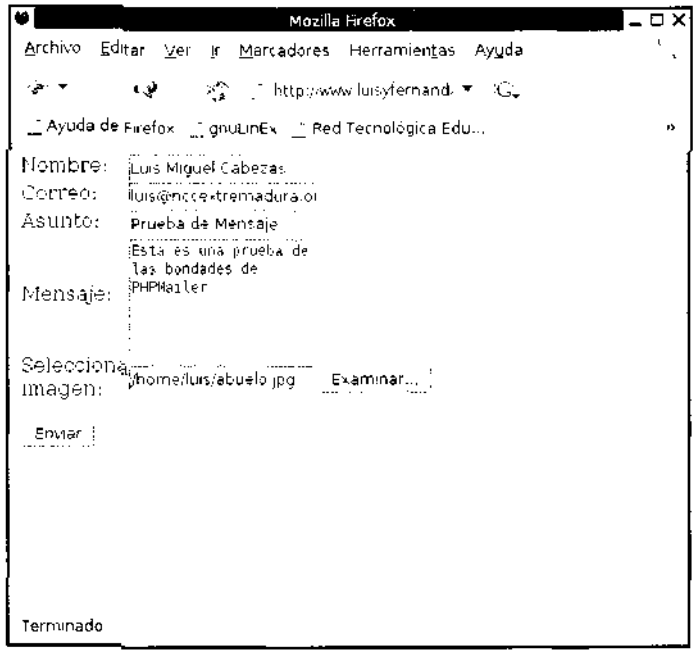



**Figura 18.2.** Formulario de envío de correos electrónicos.

# Resumen

PHP se ha promovido tanto debido a las numerosas extensiones que permiten hacer casi cualquier cosa. En concreto, las dos formas vistas para conectarse con un servidor le serán muy útiles para ofrecer servicios innovadores a sus usiiarios.

La mayoría de los portales Web que necesitan el registro de sus usuarios envían por correo electrónico las contraseñas y los datos de inscripción. La función `mail ()` le puede servir para hacer sus pequeños controles de usuarios.

Si se siente más ambicioso, también puede utilizar PHPMailer para construir una página Web que permita enviar o recibir correo electrónico. La recomendación es que continúe investigando en la clase PHPMailer y en todo lo que ofrece que no hemos podido ver en este capítulo.


Capítulo 11

# Creación de archivos PDF

**En este capítulo aprenderá a:**

- Manejar la librería FPDF.
- Crear documentos básicos.
- Crear una cabecera y un pie de página común a todo el documento.
- Crear tablas con cabeceras, distintos colores y tipos de letra.
- Añadir enlaces externos e internos.

## Introducción

Como probablemente ya sabe, el HTML no es el único formato para mostrar contenido en Internet. Además de los distintos tipos de imágenes, tenemos varios formatos capaces de contener textos como Adobe PDF, Macromedia Flash o algunos documentos alternativos escritos en XML como los archivos gráficos SVG (Gráficos Vectoriales Escalables), WML (Lenguaje de Marcas Inalámbrico) o XUL (Lenguaje de interfaz de Usuario en XML con Mozilla). Cada uno de estos tipos de documentos tiene su propio terreno y de nosotros depende elegir cuál o cuáles utilizar para dotar a nuestra Web de mayores posibilidades.

Los archivos Adobe PDF se han establecido como un estándar en los documentos de texto para la Web.

Existen dos extensiones escritas para PHP, PDFLib y ClibPDF, capaces de generar archivos PDF. Desgraciadamente estas librerías son de uso comercial y requieren el pago de algún tipo de licencia.
Además de las extensiones anteriores, tenemos una serie de aplicaciones escritas completamente en PHP que permiten llegar al mismo resultado, como R&OS PDF o FPDF. No son tan rápidas como las extensiones, pero cumplen perfectamente nuestras expectativas.

## Librería FPDF

La librería FPDF, creada en septiembre de 2001, no necesita licencia de uso, ni tampoco alguna configuración especial de PHP. Lo único que necesita es la librería, que la puede descargar del sitio Web http://www.fpdf.org.

La librería principal contiene la definición de una clase, que debe instanciar para poder generar sus documentos. Nuestro código inicial debería ser algo parecido a:

```
<?php
require_once(fpdf/fpdf.php);
$pdf = new FPDF();
?>
```

El constructor se utiliza de esta forma con los valores por defecto que son página A4 con orientación vertical y el tamaño medido en milímetros. El primer parámetro del constructor permite elegir entre orientación verti-

cal (L) u horizontal (P). El segundo parámetro sirve para que las medidas de referencia se hagan en base a puntos (pt), milímetros (mm), centímetro (cm) o pulgadas (in). El último parámetro nos ofrece la oportunidad de elegir entre los diferentes formatos de página *que* existen: A3, A4, A5, *Letter* y *Legal.*

**Nota:**

*Hay que recordar que un punto es 1/72 pulgadas.*

```
$pdf = new FPDF('P','cm','A4');
```

## Nuestro primer documento

El siguiente ejemplo muestra la creación básica de un documento PDF de forma lineal.

```
<?php
define('FPDF_FONTPATH','fpdf/font/');
require_once('fpdf/fpdf.php');
$pdf = new FPDF();
$pd£->A.ddPageU•,
$pdf->SetFont('Arial','',15);
$pdf->Write(155,'Mi primer documento!!!');
$pdf->Output();
?>
```

La primera línea define una constante que contiene la ruta donde están almacenadas las fuentes. Después de las funciones que incluyen la clase y la instancia del objeto tenemos una serie de llamadas a métodos que nos permiten manipular el documento a nuestro antojo.

**Advertencia:**

*FPDF mantiene la posición de un cursor. Las funciones que generan texto imprimirán los datos de acuerdo con la posición del cursor en ese momento. Es posible, de todas formas, cambiar de sitio el cursor con algunas funciones que veremos en este capítulo.*

El método AddPage () añade una página al documento con la orientación que Le pida. En este caso hemos añadido una con orientación vertical, que es la elegida por defecto. SetFont () selecciona una fuente tomando

como parámetro el tipo, la propiedad (negrita, cursiva) y el tamaño. Por último tenemos una llamada a los métodos Write () y Output (), que escriben el texto en una posición determinada del documento y lo muestran en pantalla respectivamente, como puede ver en la figura 19.1.

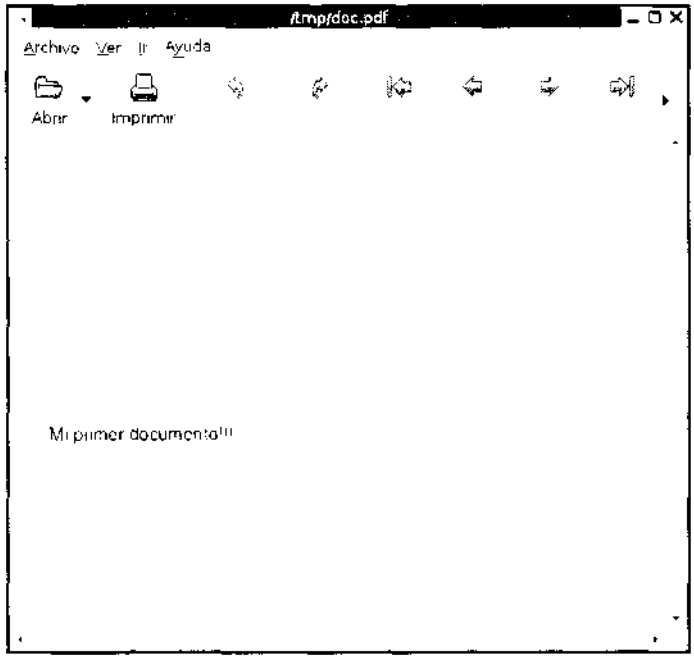



**Figura 19.1.** Primer documento en PDF.

## Funciones de texto

### Método Write()

Imprime el texto pasado como parámetro en la posición que indica el cursor. La construcción es:

```
Write(altura,texto,enlace)
```

Los parámetros son los siguientes:

- **altura:** Altura de la celda.
- **texto:** Texto a imprimir.
- **enlace:** Podemos considerar el texto como un enlace a una página o a una Web.

Como ejemplo puede ver el primer código que hemos realizado.

### Método Cell()

El método `cell()` puede tomar hasta 8 parámetros y es el encargado de imprimir en el documento PDF un cuadro de texto en la posición indicada. El cuadro de texto puede tener un borde alrededor o en alguno de los lados.

La construcción es como sigue:

```
cell(ancho, altura, texto, borde, línea, alineación, relleno, enlace)
```

Los dos primeros parámetros son obligatorios, los demás opcionales. A continuación vemos para qué sirve cada parámetro:

- **ancho:** Ancho de la celda. Si es 0 se considera el ancho total del folio.
- **altura:** Altura de la celda.
- **texto:** Texto a imprimir.
- **borde:** Un 1 significa que llevará borde. También se pueden incluir algunos caracteres para indicar en cuál de las partes de la celda debe llevar borde: L, R, T y B significan Izquierda, Derecha, Arriba y Abajo respectivamente.
- **línea:** Cuando termine la llamada a la creación de la celda, indica al generador de PDF dónde debe colocar el cursor para continuar. Un 0 indica a la derecha de la tabla, un 1 en la siguiente línea y un 2 seguido.
- **alineación:** Acepta 3 caracteres: L para alinear a la izquierda, R a la derecha y C para texto centrado.
- **relleno:** Indica con un 1 si la celda debe tener un relleno para diferenciarla del resto del folio.
- **enlace:** Enlace devuelto por la función `Addlink()`.

Como ejemplo puede ver la forma de introducir un título en una página:

```
<?php
define('FPDF_FONTPATH','fpdf/font/');
require_once('fpdf/fpdf.php');
$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Courier','',10);
$pdf->Cell(80);
$pdf->Cell(20,10,'Titulo',1,1,'C');
$pdf->Output();
?>
```

Una celda sólo puede contener el número de caracteres que quepa en una línea. Si necesita mostrar más líneas de texto, puede utilizar el método

MuíticE11 () que toma los mismo parámetros que CE11 (), pero cuando el texto llegue al final de la línea, se añade automáticamente un salto a la línea siguiente.

**Truco:**

*Puede utilizar* CE11 () *para desplazar el cursor hacia la derecha añadiendo como parámetro un ancho de 8 centímetros (80 milímetros). Después, para imprimir el texto, añada otra celda que incluya el literal "Título" centrado dentro de una celda con borde.*

## Desplazamiento de los cursores

Existen varias funciones para colocar el cursor en distintas partes del documento. SetX () y SetY () colocan el texto en el punto indicado, en la medida seleccionada (centímetros, milímetros, etc).

SetX () desplaza el cursor de izquierda a derecha tantas unidades como indique el parámetro y SetY () de arriba hacia abajo. Si se les pasa un número negativo, el origen se tomará desde la derecha o desde abajo respectivamente.

SetXY () toma dos valores y permite realizar las dos operaciones con una sola instrucción.

```
$pdf->SetXY(100,100);
$pdf->write(2,"Este texto se mostrará a 10 cm del margen");
```

## Salto de página automático

El método SetAutoPageBreak () activa o desactiva el salto de página automático.

Esto quiere decir que si está activado, cuando el texto llegue al final de la página, se creará una página nueva para poder contener el resto. El primer parámetro debe ser *true ofalse* para activa o desactivar la propiedad.

El segundo parámetro indica la distancia desde la parte inferior que desencadena la llamada a una nueva página.

```
SetAutoPageBreak(true,20);
```

El código anterior activa el salto de página automático cuando el texto llegue a 2 centímetros del margen inferior.

## Sobrescribir los métodos

La clase FPDF contiene algunos métodos vacíos que debe escribir usted si quiere que sus documentos queden profesionales. Puede crear un objeto propio que herede del objeto FPDF e implementar algunas características avanzadas.

## Cabecera

La cabecera de un documento es un conjunto de texto escrito en la parte superior del folio y que debe aparecer en todas las páginas. La clase FPDF implementa el método header () que se llama cada vez que existe un salto de página.

Puesto que el método está vacío en FPDF, tendrá que crear un objeto nuevo que herede todos sus métodos e implemente la cabecera.

```
<?php
define('FPDF_FONTPATH','fpdf/font/');
requireonce('fpdf/fpdf.php');
class PDF extends FPDF
{
 function Header()
 {
  $this->SetFont('Arial','B',15);
  $this->SetX(70);
  $this->Cell(100, 10, 'Manual Imprescindible de PHP 5',1,0,'C');
  $this->Ln(20);
 }
}
$pdf = new PDF();
for($i=l;$i<=200;$i++)
 $pdf->Cell(0,10,'Imprimiendo línea número '.$i,0,l);
$pdf->Output();
?>
```

El método header () configura un tipo de letra determinado, en negrita, avanza el cursor 8 centímetros hacia la derecha e imprime una celda con el título.

El método Ln (2 0 ) inserta un salto de línea cuya altura es determinada por el argumento.

Para probar el objeto hemos hecho un bucle que imprime 100 líneas y así comprobar que la cabecera aparece invariable. El resultado se puede observar en la figura 19.2.

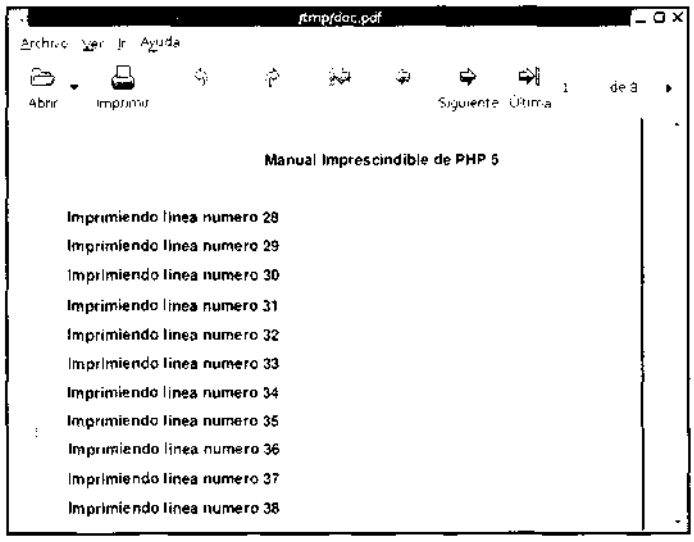



**Figura 19.2.** Cabecera de un documento PDF.

## Imagen de cabecera

También es posible añadir una imagen a la cabecera con el método Image (). La construcción es la siguiente:

```
Image(fichero, x, y, ancho, alto, tipo, enlace)
```

Los parámetros necesarios son: fichero, x e y. La descripción de todos los argumentos es:

- **fichero:** Indica la ruta hacia el fichero que va a insertar.
- x: Posición en el eje x de la imagen.
- y: Posición en el eje y de la imagen.
- **ancho:** Ancho de la imagen.
- **alto:** Alto de la imagen. Si no se indica y el parámetro ancho está definido, el alto se escala en función del ancho.
- **tipo:** El tipo de imagen. Por ahora sólo se aceptan imágenes JPG, JPEG y PNG.
- **enlace:** La imagen puede ser un enlace.

Ahora que sabe cómo insertar imágenes, puede incluir en la cabecera el logotipo de su empresa o asociación.

```
function Header()
{
```

```
    $this->SetFont{'Arial','B',15);
    $this->SetX(70);
    $this->Cell(100,10,'Manual Imprescindible de PHP 5',1,0,'C);
    $this->SetX(-20);
    $this->Image("php.png",170,5);
    $this->Ln(20);
}
```

La figura 19.3 muestra una cabecera con una imagen insertada.

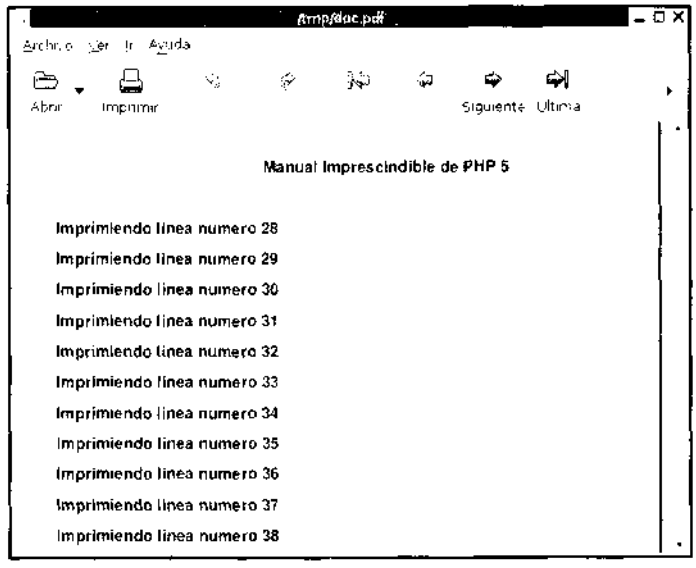



**Figura 19.3.** Cabecera de un documento PDF con imagen.

## Pie de página

De la misma forma, puede sobrescribir el método que imprime el pie de página de la clase FPDF.

Este método recibe el nombre de `Footer()` y tiene el mismo tratamiento que la cabecera; cada vez que existe un salto de página se hace una llamada al mismo.

Nuestro pie de página puede quedar de la siguiente forma:

```
function Footer()
{
    $this->SetY(-15);
    $this->SetFont('Arial','I',10);
    $this->Cell(0,10,'Página '.$this->PageNo(),0,0,'C');
}
```

SetY () nos coloca el cursor a 15 milímetros del borde derecho. Seguidamente escoja el tipo de fuente *Aria!* e imprima el número de la página actual con el método PageNo () .

Muchos documentos informan además del número de páginas que tiene el texto. Esto es complicado a primera vista, porque, a priori, no podrá saber cuántas páginas ocupará su texto. Existe un método que se ejecuta cuando el documento es cerrado y sustituye una marca por el número de folios con los que cuenta nuestro proyecto.

```
function FooterO
{
  $this->SetY(-15);
  $this->SetFont('Anal','I',10);
  $this->AliasNbPages();
  $this->Cell(0,10,'Página ' . $this->PageNo() . '/
{nb}',0,0,'C');
}
```

El método AliasNbPages () cuenta el número de páginas cuando el documento está cerrado y sustituye la marca {nb} por este número.

## Tablas

Una tabla no es más que un conjunto de celdas colocadas en un orden determinado. Podemos diferenciar entre la cabecera de la tabla, que contendrá las propiedades, y los datos. Si realiza una clase para escribir tablas puede hacer también esa diferenciación, cambiando el tipo de letra, el tamaño o el color. El código siguiente muestra cómo se puede instaurar una clase para la creación de tablas fácilmente:

```
<?php
define('FPDF_FONTPATH','fpdf/font/');
require_once('fpdf/fpdf.php');
class PDF extends FPDF
{
function tabla($cabecera,$datos)
  {
       $this->cabecera($cabecera);
       $this->datos($datos);
  }
  function cabecera($cabecera)
  {
  //Cabecera
       $this->SetFont('Arial','B',15);
```

```
            foreach($cabecera as $columna) {
                $this->Cell(40,7,$columna,1,0,'C');
            }
            $this->Ln();
    }
    function datos($datos)
    {
    //Datos
            $this->SetFont('Arial','',12);
            foreach ($datos as $dato) {
                foreach ($dato as $columna) {
                    $this->Cell(40,7,$columna,1,0,' C );
                }
            $this->Ln();
            }
    }
}
$pdf = new PDF();
$pdf->AddPage();

$cabecera = array("Usuarios","Visitas","Localidad","Teléfono");
$datos =    array(array("Luis Miguel ","12","Badajoz","555345678"),
            array("María Fernanda","45","Mérida","234234654"),
            array("Pedro Casas","3","Cáceres","342123890"));
$pdf->tabla($cabecera,$datos);
$pdf->Output();
?>
```

En este caso, los datos se sacan de un *array* de dos dimensiones, pero puede utilizar la librería de bases de datos o la de XML, creadas en los capítulos anteriores, para extraerlos.

Los datos los recibe el método `Tabla()`, que los envía a sus correspondientes métodos para ser tratados. El funcionamiento básico consiste en la creación repetitiva de celdas cuyo número coincidirá con el número de datos existentes en *el array.* Todas las celdas se muestran seguidas, dando sensación de ser una tabla, como puede ver en la figura 19.4.

Ahora que es casi un experto en la creación de tablas para facturas o albaranes en PDF, puede dar un toque de color al relleno, texto o líneas de las celdas.

Puede comprobar que, con tan sólo tres líneas de código más, el aspecto visual del documento mejora considerablemente.

```
function cabecera($cabecera)
    {
        //Cabecera
        $this->SetFillColor(255,0,0);
```

```
        $this->SetTextColor(255);
        $this->SetDrawColor(12 8,0,0);
        $this->SetFont('Arial','B',15);
        foreach($cabecera as $columna) {
            $this->Cell(40,7,$columna,1,0,'C',1);
        }
        $this->Ln();
}
function    datos($datos)
{
        //Datos
        $this->SetTextColor(1);
        $this->SetDrawColor(128,0,0);
        $this->SetFont('Arial','',12);
        foreach ($datos as $dato) {
            foreach ($dato as $columna) {
                $this->Cell(40,7,$columna,1,0,'C');
            }
        $this->Ln();
        }
}
```

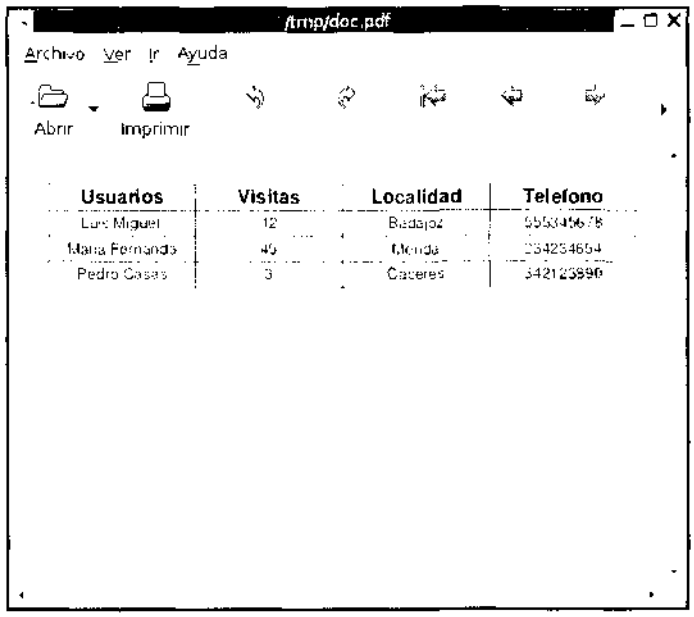



**Figura 19.4.** Tabla en un archivo PDF.

El resultado es mucho más llamativo gracias a los métodos que añaden color. El método `SetFillColor()` define el color de fondo de las celdas

y los rectángulos que no sean transparentes. Puede ser expresado con un componente RGB o en escala de grises. El valor se mantiene durante todo el documento hasta que se vuelva a invocar el método.

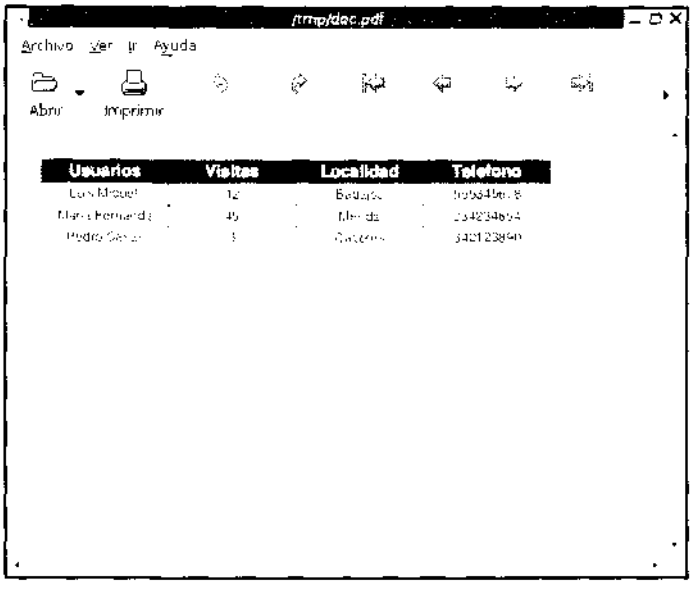



**Figura 19.5.** Tabla en un archivo PDF de color.

SetTextColor() y SetDrawColor() funcionan de la misma forma, actuando esta vez sobre el color del texto o el color de las líneas y bordes de celda. La figura 19.4 muestra los colores (tonos de grises en este caso) distintos que pueden tomar un archivo PDF.

# Enlaces

Una de las ventajas del formato PDF es que permite hacer enlaces internos, es decir, que si tiene un documento de 500 folios podrá acceder a cada uno de los capítulos haciendo clic en el índice, tal y como funcionan los enlaces en las páginas Web.

Un enlace externo es muy fácil de crear:

```
$pdf->Write(5,www.luisyfernanda.net,'http://www.luisyfernanda.net');
```

Los métodos dedicados a texto tienen un argumento para añadir un enlace que, si es externo, debe llevar la dirección URL completa.

Crear un enlace interno es algo más complicado. Tiene que hacer uso del método Addlink (), que genera una referencia que debe ponerse en el lugar hacia donde quiere que se produzca el salto. En ese lugar utilice el método SetLink () para cerrar el enlace.

El código de ejemplo puede aclarar el concepto:

```
<?php
define('FPDF_FONTPATH','fpdf/font/');
require_once('fpdf/fpdf.php');
$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial','',15);
$enLace = $pdf->AddLink();
$pdf->Write(5,"Este texto contiene un enlace",$enlace);
$pdf->AddPage{};
$pdf->SetLink($enlace);
$pdf->Write(5, "Enlace Interno");
$pdf->Output();
?>
```

Lo único a destacar es la creación del enlace con AddLink (). La variable $enlace recoge el valor de referencia y se añade al texto de la línea siguiente. En este momento el texto contiene un enlace, pero no sabemos hacia dónde. La solución está en llamar al método SetLink () en la página de destino con la referencia de parámetro.

## Resumen

Los archivos PDF se consideran desde hace unos años ideales para la generación de contenidos on-line. Así, revistas digitales como www.phparch.com o www.phpmag.net utilizan este sistema para vender su publicación.

Además de revistas digitales, podemos encontrar empresas que hacen llegar las instrucciones de sus productos en PDF o realizan las facturas de los clientes *on-line,* como el programa de facturación extremeño Galopín (http://galopin.sinuh.org).

La generación de sus propios archivos ya no es un misterio y puede implementar clases muchos más potentes que las mostradas en el libro para sus propios proyectos. La inclusión de colores, imágenes y tablas de distintos tipos harán del lector un profesional de la edición Web.


# Capítulo 11

# Plantillas con Smarty

**En este capítulo aprenderá a:**

- Conocer qué es un sistema de plantillas con Smarty.
- Independizar el código HTML de PHP.
- Crear plantillas con variables y bucles.
- Crear plugins propios y modificar los ya existentes.
- Crear filtros para mejorar el proceso.

# Introducción

Cuando PHP nació, fue considerado como un sencillo lenguaje para generar dinámicamente etiquetas de HTML. Tras la versión 3 de PHP, se añadió un compendio de funciones complejas para el manejo de cadenas, *arrays* y bases de datos, lo suficientemente completas como para crear sitios Web muy extensos.

Después de la creación de entornos completos en PHP, los desarrolladores comenzaron a pensar que no era muy útil mezclar código PHP con código HTML e inventaron una forma de independizar los dos lenguajes mediante el uso de plantillas. El primer entorno de plantillas que apareció fue FastTemplate escrito originariamente en Perl.

En un período corto de tiempo aparecieron muchos programas que facilitaban el uso de plantillas como vTemplate, Xtemplate, TemplatePower o SmartTemplate. Todos permiten sustitución de variables, bloques repetitivos, inclusión de múltiples plantillas y bloques opcionales con estructuras del tipo i f. La diferencia está en que muchos de estos sistemas utilizan expresiones regulares y hace muy complicado su uso. Además, la velocidad con la que se generan las plantillas es una propiedad a tener en cuenta en un entorno de trabajo en producción.

SmartTemplate apareció en 1999 y es el predecesor de lo que hoy conocemos como Smarty. Es un entorno muy similar a otros, pero incluye características muy avanzadas que lo hacen único, la compilación de plantillas. Esto significa que Smarty lee la plantilla y crea un *script* escrito en PHP. Una vez creado este *script,* se ejecuta y muestra el resultado en pantalla. Lo bueno es que la compilación sólo se hace la primera vez; las veces siguientes Smarty comprueba si hay nuevos datos que mostrar y, si no los hay, se utiliza la plantilla escrita en PHP, por lo que la ejecución se hace muy rápida. Algunas de las características generales de Smarty son:

- Extremadamente rápido.
- Muy eficiente, porque el parse de PHP hace todo el trabajo.
- Sólo se compilan las plantillas que cambien los datos.
- Podemos realizar funciones propias y modificadores de variables. Esto hace que Smarty sea extensible por el usuario.
- Se pueden utilizar varios delimitadores como {}, {{}j, <!—{}—>.
- Las estructuras de control son manejadas directamente por el parse de PHP, por lo tanto pueden ser tan complejas como quiera.

- Se puede incluir código PHP dentro de nuestras plantillas.
- Funciones configurables para el manejo de la caché.
- Arquitectura de *plugins* ampliable.

## Instalación de Smarty

La instalación del entorno de Smarty es muy sencillo y pasa por descargar de la Web http://smarty.php.net la última actualización del sistema. El archivo comprimido consta de varias carpetas, que deberían estar dentro del directorio de librerías de PHP. También es posible guardar la carpeta de Smarty junto con nuestros programas, pero no es recomendable.

Las librerías del sistema de plantillas, incluidas dentro del directorio `libs`, son:

```
Smarty.class.php
Smarty_Compiler.class.php
Config_File.class.php
debug.tpl
```

Además de estos archivos, existen las carpetas `core` y `plugins`, que se encargan de controlar el flujo de los objetos de Smarty.

Las plantillas que creemos las vamos a guardar en un directorio llamado `templates` y las plantillas compiladas deberán estar en `templates_c`.

**Advertencia:**

*La carpeta* `templates_c` *debe permitir la escritura, porque aquí se irán guardando las plantillas compiladas.*

## Utilización básica de Smarty

Durante el desarrollo del libro hemos visto algunas técnicas para escribir limpiamente nuestro código PHP mezclado con HTML. Vamos a ver una clase implementada por nosotros que genera una Web desde PHP y, después, vamos a ver cómo hacer lo mismo desde Smarty.

```
<?php
class  PaginaWeb
{
```

```
    prívate $titulo;
    function___construct($titulo)
    {
         $this->titulo = $titulo;
    }
    prívate function cabecera($titulo)
    {
         echo "<html>";
         echo "<head>",-
         echo  "<title>$titulo</head>";
         echo "</head>";
         echo "<body>";
    }
    prívate function cuerpo()
    {
         echo "Este es el cuerpo de nuestra Web<br>";
    }
    prívate function pie()
    {
         echo "Pie creado con PaginaWeb class";
         echo "</body>";
         echo "</html>";
    }
    public function creaWeb()
    {
         $this->cabecera($this->titulo);
         $this->cuerpo();
         $this->pie();
    }
}
$pagina = new PaginaWeb("Pagina Web generada con PHP");
$pagina->creaWeb();
?>
```

Lo que hace cada método es generar las etiquetas necesarias para crear una página Web. Imagine ahora un gran sitio Web con cientos de páginas Web con distinto tipo de información. Cambiar el aspecto visual de esa Web sería un trabajo muy laborioso de varios meses; primero buscando dónde se encuentran las etiquetas junto al código PHP y, segundo, modificando todo el código HTML para que cambie el aspecto visual. Las plantillas solucionan este problema en muy poco tiempo, separando el código de la presentación.

El resultado se puede ver en la figura 20.1.

La página Web siguiente, `plantillar.tpl`, es la misma que genera la clase anterior, pero esta vez creada como plantilla:

```
{* plantillal.tpl *}
```

```
<html>
<head>
<title>{$titulo}</title>
</head>
<body>
{$cuerpo}
{$pie}
</body>
</html>
```

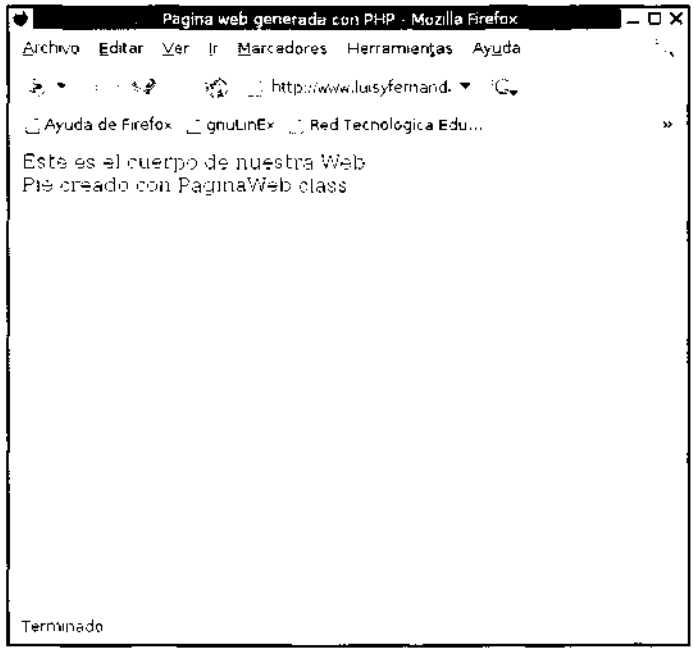



**Figura 20.1.** Página generada con PHP 5.

Todas las etiquetas conservan el estilo de HTML añadiendo símbolos de llave donde debe aparecer contenido dinámico. `{$titulo}`, `{$cuerpo}` y `{$pie}` son variables que reciben su valor desde el programa escrito en PHP.

**Nota:**

*Todo lo que está entre {* * } se considera un comentario.*

Para poder asignar valores a la plantilla tiene que hacer una instancia de la clase Smarty.

Puede modificar el objeto anterior para que haga uso de plantillas:

```
<?php
class PaginaWeb
{
  prívate $titulo;
  prívate $Web;
  function___construct($titulo)
  {
       $this->titulo = $titulo;
       requíre_once("Smarty/1ibs/Smarty.class.php");
       $this->Web = new Smarty;
  }
  prívate function cabecera($titulo)
  {
       $this->Web->assign("titulo",$this->titulo);
  }
  prívate function cuerpo()
  {
       $this->Web->assign("cuerpo","Este es el cuerpo de
nuestra Web");
  }
  prívate function pie()
  {
       $this->Web->assígn("pie", "Pie creado con PaginaWeb");
  }
  public function creaWeb()
  {
       $this~>cabecera($this->titulo);
       $this->cuerpo();
       $this->pie();
       $this->Web->display("plantilla1.tpl");
  }
}
$pagina = new PaginaWeb("Pagina Web generada con PHP");
$pagina->creaWeb();
?>
```

La clase es exactamente igual (con los mismos métodos), pero ahora se encuentra libre de etiquetas HTML. En su lugar aparecen asignaciones a variables de Smarty. El constructor instancia un objeto desde la clase Smarty y lo guarda en $this->Web. Las asignaciones de variables se hacen con el método assign() que tiene que recibir como primer parámetro el nombre de la variable escrita en la plantilla y, como segundo argumento, el texto que debe aparecer en lugar de la variable.

Después de asignar todos los valores tendremos que llamar al método display() con la plantilla como argumento, para que la Web aparezca en el navegador.

Si se fija ahora en el directorio de las plantillas compiladas, templates_c, verá que ha aparecido un archivo que contiene el siguiente código:

```
<?php /* Smarty versión 2.6.3, created on 2004-08-15 13:03:35
         compiled from plantillal.tpl */ ?>
<html>
<head>
<titlex?phpecho$this->_tpl_vars['titulo'],•?>
</title>
</head>
<body>
<?php echo $this->_tpl_vars['cuerpo']; ?>

<brx/br>
<?php echo $this->_tpl_vars['pie']; ?>

</body>
</html>
```

Como puede apreciar, la compilación de una plantilla de Smarty consiste en cambiar los valores entre llaves por código PHP que hace llamadas a las variables. Esto lo *convierte en un método muy rápido porque es el* mismo PHP el que se encarga de mostrar la plantilla ya compilada.

## Cuidado con los estilos CSS

Es muy útil utilizar estilos CSS para definir los colores, tipos de letra y demás objetos de presentación, que vamos a utilizar en nuestras páginas Web.

El problema surge porque su construcción es de la siguiente forma:

```
<style type="text/css">
<!--
P {color:green}
-->
</style>
```

Este estilo sirve para que los párrafos sean por defecto de color verde. El problema es que Smarty parsea todo lo que está entre símbolos de llave como {color:green}. En este caso se produciría un error porque el parse no identificaría ninguna función, estructura o variable de plantilla. Para solucionar esto se pueden añadir las etiquetas {literal} y {/lite-

ral} que, al ser parseadas, su contenido se identifica como un comentario y se salta hasta el siguiente juego de llaves.

```
{literal}
<style type="text/css">
<!--
P {color:green}
-->
</style>
{/literal}
```

## Llamada a varias plantillas

Casi todas las páginas Web pueden dividirse en varias partes bien diferenciadas: cabecera, menú, cuerpo, pie.

Con Smarty podrá crear una Web simple que incluya todas las partes de la Web como un único archivo HTML o crear 4 plantillas distintas y 1 que las unifique, por lo que sería más sencillo cambiar el aspecto de cada una de las partes.

La función include file de Smarty permite añadir plantillas a un fichero. La plantilla principal puede ser de la siguiente forma:

```
<html>
<head>
<title>{$titulo}</title>
</head>
<body>
{include file="cabecera.tpl"}
{include file="cuerpo.tpl"}
{include file=$pie}
</body>
</html>
```

De esta forma podemos modificar alguna de las plantillas y, automáticamente, todas las Web que hagan referencia a la misma cambiarán su aspecto.

Como puede ver, la plantilla que genera el pie de página es una variable, por lo tanto, podemos pasar desde PHP el nombre de cualquier plantilla en función de algún cálculo interno o de algún dato de MySQL, etcétera.

## Variables

Smarty acepta muchos tipos de variables diferentes.

Pueden ser utilizadas para mostrarse directamente o como parámetro de alguna función o modificador, dentro de alguna estructura de control, etcétera.

```
{$nombre}
{$contacto.nombre}
{$contacto[0]}
{$persona->nombre}
{config_load_file="configuración.conf"}
<title>{#titulo#}</title>
```

El código anterior contiene diferentes tipos de variables *y* la forma de imprimir el dato que nos interesa. Para una variable del tipo *array* puede utilizar la forma asociativa `$contacto.nombre` o la forma indicativa `$contacto[0]`. También es posible mostrar las propiedades de un objeto haciendo referencia así `$persona->nombre`.

Por último, Smarty posee una función que permite cargar una configuración de un fichero. Con la instrucción `config_load_file` cargará en memoria la configuración del fichero que dé como argumento. El fichero debe estar escrito de esta forma:

```
titulo = "Pagina Web con PHP"
nombre = "Luis Miguel"
color = "#FFFFFF"
```

## Modificadores

Los modificadores se pueden aplicar a las variables, funciones o cadenas de caracteres. Éstos permiten embellecer el texto, recortarlo o adecuarlo a un formato determinado. Para utilizar un modificador, tiene que especificar la variable, seguido de un símbolo de tubería ( | ) y el nombre del modificador.

```
<html>
<head>
{* cambia el texto a mayúsculas *}
<title>{$titulo|upper}</title>
</head>
<body>
{* Recorta el texto a 10 caracteres y le suma tren puntos
suspensivos *}
{$cuerpo¡trúncate:10:"..."}
<brx/br>
{$pie}
</body>
</html>
```

La plantilla anterior contiene el modificador upper, que convierte todos los caracteres que recibe en mayúsculas. El modificador trúncate recorta el texto dejándolo en 10 caracteres y le añade puntos suspensivos, como puede apreciar en la figura 20.2. La tabla 20.1 contiene un conjunto de modificadores que pueden utilizarse en las plantillas.

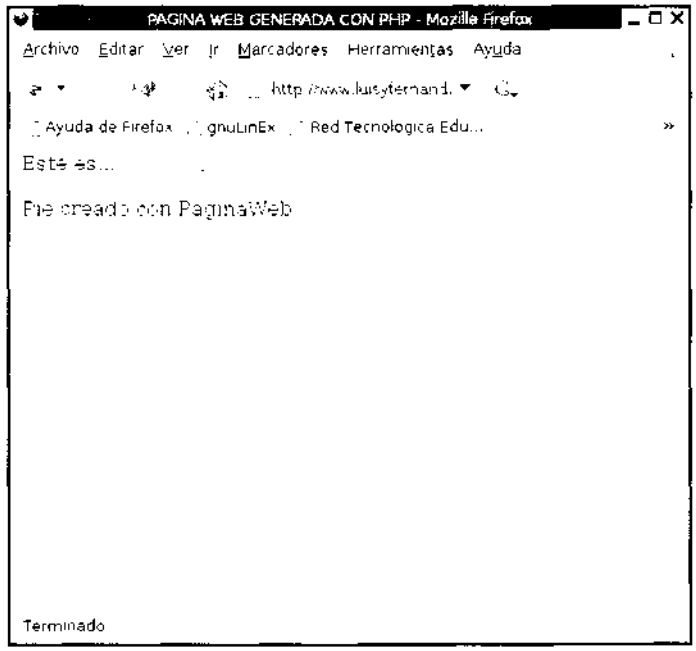



**Figura 20.2.** Plantillas con modificadores.

**Tabla 20.1.** Modificadores de Smarty.

| Modificador | Descripción | Parámetros |
|---|---|---|
| capitalize | Convierte la primera letra de cada palabra a mayúscula. | Ninguno, |
| count_characters | Se usa para contar los caracteres de una variable, | Ninguno. |
| cat | Concatena el parámetro pasado a la variable. | 1. El texto a concatenar. |
| count_paragraphs | Cuenta el número de párrafos que contiene una variable. | Ninguno. |
| count_words | Cuenta el número de palabras. | Ninguno. |

| **Modificador** | **Descripción** | **Parámetros** |
|---|---|---|
| date_format | Imprime en un formato determinado la fecha y hora del sistema. | 1. Formato de salida. 2. Si la entrada está vacía acepeste parámetro por defecto. |
| default | Valor por defecto de la variable. Si no se asigna ningún valor la variable tendrá el valor por defecto. | 1. El valor por defecto, |
| escape | Añade caracteres de escape a la variable. | 1. html, htmlall, url, quotes, hex, hexentity, javascript. |
| indent | Añade varios caracteres a la izquierda de la variable. | 1. Número de caracteres a indentar. 2. Carácter elegido para indentar. |
| lower | Convierte a minúscula todos los caracteres. | Ninguno, |
| nl2br | Convierte los saltos de línea **a </br>.** | |
| regexreplace | Busca y reemplaza una expresión regular. | 1. Expresión regular. 2. Cadena de texto a reemplazar. |
| replace | Reemplaza una cadena de caracteres por otra. | 1. Cadena a reemplazar, 2. Cadena de texto que sustituirá al parámetro 1. |
| spacify | Inserta un carácter determinado entre los caracteres de la variable. Por defecto un espacio en blanco. | 1. Carácter, |
| string_format | Formatea un texto. | 1. Formato del texto como lo usa sprintf. |
| strip | Reemplaza todos los espacios, saltos de líneas y fabuladores por un espacio en blanco. | Ninguno, |
| strip_tags | Elimina todas las etiquetas. | Ninguno. |
| trúncate | Recorta una variable el número de caracteres que indiquen los parámetros. | 1. Número de caracteres a recortar, 2. Texto que debe aparecer si se recorta, 3. *True* permite recortar en mitad de una palabra. |
| upper | Convierte a mayúsculas todos los caracteres de una variable. | |

Es bueno saber que podrá aplicar varios modificadores a una misma variable. Tiene que añadirlos en el orden en el que quiera que se ejecuten de izquierda a derecha.

```
{$cuerpo[upper|spacify j trúncate:20:"..."}
```

**Nota:**

*Si la variable es un* array *se aplicará el modificador a todos* s;;s *valores.*

## Funciones

El gestor Smarty tiene varias funciones que se pueden utilizar dentro del ámbito de las plantillas. Además, puede crear sus propias funciones o modificar las que ya están construidas.

## foreach

El bucle foreach se utiliza para mostrar un *array* simple. La sintaxis es algo distinta al bucle de PHP.

```
<table>
{foreach  from=$artistas  item=valor}
<trxtd>{ $valor} </tdx/tr>
{/foreach}
</table>
```

Lo puede utilizar para añadir los valores de un *array* a una tabla como en el código anterior. La construcción más básica de foreach recibe en el atributo from *el array* que contiene los valores. El atributo item crea una variable de Smarty, que recoge los valores del *array* y se puede utilizar para mostrar el valor unas líneas más abajo.

Otra construcción, un poco más compleja es:

```
<table>
{foreach name=buclel from=$artistas key=indice item=valor}
<trxtd>{$indice}: {$valor}</tdx/tr>
{/foreach}
</table>
```

Funciona exactamente igual que el anterior. El atributo key recoge el valor del índice del *array* y ñame da un nombre al bucle para poder acceder a él desde las variables propias de Smarty.

**Truco:**

*Se puede acceder al bucle mediante la variable global de Smarty. Esta variable es* `$smarty`. *Para acceder al bucle puede hacerlo de la siguiente forma* `$smarty.foreach.bucle1`.

El resultado de la ejecución de la plantilla es:

```
0: Paquito DRivera
1: Michel Camilo
2 : Jerry González
```

## if, elseif, else

La construcción `if` es tan flexible como la sentencia de PHP. Cada `if` debe tener una pareja del tipo `/if`.

```
{if $artistas[0]  != "Michel Camilo"}
  El primer artista toca el saxofón alto
{else}
  El primer artista toca el piano
{/if}
```

Además de los operadores básicos de PHP, también puede utilizar muchos otros como: `eq`, `ne`, `neq`, `gt`, `lt`, `lte`, `le`, `gte`, `ge`, `is even`, `is odd`, `is not odd`, `not`, `mod`, `div by`, `even by`, o `odd by`.

El código siguiente es similar al anterior:

```
{if $artistas[0] ne "Michel Camilo"}
  El primer artista toca el saxofón alto
{else}
  El primer artista toca el piano
{/if}
```

## php incluido en plantillas

Puede añadir a una plantilla código PHP encerrado entre las etiquetas de Smarty {php} y {/php¡. Si recuerda la filosofía de las plantillas, separar el código de la presentación, no se me ocurre ninguna excusa para utilizar esta habilidad en algún proyecto; por si acaso alguien le encuentra algún sentido, su uso es muy sencillo:

```
{php}
//Incluir aquí algún código escrito en PHP
include_once("logotipos.php");
{/Php}
```

## assign

Crea una variable y le asigna un valor determinado.

```
{assign var="variable" value="Michel Camilo"}
El valor de $variable es {$variable}
```

Se utiliza en tiempo de ejecución de la plantilla.

## counter

Se utiliza para imprimir un contador en pantalla.

```
{counter start=0 skip=3 print=false}<br>
{counter}<br>
{counter}<br>
{counter}<br>
```

El atributo start indica el número desde el que va a empezar el contador y skip el número que debe sumarse al contador cada vez que exista una ocurrencia. Cada vez que el parse encuentre {counter} se incrementará el contador y el navegador mostrará el resultado.
Para el ejemplo anterior podemos obtener:

```
3
6
9
```

## cycle

Una función interesante que permite alternar entre varios valores pasados. Cada vez que se ejecuta, se recupera un valor distinto hasta el final de los valores. Cuando se llega al final de la lista se vuelve a empezar. Es muy útil cuando queremos mostrar valores en una tabla y que, cada línea de detalle muestre un color diferente.

```
<table>
{foreach name=buclel from=$artistas key=indice item=valor}
<tr bgcolor="{cycle valués="#EEEEEE,#d0d0d0"}"><td>{$indice}:
{$valor}</tdx/tr>
{/foreach}
</table>
```

En este caso el bucle genera nombres de artistas del Jazz Latino y los introduce en una tabla. Gracias a la función cycle se pueden diferenciar unos de otros por tener los colores de fondo de cada línea distintos.

# Opciones avanzadas de Smarty

Existen muchos tutoriales en Internet sobre el sistema de plantillas. Todos abarcan la forma de utilizar las funciones y variables del entorno, pero pocos se ocupan de las características que lo hacen único.

La arquitectura de Smarty está escrita en PHP. Los ficheros que contienen las funciones básicas para que el parse pueda realizar su trabajo están dentro de `core`. Pero la mayoría de las funciones de presentación están incluidas dentro de la carpeta de `plugins`.

## Plugins

Un *plugin,* dentro de la estructura de Smarty, es un pequeño programa que realiza una tarea determinada. Por ejemplo, todos los modificadores son *plugins* y cada uno se encarga de hacer algo con el texto. Veamos la estructura *del plugin* `capitalize` que, recordemos, modifica la primera letra de cada palabra y la convierte a mayúscula.

```
<?php
function smarty_modifíer_capitalize($string)
{
   return ucwords($string);
}
?>
```

El funcionamiento del *plugin* es obvio. Tan sólo hace uso de una función existente en PHP para mostrar el resultado. Puesto que es tan sencillo, vamos a crear un plugin que servirá para enfatizar nuestro texto en determinadas ocasiones.

```
<?php
  function smarty_modifier_enfatizar($string)
{
   return "<b>" . $string . "</b>";

}

?>
```

Para crear un *plugin* modificador cree un fichero con el nombre `modifier.nombre.php` y guárdelo en el directorio `deplugins`. En este caso tendrá que crear un archivo con el nombre `modifier.enfatizar.php`.

El nombre de la función también debe llevar un nombre estructurado. Nuestro modificador se llamará `smarty_modifier_enfatizar` y reci-

birá como parámetro la cadena de caracteres que lo utilice. Dentro de la función puede añadir el código que sea necesario para cumplir su propósito y después devolver el valor con la función return.

En nuestro caso devolvemos la misma cadena, pero añadiendo la etiqueta de HTML para mostrar el texto en negrita, por lo que quedará enfatizado.

También podemos añadir algún parámetro para elegir entre distintos tipos de énfasis.

```
<?php
funetion  smarty_modifier_enfatizar($string, $parametrol)
{
   switch ($parametrol) {
        case 1:
             return "<b>" . $stríng . "</b>";
             break,-
        case 2:
             return "<em>" . $string . "</em>";
             break;
        case 3:
             return "i i ¡" . $string . "!!!";
             break;
        default:
             return "<b>" . $string . "</b>";
   }
}
?>
```

Ahora tenemos tres formas distintas de mostrar un texto que llame la atención. Para invocar desde la plantilla cualquiera de los tipos de énfasis habrá que separar el modificador con un símbolo de dos puntos ( : ).

```
{$valor|enfatizar:3}
```

## Filtros

Los filtros son funciones que realizan una tarea determinada en nuestra plantilla justo antes o después de que sea compilada. Se pueden considerar los pre-filtros y los post-filtros respectivamente. Existe un tercer tipo de filtro que se ejecuta en el momento de la llamada al método display ().

Crear un filtro para Smarty es tan sencillo como crear un *plugin*. Hay que seguir estrictamente la nomenclatura del archivo y del nombre de la función para que funcione correctamente.

```
<?php
function  smarty_j?ostfilter_HechoCon($compiled, &$smarty)
{
```

```
    return $compiled . "<b>Hecho con Smarty</b>";
}
?>
```

El filtro HechoCon () es una función que recibe como parámetros la fuente de la página Web ya compilada y el objeto Smarty que estamos utilizando. Nuestro filtro HechoCon devuelve la Web y añade un pequeño texto que indica que ha sido creado con Smarty.

Para poder utilizar el filtro tenemos que cargarlo antes de utilizar el método display() en nuestro código. Esto se hace con la función load_filter().

```
$this->Web->load_filter("pre","trim");
$this->Web->load_filter("post","HechoCon");
$this->Web->load_filter("output","compress");
```

El primer parámetro del método load_filter() indica el tipo de filtro que vamos a cargar y el segundo el nombre que recibe.

# Resumen

Algún lector se preguntará, después de entender correctamente lo bueno que es separar el código de la presentación, por qué he esperado tanto tiempo en contar el secreto de Smarty. La verdad es que es mejor aprender PHP paso a paso haciendo pequeños programas con HTML embebido, para después ir concibiendo la mejor forma de atajar los diferentes proyectos que nos propongan.

Con Smarty tiene una vía libre de exploración más, porque muchas de sus características no han podido ser tratadas en este capítulo. Le queda averiguar cómo utilizar la caché, los recursos (habilidad para leer de una base de datos una plantilla) y las numerosas funciones que hemos dejado en el tintero. Una vez que conozca la potencia de Smarty, no querrá mezclar nunca más su código con HTML.


Apéndice A

# Instalación de PHP 5 y MySQL

# Antes de comenzar

El primer paso para desarrollar programas en PHP 5 es encontrar un servidor donde hacer las pruebas del programa y, posteriormente, dejarlo almacenado para que los usuarios puedan utilizarlo. Este servidor debería estar disponible las 24 horas del día con una conexión fija a Internet.

Como esto puede ser algo complicado al principio, vamos a ver los pasos necesarios para instalar un servidor de pruebas, donde escribir nuestros primeros *scripts.*

PHP 5 se distribuye como código fuente, listo para compilar en el sistema operativo que quiera. El problema es que es algo complicado si nunca ha compilado un programa. Por eso, he elegido paquetes ya compilados que sólo necesitan ser instalados para comenzar a funcionar.

Vamos a instalar los siguientes programas:

- Servidor Web Apache.
- PHP 5.
- MySQL 4.

# Instalación en MacOSX

Uno de los más excitantes desarrollos *Open Source* ha sido la apertura parcial del código de la plataforma Macintosh. Gracias a esto muchos usuarios están contribuyendo con la creación de paquetes de instalación, como el de PHP 5.

## Apache

El servidor Web Apache es necesario para enviar las peticiones HTTP de los clientes.

Afortunadamente, Apache viene ya instalado en MacOSX. Para ponerlo en funcionamiento tendrá que seguir los siguientes pasos:

- Abra una Terminal.
- Entre como súper usuario con el comando su.
- Introduzca la *passzvord* de súper usuario.
- Escriba `apachectl start`.

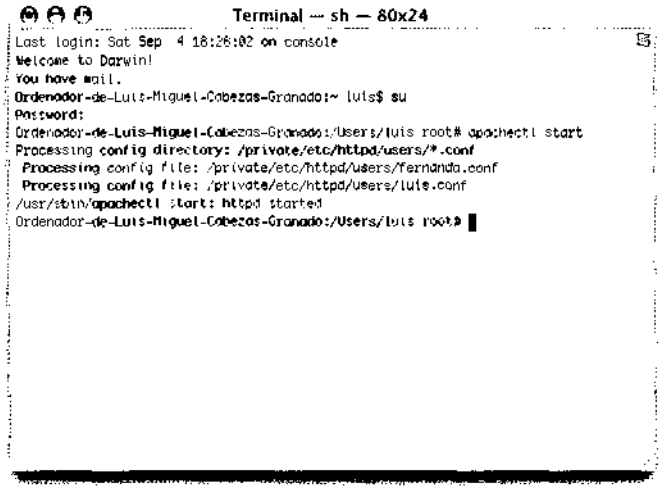



**Figura A.1.** Pasos en un terminal MacOSX.

La figura A.1 muestra los pasos realizados en el terminal para iniciar el servidor Web.

A partir de ahora puede probar si funciona Apache escribiendo en el navegador la dirección `localhost`, como muestra la figura A.2.

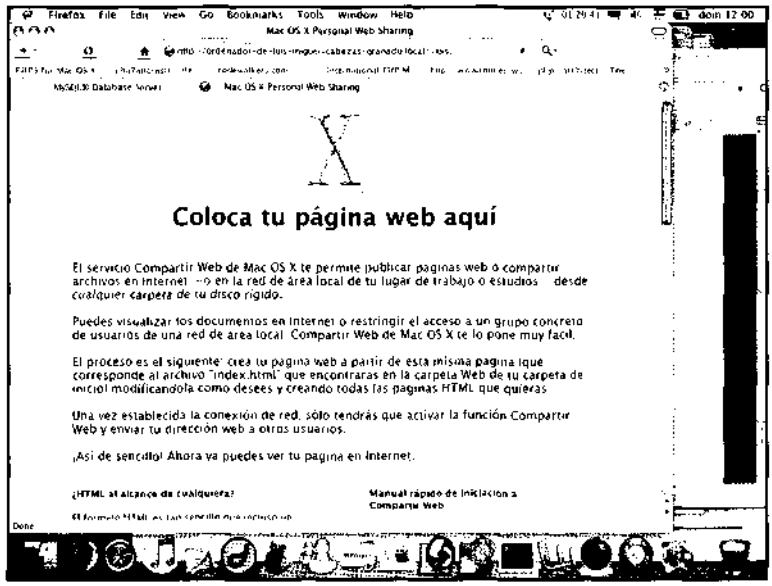



**Figura A.2.** Servidor Web Apache en MacOSX.

## PHP5

Existe una Web dedicada al mundo de MacOSX, que se actualiza todos los días.

Su administrador, Liyanage, es el encargado de crear paquetes con todas las versiones de PHP 5 que ven la luz, que en el momento de redactar el libro es la 5.0.1.

Para instalar PHP 5, tendrá que descargar el paquete de:

http://www2.entropy.ch/download/Entropy-PHP-5-0.1-1.dmg

Una vez copiado el archivo en su sistema, sólo tiene que hacer doble clic en él y seguir los pasos de instalación del sistema operativo, como puede ver en la figura A.3.

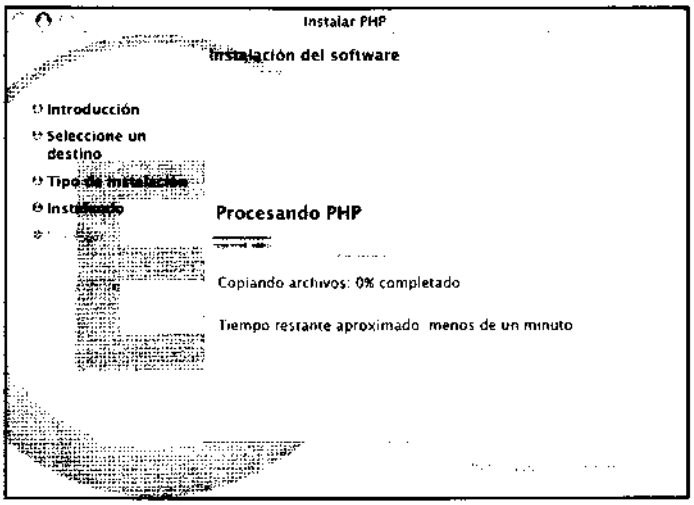



**Figura A.3.** Instalación de PHP 5.

## MySQL 4

Los pasos son exactamente igual que en la instalación de PHP 5.

El primer paso es descargar el paquete para MacOSXmysql-standard-4.0.20-apple-darwin7.3.0-powerpc.dmg, que se encuentra en la sección de descargas de la Web www.mysql.com.

El paquete se instala como cualquier paquete de MacOSX, simplemente haciendo doble clic.

## Comprobación final

Para comprobar que todo funciona correctamente tendrá que seguir los pasos siguientes:

- Abra una Terminal.
- Entre como súper usuario con el comando su.
- Introduzca la *password* de súper usuario.
- Reinicie la ejecución de Apache para que acepte la nueva configuración de PHP 5, con el comando apachectl restart.
- Inicie MySQL con el comando /usr/local/mysql-standard-4.0.20-apple-darwin7.3.O-powerpc/bin/mysqld -u root.
- Cree una Web de prueba y guárdela en la ruta del usuario para páginas Web. Normalmente es /Users/usuario/Sites/.
- Abra el navegador y escriba la dirección del servidor junto con el nombre de la Web. La dirección será parecida a esta: localhost/-usuario/información . php.

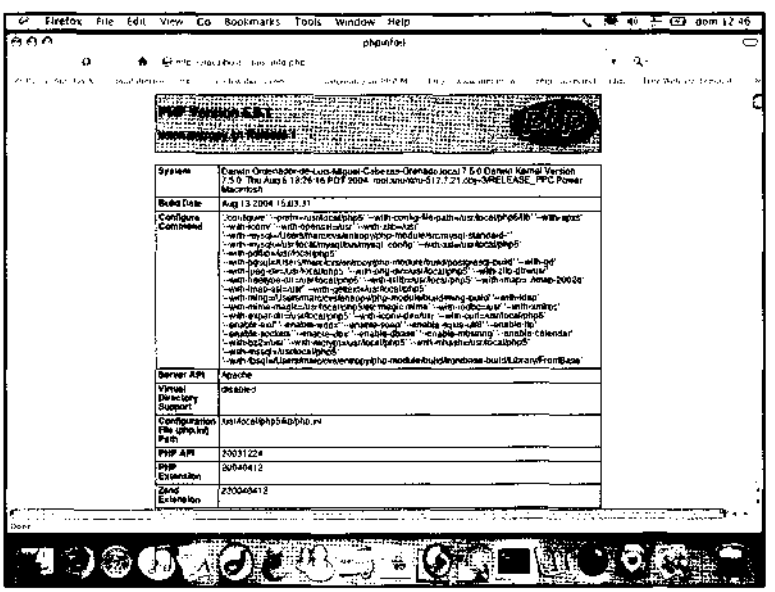


**Figura A.4.** Comprobación del sistema.

El programa que hemos escrito para comprobar que todo funciona correctamente y muestra el contenido de la figura A.4 es:

```
<?php
```

```
phpinfo();
?>
```

En la figura A.4 puede ver el funcionamiento de todo el sistema instalado.

## Instalación en Windows

Instalar un sistema completamente funcional en un sistema operativo Windows puede ser tan sencillo o tan complicado como quiera. Existe un paquete de aplicaciones, llamado XAMPP, que contiene todo lo necesario para funcionar con Apache, PHP 5 y MySQL.

En la Web www.apachefriends.org puede encontrar un archivo de instalación de Windows.

Después de descargar el archivo, tiene que hacer doble clic y seguir las instrucciones impresas en pantalla, como muestra la figura A.5.

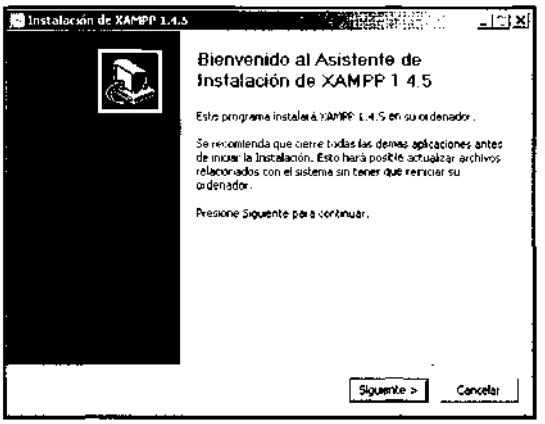



**Figura A.5.** Instalación de XAMPP en Windows.

Una vez instalado en la ruta lnicio>Programas>apachefriends>xampp verá tres apartados importantes:

- **xampp basic start:** Se encarga de iniciar Apache, PHP 5 y MySQL.
- **xampp basic stop:** Para la ejecución de los servidores.
- **xampp httpdoc folder:** Carpeta donde tiene que guardar todos sus programas escritos en PHP 5.

Para probar que todo funciona, puede ejecutar el programa xampp basic start, abrir un navegador y escribir como dirección localhost. Si

todo se instaló correctamente podrá ver una Web parecida a la de la figura A.6.

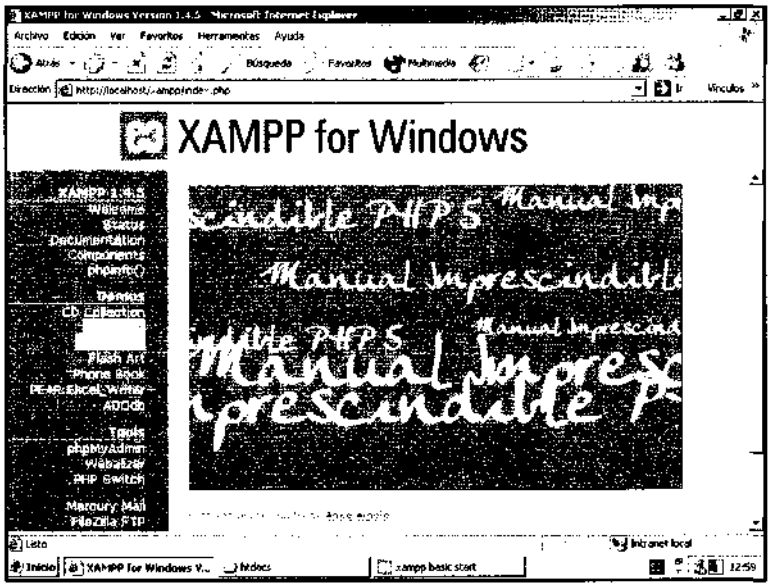



**Figura A.6.** XAMPP funcionando en Windows.

## Instalación en gnuünux

Existen en el mercado varias versiones de gnuLinux como, gnuLinEx 2004, SuSe, RedHat, Gentoo. Cada una de ellas posee un sistema de paquetes distinto. Puesto que no tenemos espacio para describir 3 formas diferentes de instalar PHP 5, nos conformaremos con describir un tema general que sirve para todos los sistemas gnuLinux.

Como ya puede imaginar, nos vamos a apoyar en XAMPP para gnuLinux.

Una vez descargada la versión para gnuLinux en el disco duro tendrá que descomprimirla en un directorio. El archivo comprimido debería llamarse `xampp-linux-1.4.7.tar.gz`. Para instalarlo debe seguir los pasos siguientes:

- Descomprima con el comando `gzip -dxampp-linux-1.4.7.tar.gz`.
- Vuelva a descomprimir el archivo, esta vez con `tar xvf xampp-linux-1.4.7.tar`.

- Se creará una carpeta llamada lampp.
- Copie la carpeta al directorio /opt con el comando cp -r lampp /opt.
- Ya lo tiene instalado. Para ejecutarlo debe ser súper usuario e invocar el comando /opt/lampp/lampp start.

La figura A.7 muestra un terminal con el comando de inicio.

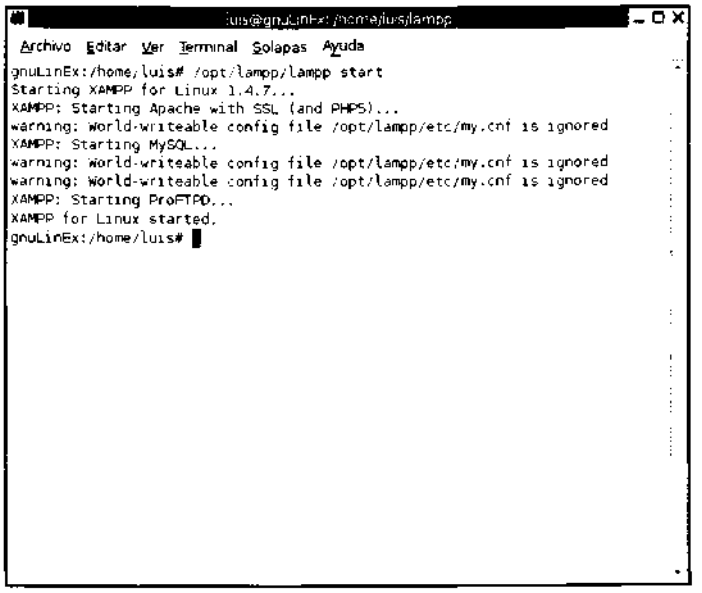



**Figura A.7.** Inicio de XAMPP en gnuLinEx.

En la figura A.8 puede ver la página de inicio de XAMPP, tras poner en el navegador localhost como dirección Web.

Todas las páginas Web que genere tendrá que guardarlas en el directorio /opt/lampp/htdocs para poder verlas a través del navegador.

## Recomendación final

La forma más rápida y sencilla de hacer funcionar PHP 5 en su sistema es la que hemos visto, pero no es la mejor.

La forma más recomendable, que se escapa del enfoque de este libro, es compilar su PHP 5. De esta manera tendrá la posibilidad de añadir los

módulos y extensiones que realmente vaya a utilizar. La recomendación es que navegue por la Web www.php.net e investigue la forma de compilar su sistema.

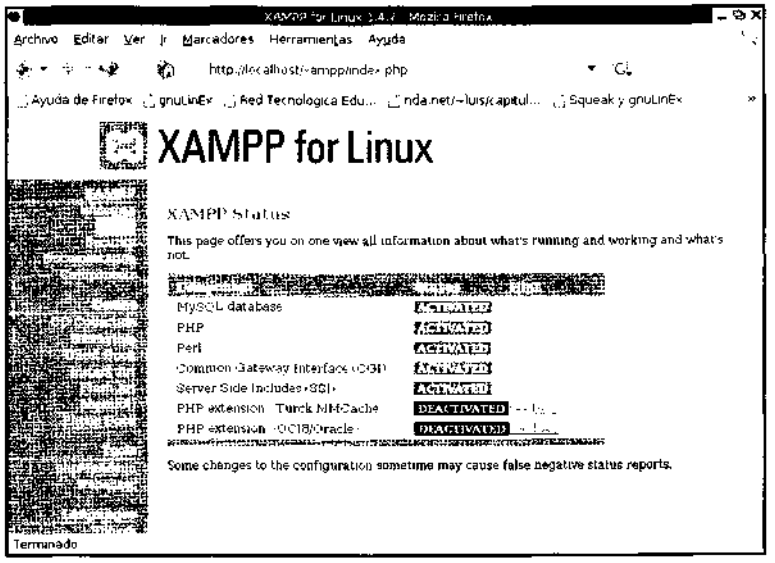
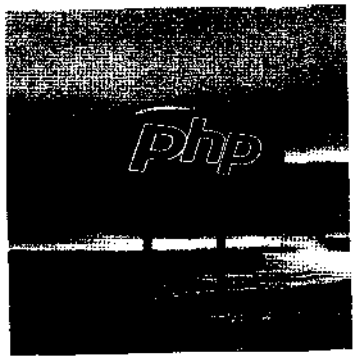



**Figura A.8.** XAMPP funcionando en gnuLinEx.




Apéndice B

# Configuración de php.ini

# Introducción

El archivophp . ini contiene información de configuración que puede modificar a su antojo. Es un fichero de texto que se lee cada vez que se inicia el servidor Web Apache.

Para averiguar la ruta del disco duro donde se almacena el archivo php. ini sólo tiene que fijarse en el apartado Configuration File (php . ini) Path de la Web que recibió al ejecutar phpinfo ().

La configuración contiene parejas de variables / valores, separadas por puntos y comas. Los valores que tenemos que asignar suelen ser rutas del disco duro, *1/0, Yes / No, On / Off o True / False.*

Cada vez que cambie alguna parte de la configuración, tendrá que reiniciar Apache para que los cambios surtan efecto.

## short_open_tag

Permite la forma corta de añadir *scripts* de PHP de la forma: <? ?>. Si quiere utilizar funciones XML es mejor dejarlo a *Off.*

## disable_functions

Desactiva funciones que nos resulten peligrosas, como el envío de correos, funciones de red, etcétera.

## max_execution_time

Tiempo máximo de ejecución de un *script* PHP en el servidor.

## error_reporting

Muestra el nivel de error que configure cuando un *script* falla. Los tipos de errores los puede ver en el capítulo 17.

## register_globals

Esta configuración ya no se utiliza a partir de la versión PHP 4.2 y está destinada a desaparecer.

Si la activa todas las variables serán pasadas como globales y no dentro de las variables súper globales.

### magic_quotes_runtime

Añade el símbolo de escape cuando se pasan variables con los símbolos de comillas simples o dobles.

### include_path

Añade rutas donde puede almacenar librerías que pueda utilizar sus programas.

## Resumen

Hemos visto las principales opciones de configuración del archivo php.ini. La configuración es mucho más extensa, pero no merece la pena indagar en otras opciones, a menos que vaya a dedicarse a la administración del Sistema.


Apéndice C

# Bibliografía

# Bibliografía

El desarrollo del libro ha sido posible gracias a la consulta de numerosas publicaciones escritas, páginas Web y revistas profesionales que han dado una visión actual de la programación en PHP 5. La bibliografía presente en este libro muestra un compendio de libros, revistas y páginas Web que pueden ser útiles al lector si decide ampliar sus conocimientos una vez terminado este libro.

## Libros de PHP 5


- **PHP5 and MySQL Bible.** Tim Converse y Joyce Park: Completo libro que abarca numerosos temas. Especializado en generar bases de datos con MySQL y PHP 5.
- **The PHP Anthology.** Harry Fuecks: El mejor libro sobre PHP escrito nunca. Abarca casi todos los temas importantes a la hora de crear un sitio Web profesional. Todos los *scripts* están cuidadosamente pensados y orientados a objetos. El autor es uno de los propulsores de los patrones de diseño.
- **Learning PHP5.** David Sklar: Contenido muy sencillo, pero directo. Para aprender PHP 5 desde 0.
- **PHP5 for Dummies.** Janet Valade. Un buen libro para aprender y profundizar en varias áreas. Desde luego el término *Dummies* no le hace justicia, porque a veces se hace algo complicado de seguir.
- **Proyectos Profesionales PHP.** Ashish Wilfred, Meeta Gupta, Kartik Bhatnagar. Proyectos orientados a PHP 4, pero muy bien construidos.
- **PHP** 4. Esteban Trigos. Libro muy sencillo para iniciarse en PHP 4.
- **PHP 5 Power Programming.** Andi Gutmans. De uno de los creadores de PHP. Muy bien enfocado y con múltiples temas.
- **Beginning PHP 5.** Equipo Wrox. Libro extenso sobre PHP 5. Cubre todo lo que se puede conocer sobre PHP 5. En Octubre saldrá la segunda parte, llamada Proffesional PHP 5.

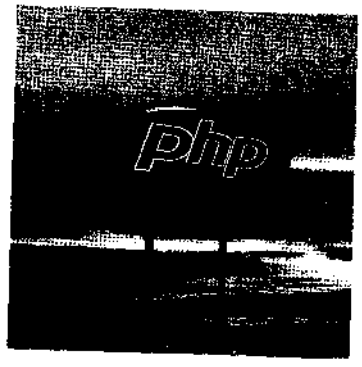

## Revistas profesionales


- **PHP Solutions:** Revista polaca traducida al español. Contiene muy buenos artículos sobre desarrollo actual con PHP.

- **PHP Magazine.** Revista alemana de gran alcance.
- **PHP Architect.** Revista canadiense, donde habitualmente escriben desabolladores como Harry Fuecks, Marco Tabini o Andi Gutmans. La misma editora de esta revista ha sacado el libro de estudio para presentarse al examen de Certificación de PHP.

## Páginas Web

- **www.phpsolmag.org:** Web de la revista PHP Solutions.
- **www.phpmag.net:** Web de la revista PHP Magazine.
- **www.phparch.com:** Web de la revista PHP Architect.
- **www.sinuh.org:** Comunidad GnuLinux de Extremadura.
- **www.phpbuilder.com:** Comunidad de usuarios con mucho código libre.
- **www.luisyfernanda.net:** Web del autor.
- **www.phppatterns.com:** La Web de Harry Fuecks.
- **www.php.net:** Web oficial de PHP.
- **www.zend.com:** La empresa que hay detrás de PHP. Ofrece cursos y seminarios *on-line.*
- **www.codewalkers.com:** Web con mucho código libre.




# Glosario

## A

- **API.** Acrónimo de Application Programming Interface. Conjunto de definiciones de los caminos que debe tomar un software para comunicarse con otro.
- **Applet.** Pequeño programa que funciona en el contexto de un programa más complejo en el servidor local de un usuario.
- **ASCII.** Acrónimo de American Standard Code for Information Interchange, un código de 7-bit que sustituye las letras del alfabeto romano por cifras y otros caracteres informáticos. Los caracteres ASCII permiten comunicar con ordenadores, que utilizan un lenguaje especial llamado binario formados por ceros y unos. Al escribir en el teclado, el ordenador interpreta cada letra escrita en lenguaje binario ASCII, para que puedan ser leídas, manipuladas, almacenadas o recuperadas. Los ficheros ASCII son denominados ficheros de texto.
- **ASP.** Siglas de Active Server Pages. Tecnología del lado del servidor de la compañía Microsoft, para generar páginas Web dinámicas.

## B

- **Bit.** Acrónimo de binary digit, un bit es la unidad más pequeña de datos que un ordenador puede manejar. Los bits se utilizan en distintas combinaciones para representar distintos tipos de datos. Cada bit tiene un valor 0 ó 1.

## C

- **C.** Lenguaje de programación creado en 1969 por Ken Thompson y Dennis Ritchie. Diseñado con el objetivo de ser un lenguaje orientado al diseño de sistemas operativos.
- **Callback.** Técnica de programación que permite a los programas hacer llamadas al software de bajo nivel.
- **CMS.** Siglas de Content Management System. Sistema utilizado para organizar la información y facilitar la colaboración para la creación de contenidos.
- **Cookie.** Información enviada por un servidor a un navegador Web, que puede ser recuperada posteriormente por algún programa.
- **CSS.** Siglas de Cascading Style Sheets. Lenguaje para describir el estilo de presentación de un documento estructurado escrito en HTML.

## D

- **Debian.** Distribución de software libre creada por voluntarios alrededor del mundo. Debian gnuLinux se basa en el kernel Linux con muchas herramientas básicas del proyecto GNU.
- **DOM.** Acrónimo de Document Object Model. Forma de representar un documento estructurado como parte de un modelo orientado a objetos. Es un estándar oficial de W3C.

## E

- **Engine.** Del inglés "motor". Parte principal de un programa.

## F

- **FTP.** Siglas de File Transfer Protocol. Parte del conjunto de protocolos para Internet. Es capaz de transmitir ficheros entre dos máquinas con distinto sistema operativo.

## G

- **GD.** Siglas de Graphics Library. Librería escrita en C por Thomas Boutell para la manipulación dinámica de imágenes.
- **GIF.** Acrónimo de Graphics Interchange Format. Formato estándar de imágenes que permite un máximo de 256 colores.
- **Gnome.** Acrónimo de GNU Network Object Model Enviroment. Entorno de escritorio para sistemas operativos de tipo UNIX bajo tecnología X Window.
- **GNU.** Acrónimo recursivo de GNU's Not Unix. El proyecto GNU fue iniciado en 1983 por Richard Stallman con la intención de crear un sistema operativo libre.
- **GnuLinEx.** Sistema operativo libre creado en Extremadura basado en gnuLinux.
- **GnuLinux.** Sistema operativo libre basado en los programas de GNU y el kernel (núcleo) de Linus Torvalds.

## H

- **HTML.** Acrónimo de Hyper Text Markup Language. Lenguaje de descripción de documentos utilizado para publicar información en la World Wide Web.
- **HTTP.** Siglas de HyperText Transfer Protocol. Desarrollado por W3C, es el método primario usado para comunicar información en la WWW.

## I

- **IMAP.** Acrónimo de Internet Message Access Protocol. Capa de aplicación usada para acceder al correo electrónico en un servidor remoto.
- **IMAP.** Acrónimo de Internet Message Access Protocol. Protocolo de red para el acceso a los mensajes electrónicos almacenados en un servidor.
- **Internet.** Sistema que aglutina las redes de datos de todo el mundo, uniendo miles de ellas mediante el protocolo TCP/IP. El mayor conjunto que existe de información, personas, ordenadores y software funcionando de forma cooperativa. La "i" mayúscula la diferencia de una Internet convencional, que simplemente une varias redes. Al ser única se la conoce también simplemente por "la Red".
- **ISO.** Acrónimo de International Organization for Standarization. Organización que ha definido un conjunto de protocolos diferentes, llamados protocolos ISO; y es responsable de la creación de estándares internacionales en muchas áreas, incluyendo la informática, las ecológicas y las comunicaciones. Está formada por las organizaciones de normalización de sus 89 países miembros.

## J

- **Java.** Lenguaje de programación orientado a objetos creado por James Goslin y otros ingenieros de Sun Microsystems.

- **Javascript.** Lenguaje de script orientado a objetos de uso común en las páginas Web. Desarrollado originariamente por Brendan Eich de Netscape Communications.
- **JPEG.** Acrónimo de Joint Photographic Experts Group. Formato estándar de imágenes que permite miles de colores.

## K

- **Kilobyte.** Unidad de medida de la capacidad de transmisión de una línea de telecomunicación equivalente a mil bytes aunque actualmente es usado como 1024 (dos elevado a la 10) bytes.

## L

- **Linux.** Se refiere al kernel del sistema operativo gnuLinux. Creado por Linus Torvalds en 1991, es utilizado como núcleo de GNU.

## M

- **MacOSX.** Sistema operativo propio de ordenadores Apple.
- **MIME.** Acrónimo de Multipurpose Internet Mail Extensions. Estándar de Internet para el intercambio de documentos a través del correo electrónico.
- **MySQL.** Sistema gestor de bases de datos multi-hilo, multi-usuario y basado en bases de datos relacionales. Se distribuye bajo dos licencias; una libre y otra comercial.

## N

- **Navegador.** Aplicación para visualizar todo tipo de información y navegar por el ciberespacio que cuentan con funcionalidades multimedia.

## O

- **OSI.** Acrónimo de Open Systems Interconnect. Es el protocolo en el que se apoya Internet. Establece la manera en la que se realiza la comunicación entre dos ordenadores a través de siete capas: Física, Datos, Red, Transporte, Sesión, Presentación y Aplicación.

## P

- **PDF.** Acrónimo de Portatile Document Format. Es el estándar de facto para la distribución e intercambio seguros y fiables de documentos y formularios electrónicos por todo el mundo.
- **PHP.** Acrónimo recursivo de PHP Hypertext Preprocessor. Lenguaje de Programación del lado del Servidor creado por Zeev Suraski y Andi Gutmans.
- **Píxel.** Unidad de medida relacionada al área ocupada por un banner en una página Web.
- **PNG.** Acrónimo de Portable Network Graphic. Formato estándar de imágenes que permite hasta 32 bits, de modo que puede contener un canal alfa.
- **Protocolo.** Conjunto de normas para una forma de comunicación.

## Q

- **QWERTY.** QWERTY quiere decir la sucesión de letras QWERTY, que es el comienzo de la fila superior de letras del teclado de un ordenador, más común. Un teclado QWERTY es, por tanto, el tipo de teclado que tenemos en nuestros equipos.

## R

- **RDF.** Siglas de Resource Description Framework. Especificación de un modelo de metadatos, mantenido por la W3C. Permite la difusión de contenidos a través de la Web.
- **RPC.** Siglas de Remote Procedure Call. Protocolo que permite a un programa ejecutar funciones o procedimientos almacenados en un servidor remoto.
- **RSS.** Siglas de Really Simple Syndication. Familia de XML basado en la distribución de contenidos para la Web.

## S

- **SAX.** Acrónimo de Serial Access Parser API para XML o de Simple API para XML. Un parser SAX maneja información XML como un flujo unidireccional.
- **Script.** Conjunto de instrucciones que al ser ejecutadas realizan una tarea determinada.
- **Servidor.** Aplicación Software que permite trabajar a muchos usuarios con tareas determinadas.
- **SGML.** Siglas de Standard Generalized Markup Language. Metalenguaje que permite definir lenguajes de marcas para documentos. Desciende de GML (Generalized Markup Language).
- **Smarty.** Motor de plantillas escrito en PHP que permite separar el código de la representación gráfica.
- **SOAP.** Acrónimo de Simple Object Access Protocol. Protocolo ligero para el intercambio de mensajes entre programas de ordenador.
- **SQL.** Siglas de Structured Query Language. Lenguaje muy popular para la creación, modificación y consulta de bases de datos.
- **SQLite.** Pequeña librería escrita en C que implementa una API para el manejo de ficheros mediante SQL.
- **SVG.** Acrónimo de Scalable Vector Graphics. Lenguaje para describir imágenes vectoriales en dos dimensiones, estáticas o dinámicas. La descripción se hace en XML.

## T

- **TCP/IP.** Siglas de la unión entre Transmission Control Protocol e Internet Protocol. Conjunto de protocolos con los que funciona Internet.

## U

- **UNIX.** Sistema operativo portable, multi-usuario y multi-tarea desarrollado originariamente por el grupo AT&T Bell Labs incluido Ken Thompson, Dennis Ritchie y Douglas McIlroy.

- **URL.** Siglas de Uniform Resource Locator. Estándar para direccionar recursos en Internet.

## V

- **Variable.** Representa un lugar en memoria donde se puede almacenar un dato.
- **VBScript.** Acrónimo de Virtual Basic Script. Lenguaje de programación para WWW desarrollado por Microsoft. Hay que destacar que VBScript y JavaScript de Netscape son muy similares.
- **Vínculo.** Apuntadores hipertexto que sirven para saltar de una información a otra, o de un servidor a otro, cuando se navega por Internet; o bien la acción de realizar dicho salto.

## W

- **W3C.** Siglas de World Wide Web Consortium. Consorcio que produce estándares o recomendaciones para la Web. Creado por Al Vezza, diseñador de la URL.
- **Weblog.** Sitio Web donde se recopilan cronológicamente mensajes de uno o varios autores, sobre una temática en particular.

## X

- **XML.** Siglas de Extensible Markup Language. Recomendación de W3C para crear lenguajes de marcas. Es un subconjunto de SGML capaz de describir diferentes tipos de datos.
- **XUL.** Acrónimo de XML-based User-interface Language. Aplicación de XML usada para describir capas de ventanas en el navegador Mozilla.

## Z

- **Zend.** Empresa de Zeev Suraski y Andi Gutmans dedicada al desarrollo del motor de PHP y aplicaciones para depuración y desarrollo.

# Índice alfabético


## A

abstracta, 11,35,172-173
AddAddress(), 320-321
AddAttachment(), 320
AddLink(), 329, 338
AddPage(), 327-329, 335-338
addslashes(), 227-228
AddStringAttachment(), 320
ALTER, 201
AND, operador lógico, 73
Apache, 358-359
applet (Java), 42-43
Argumentos, 10-11
  exceso, 102
  insuficientes, 101
  por defecto, 106
array, 133-148
  array(), 135
  arrays súper-globales, 158
  asignación directa, 134
  avanzar en un array, 140
  avanzar en un array, 140
  interacción, 138
  mezcla de valores, 145
  multidimensionales, 136
  propiedades, 137
array_flip(), 110
array_pop(), 145
array_push(), 145
array_reverse(), 144
arsort(), 146
asort(), 146
ASP, 31, 33, 40, 43

## B

BACKGROUND, 48
Bases de datos
  MySQL, 219
  SQLite, 199
BGCOLOR, 48-49, 186, 352
*Bit a Bit, 70, 74-75, 121*
Boolean, 56-57, 61
break, 84-85, 92, 156, 236
Bucles, 86-91
  do-while, 89
  for, 89
  while, 86

## C

callback, 251
capitalize, 348, 353
cascada if-elseif, 83
case, 84-85, 156, 236, 254
cell(), 329-331, 333-336
cellpadding, 151, 157, 186
cellspacing, 151, 157, 186
CHAR, 207-208, 210-214, 281
cheked, 152
chop(), 124
Clases, 162-179
  Alcance de variables, 168
  Clases abstractas, 172
  Definición, 162
  Funciones padre, 174
  Funciones, 177
  Herencia encadenada, 168
  Herencia, 165
  Instancia, 163
  Interfaces, 172
  Métodos estáticos, 173
  Métodos, 165
  Serialización, 176
  Sobrecarga, 176
class_exists(), 177
close(), 191
closedir(), 190-191
color(), 288-289, 335-336
comparación (operadores), 71
concatenar, 110, 115, 118, 348
constantes, 55, 57, 64
*continue*, 92-93
copy(), 188, 194-195, 288-289
count(), 137-138, 203, 229
count_characters, 348
count_paragraphs, 348
count_words, 348
counter, 352
CREATE, 201, 207-212, 214
createElement(), 270-272
createTextNode(), 271-272
CSS, 42-43, 46, 105, 345-346

## D

DATE (MySQL), 208, 211
date(), 32, 97-98, 275, 309
date_format, 349
DATETIME, (MySQL), 233
debug, 274, 341
define(), 63, 327, 329, 331, 338
defined(), 63
die(), 93
directorios, 181, 188, 190, 314
disable_functions, 368
display(), 344-245, 354-355
domDocument(), 255-259, 270
domNodeList, 256
doubleval(), 68
do-while, 89, 214, 216-217
DROP, 201
DTD, 264

## E

E_USER_ALL, 306
E_USER_ERROR, 306
E_USER_NOTICE, 306, 308
E_USER_WARNING, 306
ElementsByTagname(), 255-257
else, 82-83, 92, 106

elseif, 83-84, 121, 190, 191, 351
--enable-track-vars, 214
--enable-trans-sid, 214
encapsulación, 31, 272, 313
end(), 142, 294
endfor, 94
endif, 94
endswitch, 94
endwhile, 94
ereg(), 129-131, 316
ereg_replace(), 131
eregi(), 130-131
eregi_replace(), 131
error_log(), 308-309
error_reporting(), 306, 308, 368
Errores, 299-309
  Control de errores, 306
  Depuración de errores, 308
  Errores de usuario, 308
  Exception, 301-303
  Heredar Exception, 304-305
  Try / Catch, 303-304
escape (símbolos), 60, 227
Exception, 301-306, 309
exit(), 93
explode(), 184-185
ExplodeSlice(), 296
extends, 165, 168, 175, 176

## F

ftp_put(), 315-316
$_FILES, 158, 193-194, 289, 290
__FILE__, 64
fclose(), 184-186, 190
feof(), 184-185, 251
feof(), 184-185, 251
fgetc(), 185
fgets(), 184-185
file_exits(), 189, 316
filesize(), 184, 186, 196-197, 251
filetype(), 189
final (método), 167
FLOAT (MySQL), 232
FLOAT (SQL), 207
FONTPATH, 327, 329, 331, 334
Footer(), 333-334
fopen(), 78, 182-184, 186, 190
for, 89-92, 100, 104
foreach, 138-139, 143, 146, 150
formularios, 149-150, 151, 154
FPDF, 325-327, 329, 331, 333
fread(), 183-184, 186, 190, 251
fread(), 183-184, 186, 190, 251
FromName, 320-321
FTP, 183, 311-317
FTP_ASCII, 315-316
ftp_cdup(), 316
ftp_chdir(), 314, 316
ftp_close(), 316
ftp_connect(), 312-314
ftp_delete(), 316
ftp_exec(), 316
ftp_get(), 315-316
ftp_login(), 312-313, 317
ftp_mdtm(), 317
ftp_mkdir(), 317
ftp_nlist(), 314, 316-317
ftp_pwd(), 314, 317
ftp_rename(), 317
ftp_size(), 317
ftp_systype(), 317
func_get_arg(), 106, 109-110
func_get_args(), 106, 109-110
func_num_args(), 106, 109-110
Funciones
  definición, 99-101
  documentación, 98-99
  funciones de usuario, 99
  recursividad, 105-106
  valores, 96
fwrite(), 185-186

## G

$_GET, 150-154, 156-158, 238, 245, 288, 300-301, 304
GD, 280, 285-286, 288, 292
gDateLocale, 296
GET, 149-151, 155, 158, 289
get_class(), 177-178
get_class_method(), 178
get_class_vars(), 178
get_declared_class(), 177
get_magic_quotes_gpc(), 228
get_object_vars(), 178
get_parent_class(), 177
gettype(), 66-68
GIF, 53, 285-286
global (variables), 103-104, 170
GNU, 31
gnuLinEx, 26, 33, 363-365
gnuLinux, 6, 16, 25, 33, 40-42, 45, 76-77, 182, 188, 189
Graph, 292

## H

hacker, 31, 226
header(), 196, 238, 245, 286-288
Herencia, 165, 168, 177, 252
HTML, 39-43, 46-54, 90-91, 118, 125, 150-157, 188, 193-195, 226, 236, 249, 288-290, 319-322, 326, 339-347, 354-355
http, 183, 195, 235-238, 243-246, 265-267, 269-273, 275, 326, 337, 338, 341, 362
HTTP_USER_AGENT, 196

## I

if-else, 82-84, 92, 103, 106, 121, 129-130, 137, 142-143, 176, 186, 189-191, 242, 244-245, 251, 274-275, 282, 286, 300, 305, 308-309, 318, 320-321, 351
imageallocate(), 292
imagecolorallocate(), 291
imagecopy(), 291
imagecopyresampled(), 288-289
imagecreatefromjpeg(), 288
imagecreatefrompng(), 286-287
imagecreatetruecolor(), 288-289
imagejpeg(), 288-289
imagen (tipos MIME), 286-288
  image/bmp, 286
  image/gif, 286
  image/jpeg, 286, 288
  image/png, 286-287
  image/xml+svg, 286
imagepng(), 286-287
imagestring(), 291
IMAP, 317
include(), 105, 295, 346
include_once, 105, 351
include_path, 369
indent, 349
INSERT, 205-206, 209-210, 212, 214, 224, 226-228, 230-231, 259, 268, 281
instalación de PHP 5
  gnuLinux, 363-364
  Windows, 362-363
  MacOsX, 358-361
INTEGER (MySQL), 207-208, 210, 214, 232
integer, 56, 58, 66-68, 99, 101
interface, 172, 173, 209, 213, 300
INTERNET, 20, 24, 33, 36, 41, 43, 54, 261, 285, 312, 317, 320, 326, 353, 358
introspección, 140
intval(), 68, 284

is_a(), 178
is_dir(), 189-191
is_double(), 67
is_file(), 189-191
is_integer(), 67-68
is_link(), 189
is_string(), 67
is_subclass_of(), 178
isset(), 65-66, 103, 107, 186, 215

## J

Java, 40, 42-43, 116, 118, 300
Javascript, 42-43, 46, 349
JPEG, 286, 288, 332
JPG, 53, 266, 279, 285-286, 289, 292-296, 320, 322
JpGraph, 279, 292-296
jpgraph_line.php, 292-293
jpgraph_pie.php, 295-296
jpgraph_pie3d.php, 295-296
JSP, 33, 40, 43, 281

## K

key(), 143
KEY (SQLite, MySQL), 208-211
key (Smarty), 350, 352
ksort(), 146-147
krsort(), 146-147

## L

LAMP (Linux, Apache, MySQL, PHP), 23, 220, 364
libxml2, 260
LINK, 48-49
listas, 51-52
load(), 255-257, 259-260, 268
load_file(), 260, 268, 347
load_filter(), 355
loadXML(), 255
ltrim(), 124

## M

__METHOD__, 64
MacOsX, 25-26, 44, 45, 182, 280
Macromedia, 116, 151, 326
magic_quotes_gpc, 227
magic_quotes_runtime, 369
mail(), 317-323
Manual de PHP, 99
MAX (MySQL), 284
max_execution_time(), 368
MEDIUMINT (MySQL), 232
META, 48-50, 53, 186
method, 151-152, 157, 186
method_exists(), 178
methodCall, 273
methodName, 273
microtime(), 287
Miembros (funciones de objetos), 170-172
  Miembro privados, 171
  Miembro protegidos, 171
  Miembro públicos, 170
MIME, 195-197, 285, 286, 319
mktime(), 243
MTA, 317
MySQL, 200, 208-209, 218-233
mysql_affected_rows(), 230
mysql_close(), 221
mysql_connect(), 221-222
mysql_create_db(), 231-232
mysql_drop_db(), 231, 233
mysql_escape_string(), 228
mysql_fetch_array(), 222-223
mysql_insert_id(), 230-231
mysql_num_fields(), 229
mysql_num_rows(), 228-229
mysql_query(), 222, 227, 233
mysql_select_db(), 221-222, 232

## N

\n, 61, 124
NAME, 48-50, 53, 119, 151-152
new, 79, 164, 166, 169, 173, 175
next(), 140, 142-143, 214-215
NULL, 57, 62, 206, 208, 210, 214
numRows(), 217

## O

O lógico, 79, 90, 129, 134
objetos, 56, 114, 134, 161-163, 165, 167-177, 179, 185, 191
Open Source, 7, 34, 273, 358
operadores, 69-80
  Aritmético, 71
  Bit a bit, 74
  de asignación combinados, 75
  de Asignación, 70
  de Comparación, 71
  de ejecución, 76
  de supresión de errores, 77
  Lógicos, 73
  Ternarios, 74
  Unario, 71
ordenación de array, 146
  arsort(), 146
  asort(), 146
  krsort(), 146
  ksort(), 146
  rsort(), 146
  sort(), 146

## P

$_POST, 157-158, 186, 194, 226
include_path, 369
padre (clase), 165-168, 174-175
paquetes software, 280, 358, 360
parent, 175-177, 304-305
paréntesis (), 79, 97, 113-114
parse, 43, 45, 46, 64, 185, 192, 220, 247, 250, 251-255, 260
parse_ini_file(), 192
paso de valores, 31, 111-114, 168-169
  por referencia, 31, 111-112
  por valor, 31, 110, 114, 140
Path, 368
PEAR, 34, 64, 260, 310
Perl, 40, 340
Permisos (MySQL), 220
PHP (PHP Hypertext Preprocessor), 28, 30, 31, 32
php.ini, 45, 151, 191, 227-228
PHPArch Web, 338, 373
PHPBuilder Web, 373
phpinfo(), 46, 362, 368
PHPmag Web, 338, 373
PHPMailer, 319-321, 323
PHPSESSID, 240-241
PHPSolutions Web 338, 373
pi(), 96, 103-104
plantillas, 148, 339-352, 353, 355
PNG, 53, 284-287, 291, 332-333
POP, 24, 34, 317
POST, 149, 157-158, 187, 227
PostgreSQL, 34, 208, 220, 233
preserveWhiteSpace, 255-257
prev(), 142-143, 217-218
print(), 59, 352
printf(), 59, 349
procedural (programación), 162, 213

## R

\r, 61, 318, 319
RADIO, 152
rand(), 96, 287
range(), 136
readfile(), 196-197
register_globals, 151, 368

registrar variables, 243, 251
regular (expresiones), 127-131
  Definición, 127-129
  Reemplazar patrones, 131
rename(), 188
require(), 105
require_once(), 105, 187, 191
reset(), 142, 143
return, 100, 101, 103-104, 106
rsort(), 147
RSS, 261, 263-265, 267-272, 277

## S

$_SESSION, 158-159, 239, 241
save(), 258-259, 271-272
SAX, 34, 247, 250-253, 255, 260
SELECT, 200, 202-206, 212
Send(), 320-321
serialize(), 177
session_destroy(), 241-242
session_id(), 241
session_name(), 241
session_start(), 237, 239, 241
SET (SQL), 206, 211, 215, 225
set_error_handler(), 307-308
set_time_limit(), 313-314
SetAngle(), 295-296
setAttribute(), 258-259, 270
SetAutoPage(), 330
SetCenter(), 296
SetColor(), 292-294
setcookie(), 243-245
SetDrawColor(), 336-337
SetFillColor(), 335-336
SetFont(), 327, 329, 331, 333
SetLegends(), 294, 296
SetLink(), 338
setScale(), 292
setShadow(), 295, 296
SetSize(), 296
SetTextColor(), 336-337
settype(), 67
SetX, 330, 331, 333
SetY, 330, 333, 334
SimpleXML, 28, 34, 247, 250
simplexml_load_file(), 260, 268
sizeof(), 137
SMALLINT (SQL), 207, 232
SMTP, 24, 317
SOAP, 28, 260-261, 266, 277
sort(), 146-147
SQL, 199-205, 209, 211, 213
SQL_ASSOC, 222-223
SQLite, 28, 36, 199-201, 203
SQLITE_ASSOC, 281
sqlite_fetch_array(), 212-213
sqlite_last_insert_rowid(), 211
sqlite_open(), 209, 211-212, 281
sqlite_query(), 210, 212-214
sqlite_single_query(), 209-211
static, 104
str_replace(), 126
strcmp(), 121
string (cadena de caracteres), 59, 60, 66-68, 99, 116, 118
strip_tags(), 349
strlen(), 117, 120, 124, 301-302
strpos(), 120-121, 196, 301-302
strstr(), 122, 301-302
strtolower(), 126
strtoupper(), 126
strval(), 68
substr(), 99, 122-124
súper-globales (variables), 149-150, 152, 157-160, 239, 243

## T

\t (tabulador), 61
Tablas, 53, 200-201, 203-205
TCP/IP, 317
templates, 341, 345
TEXT (MySQL), 232
TEXT, 48
TEXTAREA, 187, 321
time(), 287
TIME, 208, 233
TIMESTAMP, 208
TINYBLOB (MySQL), 232
TINYTEXT (MySQL), 232
tipos de datos, 56
  array, 56, 66-68, 79, 96, 98, 99, 106-110, 131, 133
  boolean, 56, 57, 61, 232
  carácter, 56, 57, 59, 60, 67
  coma flotante, 56, 57, 59, 62
  entero, 56-59, 62, 67, 68, 71
  nulo, 56-57, 62, 206, 208, 210
  objetos, 56, 114, 134, 161-163, 165, 167-173
trim(), 124
truncate, 347-350
Try, 299, 301-304, 309

## U

ucfirst(), 127
ucwords(), 127, 353
unlink(), 188-189
UNIX, 31, 33, 76-77, 188-189
unserialize(), 177
unset(), 66, 138, 242
UPDATE (SQL), 200, 206, 211
URL, 150, 155, 241, 267, 270

## V

value, 74, 119, 151-152, 154
VARCHAR, 208, 232, 233
variables, 55-57, 59-63, 65, 67

## W

Web, 40-48, 52-54, 64, 105, 120, 150-152, 155-157, 159-160, 163-167, 170-172, 179, 185, 190, 193, 195, 197, 200-201, 218, 220, 223, 225, 227, 235-237, 239-240, 243, 245-246, 248, 254, 264-269, 271-277, 280, 282, 286-290, 293, 297, 307, 312, 322, 323, 326, 328, 337, 338, 340-342, 344-347, 355, 358-365, 368, 372-373
WHERE, 202, 204-207, 225-226, 231, 282
while, 86-90, 93-94, 140, 142-143, 146, 184-185, 190-191, 212-215, 217, 223, 229
Write(), 327-328, 330, 337-338

## X

XML, 148, 247-261, 263-265, 267-277, 286, 326, 335, 368
xml_get_error_code(), 251
xml_parser_create(), 250-251
xml_parser_free, 250-251
xml_set_character_data_handler(), 252
xml_set_element_handler(), 251
xml_set_handler(), 252
xml_set_object(), 251
XOR, 73, 79

## Z

Zend, 31, 34, 43-45, 63, 127-128, 162, 167-168, 267