PROGRAMACIÓN EN PHP5

Nivel Básico

Programación en PHP5. Nivel Básico

Carlos Vázquez Mariño
Ferrol, Septiembre de 2008

Mi agradecimiento a **Enrique Cabanas Olmo**, co-autor del manual *Programación en PHP. Nivel I*, en el que está basado este manual.

INDICE

1. Introducción a PHP

- *Funcionamiento de un Servidor Web*
- *Historia de PHP*
- *PHP frente a otros lenguajes*
- *Instalación de Apache*
- *Instalación de PHP*

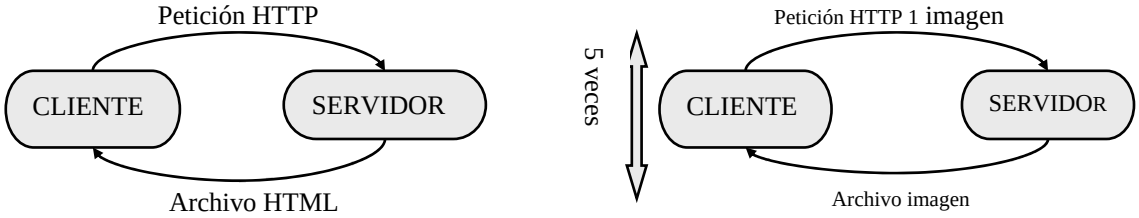
1.1. Funcionamiento de un Servidor Web

Dado que en este curso se va a tratar el desarrollo de sitios web dinámicos mediante PHP, antes de entrar en porque elegir PHP y no otro lenguaje de programación, y como funciona dicho lenguaje, hemos de comprender como funciona un servidor web cuando un usuario solicita una de esas páginas que nosotros vamos a construir con PHP.

Vamos a comenzar explicando como funciona un servidor web (como Apache, IIS, PWS, ...) cuando un usuario solicita una página HTML estática que contiene 5 imágenes en su contenido.

Para resolver dicha petición, el navegador del usuario debe establecer una conexión HTTP distinta para cada elemento que se desea solicitar al servidor web, es decir, una conexión HTTP para la página HTML y 5 conexiones HTTP adicionales para las imágenes (una por imagen).

Es decir, la situación vendría a ser como se muestra a continuación:



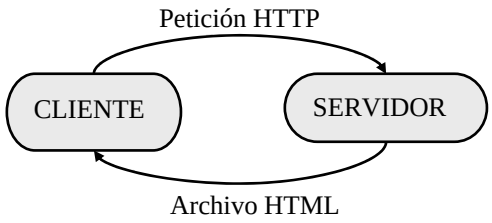
Si en lugar de páginas estáticas, las páginas son dinámicas, es decir muestran información obtenida de ficheros, bases de datos, u otras fuentes, pero que varía en función de una serie de parámetros sin la intervención del programador web, se nos presentan distintas alternativas para dar el carácter dinámico a las páginas: bien darles dinamismo en la parte de clientes (por ejemplo usando applets), o bien darles dinamismo en la parte del servidor (usando CGI, SERVLETS o lenguajes como PHP, ASP O JSP).

Si optamos por dar dinamismo en la parte del cliente, el funcionamiento cuando el usuario solicita una página que contiene un APPLET es bastante similar al visto para las páginas estáticas, es decir, se tendrá que realizar una conexión HTTP para obtener la página y tantas conexiones HTTP como APPLETS contenga dicha página. Un APPLET, que es un programa escrito en el lenguaje JAVA, se ejecuta en la propia

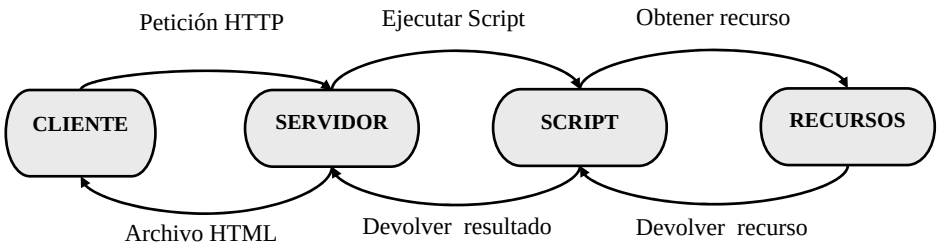
máquina del cliente, con las consiguientes desventajas de acceso a recursos ubicados en el servidor, y con la desventaja de que se tiene que ejecutar en una JVM (Java Virtual Machine) que hay que tener instalada en la máquina en la que se sitúa el cliente.

Su uso se reduce a dotar a las páginas de efectos visuales y sonoros, textos en movimiento, pequeñas utilidades para incluir en la página como un reloj, pequeños programas educativos, pequeños juegos interactivos, presentaciones multimedia, etc

Sin embargo, si optamos por dar dinamismo en la parte del servidor el funcionamiento es un poco diferente. Desde el punto de vista del usuario que solicita una página el funcionamiento es el ya visto, es decir, el usuario solicita una página y se le devuelve un archivo HTML (lo que genera una conexión HTTP al servidor) y tantas imágenes o elementos como tenga incluidos la página (una conexión HTTP por cada uno de estas imágenes o elementos).



Pero desde el punto de vista del servidor la cosa es un poco más compleja. Cuando se solicita una pagina web que contiene código denominado script, es decir, incluido dentro de la página HTML, escrito en PHP, ASP, JSP o cualquier otro lenguaje similar, el servidor web antes de devolver dicha página HTML, solicita al interprete de scripts que ejecute ese código como si de un programa se tratase y le devuelva un resultado. Ese resultado incrustado en el documento HTML original constituye el documento HTML devuelto al usuario. El interprete de scripts para ejecutar dicho código script y devolver un resultado puede acceder a multitud de recursos del servidor (bases de datos, ficheros, servicios de correo, ftp, news, ...) u de otros servidores. De esta forma el esquema anterior se complica para quedar de la siguiente forma:



1.2. Historia de PHP

PHP/FI

PHP es el heredero de un producto anterior, llamado PHP/FI. PHP/FI fue creado por Rasmus Lerdorf en 1995, inicialmente como un simple conjunto de scripts de Perl para controlar los accesos a su trabajo online. Llamó a ese conjunto de scripts 'Personal Home Page Tools'. Según se requería más funcionalidad, Rasmus fue escribiendo una implementación C mucho mayor, que era capaz de comunicarse con bases de datos, y

permitía a los usuarios desarrollar sencillas aplicaciones Web dinámicas. Rasmus eligió liberar el código fuente de PHP/FI para que cualquiera pudiese utilizarlo, así como arreglar errores y mejorar el código.

PHP/FI, que se mantuvo para páginas personales y como intérprete de formularios, incluía algunas de las funcionalidades básicas de PHP tal y como lo conocemos hoy. Tenía variables como las de Perl, interpretación automática de variables de formulario y sintaxis embebida HTML. La sintaxis por sí misma era similar a la de Perl, aunque mucho más limitada, simple y algo inconsistente.

Por 1997, PHP/FI 2.0, la segunda escritura de la implementación en C, tuvo un seguimiento estimado de varios miles de usuarios en todo el mundo, con aproximadamente 50.000 dominios informando que lo tenían instalado, sumando alrededor del 1% de los dominios de Internet. Mientras había mucha gente contribuyendo con bits de código a este proyecto, era todavía en su mayor parte el proyecto de una sola persona.

PHP/FI 2.0 no se liberó oficialmente hasta Noviembre de 1997, después de gastar la mayoría de su vida en desarrollos beta. Fue sucedido en breve tiempo por las primeras versiones alfa de PHP 3.0.

PHP 3

PHP 3.0 era la primera versión que se parecía fielmente al PHP tal y como lo conocemos hoy en día. Fue creado por Andi Gutmans y Zeev Zuraski en 1997 rescribiéndolo completamente, después de que encontraran que PHP/FI 2.0 tenía pocas posibilidades para desarrollar una aplicación comercial que estaban desarrollando para un proyecto universitario. En un esfuerzo para cooperar y empezar a construir sobre la base de usuarios de PHP/FI existente, Andi, Rasmus y Zeev decidieron cooperar y anunciar PHP 3.0 como el sucesor oficial de PHP/FI 2.0, interrumpiéndose en su mayor parte el desarrollo de PHP/FI 2.0.

Una de las mejores características de PHP 3.0 era su gran extensibilidad. Además de proveer a los usuarios finales de una sólida infraestructura para muchísimas bases de datos, protocolos y APIs, las características de extensibilidad de PHP 3.0 atrajeron a docenas de desarrolladores a unirse y enviar nuevos módulos de extensión. Sin duda, ésta fue la clave del enorme éxito de PHP 3.0. Otras características clave introducidas en PHP 3.0 fueron el soporte de sintaxis orientado a objetos y una sintaxis de lenguaje mucho más potente y consistente.

Todo el nuevo lenguaje fue liberado bajo un nuevo nombre, que borraba la implicación de uso personal limitado que tenía el nombre PHP/FI 2.0. Se llamó 'PHP' a secas, con el significado de ser un acrónimo recursivo - PHP: Hypertext Preprocessor.

A finales de 1998, PHP creció hasta una base de instalación de decenas de millares de usuarios (estimados) y cientos de miles de sitios Web informando de su instalación. En su apogeo, PHP 3.0 estaba instalado en aproximadamente un 10% de los servidores Web en Internet.

PHP 3.0 se liberó oficialmente en Junio de 1998, después de haber gastado unos 9 meses en pruebas públicas.

PHP 4

En el invierno de 1998, poco después del lanzamiento oficial de PHP 3.0, Andi Gutmans y Zeev Suraski comenzaron a trabajar en la reescritura del núcleo de PHP. Los objetivos de diseño fueron mejorar la ejecución de aplicaciones complejas, y mejorar la modularidad del código base de PHP. Estas aplicaciones se hicieron posibles por las nuevas características de PHP 3.0 y el apoyo de una gran variedad de bases de datos y APIs de terceros, pero PHP 3.0 no fue diseñado para el mantenimiento tan complejo de aplicaciones eficientemente.

El nuevo motor, apodado 'Motor Zend' (comprimido de sus apellidos, Zeev y Andi), alcanzó estos objetivos de diseño satisfactoriamente, y se introdujo por primera vez a mediados de 1999. PHP 4.0, basado en este motor, y acoplado con un gran rango de nuevas características adicionales, fue oficialmente liberado en Mayo de 2000, casi dos años después que su predecesor, PHP 3.0. Además de la mejora de ejecución de esta versión, PHP 4.0 incluía otras características clave como el soporte para la mayoría de los servidores Web, sesiones HTTP, buffers de salida, formas más seguras de controlar las entradas de usuario y muchas nuevas construcciones de lenguaje.

El día 13 de julio de 2007 se anunció la suspensión del soporte y desarrollo de la versión 4 de PHP, a pesar de lo anunciado se ha liberado una nueva versión con mejoras de seguridad, la 4.4.8 publicada el 13 de Enero del 2008. Según se ha anunciado se dará soporte a fallos críticos hasta el 2008-08-08.

PHP 5

El 13 de julio de 2004, fue lanzado PHP 5, utilizando el motor Zend Engine II (o Zend Engine 2). La versión más reciente de PHP es la 5.2.6 (1 de Mayo de 2008), que incluye todas las ventajas que provee el nuevo Zend Engine 2 como:

- ⇨ Mejor soporte para la Programación Orientada a Objetos, que en versiones anteriores era extremadamente rudimentario, con PHP Data Objects.
- ⇨ Mejoras de rendimiento.
- ⇨ Mejor soporte para MySQL con extensión completamente reescrita.
- ⇨ Mejor soporte a XML (XPath, DOM, etc.).
- ⇨ Soporte nativo para SQLite.
- ⇨ Soporte integrado para SOAP.
- ⇨ Iteradores de datos.
- ⇨ Manejo de excepciones.

PHP 6

Está previsto el lanzamiento en breve de la rama 6 de PHP. Cuando se lance esta nueva versión quedarán solo dos ramas activas en desarrollo (PHP 5 y 6), pues se abandonó el desarrollo y soporte de PHP 4 el 13 de julio de 2007.

Las diferencias que encontraremos frente a PHP 5.* serán:

- ⇨ Soportará Unicode.
- ⇨ Limpieza de funcionalidades obsoletas como register_globals, safe_mode...
- ⇨ PECL.
- ⇨ Mejoras en orientación a objetos.

1.3. PHP frente a otros lenguajes

Para seleccionar un lenguaje de Script las cuatro grandes características que debe cumplir son: Velocidad, estabilidad, seguridad y simplicidad.

- **Velocidad:** No sólo la velocidad de ejecución, la cual es importante, sino además no crear demoras en la máquina. Por esta razón no debe requerir demasiados recursos de sistema. PHP se integra muy bien junto a otro software, especialmente bajo ambientes Unix, cuando se configura como módulo de Apache, esta listo para ser utilizado.
- **Estabilidad:** La velocidad no sirve de mucho si el sistema se cae cada cierta cantidad de ejecuciones. Ninguna aplicación es 100% libre de bugs, pero teniendo de respaldo una increíble comunidad de programadores y usuarios es mucho mas difícil para lo bugs sobrevivir. PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.
- **Seguridad:** El sistema debe poseer protecciones contra ataques. PHP provee diferentes niveles de seguridad, estos pueden ser configurados desde el archivo .ini
- **Simplicidad:** Se les debe permitir a los programadores generar código productivamente en el menor tiempo posible. Usuarios con experiencia en C y C++ podrán utilizar PHP rápidamente.

Bueno otra característica a tener en cuenta seria la conectividad. PHP dispone de una amplia gama de librerías, y agregarle extensiones es muy fácil. Esto le permite al PHP ser utilizado en muchas áreas diferentes, tales como encriptado, gráficos, XML y otras.

Ventajas adicionales de PHP

- PHP corre en (casi) cualquier plataforma utilizando el mismo código fuente, pudiendo ser compilado y ejecutado en algo así como 25 plataformas, incluyendo diferentes versiones de Unix, Windows (95,98,NT,ME,2000,XP,...) y Macs. Como en todos los sistemas se utiliza el mismo código base, los scripts pueden ser ejecutados de manera independiente al OS.
- La sintaxis de PHP es similar a la del C, por esto cualquiera con experiencia en lenguajes del estilo C podrá entender rápidamente PHP. Entre los lenguajes del tipo C incluimos al Java y Javascript, de hecho mucha de la funcionalidad del PHP se la debe al C en funciones como fread() o srtlen(), así que muchos programadores se sentirán como en casa.
- PHP es completamente expandible. Está compuesto de un sistema principal (escrito por Zend), un conjunto de módulos y una variedad de extensiones de código.
- Muchas interfaces distintas para cada tipo de servidor. PHP actualmente se puede ejecutar bajo Apache 1.3 y 2.X, IIS, Lighttpd, Netscape servers (Sun Java System, Sun One, iPlanet, …),… . Otra alternativa es configurarlo como modulo CGI.
- Puede interactuar con muchos motores de bases de datos tales como MySQL, MS SQL, Oracle, Informix, PostgreSQL, y otros muchos. Siempre podrás disponer de ODBC para situaciones que lo requieran.
- Una gran variedad de módulos cuando un programador PHP necesite una interfase para una librería en particular, fácilmente podrá crear una API para esta. Algunas de

las que ya vienen implementadas permiten manejo de gráficos, archivos PDF, Flash, Cybercash, calendarios, XML, IMAP, POP, etc.
- Rapidez. PHP generalmente es utilizado como modulo de Apache, lo que lo hace extremadamente veloz. Esta completamente escrito en C, así que se ejecuta rápidamente utilizando poca memoria.
- PHP es Open Source, lo cual significa que el usuario no depende de una compañía específica para arreglar cosas que no funcionan, además no estás forzado a pagar actualizaciones anuales para tener una versión que funcione.

Algunas desventajas

- El manejo de errores no es tan sofisticado como otros lenguajes (ASP.Net, …) .
- No existe IDE o Debugger consolidado. Aunque en la actualidad existen proyectos varios entre los mas destacados está Eclipse (gratuito) o Zend Studio (de pago).

Cuando debes utilizar PHP y cuando debes utilizar otro lenguaje

Si estás desarrollando bajo una plataforma UNIX o Linux, debes elegir entre Perl y PHP, ambos excelentes. Para mucha gente PHP es más simple a la hora de escribir scripts, haciéndolo más productivo en proyectos.

En ambientes Windows compite muy de cerca con ASP.Net , aquí la elección se basa en asuntos un poco más técnicos y en la política que desee utilizarse para el sitio. ASP.Net junto a IIS es probablemente más estable que PHP con IIS. Pero en términos puramente técnicos, PHP bajo Windows Server es mucho más estable (además de ser más rápido y utilizar menos recursos). De cualquier manera ASP.Net ofrece una mejor integración con este ambiente sobre todo si se desea utilizar COM+.

1.4. Instalación de Apache

¿Que requerimientos necesitamos?

En este manual nos referimos a la instalación bajo sistemas operativos de la familia Windows, que será sensiblemente igual en toda la familia, aunque la configuración es común a prácticamente cualquier S.O.

Necesitamos un ordenador con un procesador de la familia x86 (cualquiera desde el 386 a la familia Pentium), y el protocolo TCP/IP instalado. La documentación nos indica que bajo Windows 95 necesitamos tener instaladas las winsock 2.

Obteniendo el software

Apache es un proyecto Open Source de la fundación apache.org. Puedes obtener el programa en su sitio web, http://www.apache.org/.

Aunque se trata de un proyecto orientado al sistema operativo unix/linux, existen versiones del servidor web para Windows. Apache mantiene tres líneas de su servidor web: 1.x, 2.0 y 2.2. En sistemas Windows la propia fundación aconseja usar un servidor web de la línea Apache 2, reescrito para adaptarse a las peculiaridades de Windows. El uso de cualquiera de las líneas de Apache 2 vendrá determinado por los requisitos

específicos que se le exijan al servidor web, en nuestro caso podemos usar un servidor de cualquiera de las dos líneas: Apache 2.0 o Apache 2.2.

Instalando los archivos

La versión binaria de apache para Windows viene, desde la versión 1.3.17, en formato msi; se trata de un instalador de Microsoft, incorporado de serie en Windows Me y Windows 2000.

Para otros sistemas operativos Windows, hay que bajarse previamente el MSI INSTALLER. Por ejemplo, para Windows 95 o 98 se puede obtener (gratis) en esta dirección:

http://download.microsoft.com/download/platformsdk/wininst/1.1/W9X/EN-US/InstMsi.exe

La instalacion de los archivos en nuestro disco duro tiene nula dificultad.

La única precaución destacable es la de desinstalar cualquier versión previa de apache que tengamos, lo que conseguimos fácilmente desde inicio/panel de control/instalar desinstalar programas.

Bajamos el ejecutable de la red a un directorio temporal, por ejemplo c:\temp, y una vez allí lo ejecutamos con un doble click de ratón, comenzando un proceso típico de instalación, en el que apenas habremos de intervenir: tras aceptar los términos de la licencia y pasar un readme file, nos solicita que introduzcamos el nombre de la red, nombre del servidor y email del webmaster; si tienes un nombre de dominio y las dns correctamente configuradas (es decir, si tu apache va a operar en internet) puedes meter aqui los datos pertinentes.

Si aun no sabes de quelocal estamos hablando al referirnos a las DNS, o simplemente piensas usar apache como servidor de pruebas, puedes especificar localhost como nombre del dominio y del servidor, y cualquier email como direccion del webmaster. Seleccionamos que se instale Apache para todos los usuarios en el puerto 80.

A continuación corresponde elegir el tipo de instalación que queremos, es decir, "tipica", que instala todo menos el código fuente (es decir, el programa y la documentación) o "custom", que permite por ejemplo no instalar la documentación. Elegimos la opción personalizada (“custom”). Lo siguiente que nos pregunta es la carpeta donde queremos que se instale. Por defecto, apache crea una carpeta en c:\Archivos de programa, siendo la ruta completa *C:\Archivos de programa\Apache Group\Apache*, aunque podemos elegir otro destino. Para el curso elegimos *C:\WAMP\Apache22* suponiendo que en C: ya existe la carpeta *WAMP*.

Los directorios de apache

Un servidor apache tiene dos directorios esenciales, o tres, segun la instalación elegida:

- **ServerRoot** es el directorio donde están los programas que ejecutan el servidor. Por defecto se instala en *C:\Archivos de programa\Apache Group\Apache*, con sus subdirectorios correspondientes. En nuestro curso *C:\WAMP\Apache22*
- **DocumentRoot** es el directorio donde se instalan los archivos que serán servidos a los visitantes; las páginas web. Por defecto está en *C:\Archivos de programa\Apache Group\Apache\htdocs*. En nuestro curso *C:\WAMP\Apache22\htdocs*
- **Source tree** es el directorio donde está el código fuente del servidor, si lo hemos instalado (en *C:\Archivos de programa\Apache Group\Apache\src*). Nosotros no lo hemos instalado.

Prueba de funcionamiento

Apache se instala por defecto como un servicio denominado Apache2.2 , de modo que podemos comenzar a operar directamente con él, aunque luego veremos alguna configuración básica.

Apache se ejecuta y controla en modo dos, por lo que abriremos una sesion MSDOS y nos movemos al directorio de apache, y escribimos *httpd -t*:

```
C:\WAMP\Apache22\bin>httpd -t
Syntax OK
```

Donde la primera línea es la entrada que tecleamos nosotros (httpd -t) y la segunda la respuesta del servidor (syntax OK).

Bueno, y esto esta muy bien, pero no nos dice nada. Hagamos para asegurarnos de que todo está bien, una prueba con el navegador: iniciamos el servidor. Hay distintas formas de hacerlo, pero ya que estamos en una sesión DOS, escribimos

```
C:\Archivos de programa\Apache Group\Apache>apache –k start
```

En nuestro caso esto último no es necesario pues el servidor ya está funcionando como servicio. Si todo ha sido correcto nuestro servidor Apache está ejecutandose !!. Abrimos nuestro navegador favorito, y en la dirección escribimos:

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

y deberemos visualizar una página que nos informa que la instalación funciona (mostrará el mensaje ***It Works!***).

¿Que cual es esa dirección que hemos dado? es el nombre por defecto de nuestro ordenador, lo que nos permite usar el servidor para testear páginas incluso desconectados de internet.

Configurando el servidor

Apache crea al instalarse un directorio llamado *conf* (ruta completa *C:\Archivos de programa\Apache Group\Apache\conf* en una instalación por defecto y *C:\WAMP\Apache22\conf* en nuestro curso), dentro del cual están los archivos de

configuración del servidor, archivos que tendremos que modificar para adecuar a nuestro sistema.

El archivo que nos interesa es **httpd.conf** (que encontramos en el directorio conf). Apache no sobreescribe los archivos de instalación, si encuentra alguno. Por ejemplo, si ya existe un *httpd.conf*, no se borra, y mantiene la versión existente de instalaciones anteriores. Dentro de la carpeta conf hay una carpeta denominada *original* que tiene una copia de los ficheros de configuración en su estado inicial por si fuese necesario volver a usarlos.

Se trata de un archivo de texto, que podemos manejar con cualquier editor capaz de guardar texto sin formato (notepad, por ejemplo). Veremos un montón de lineas. Todas las que empiezan por el carácter # son comentarios orientadores de las distintas opciones, o incluso opciones deshabilitadas (se activan "descomentándolas"). Las directivas más importantes son:

ServerAdmin you@your.adress

La dirección de correo del administrador. Esta dirección es la que aparecerá en páginas generadas por el servidor (por ejemplo si indica algun error) y permite a los usuarios informarte de problemas. Escribimos nuestra dirección, por ejemplo:

```
ServerAdmin webmaster@misitio.com
```

#ServerName new.host.name

El nombre del servidor; útil principalmente para redireccionamientos de url; advierte que por defecto viene comentado, luego no es esencial para el funcionamiento (si no hay un nombre de servidor especificado, apache toma la dirección IP de nuestro ordenador). Podemos escribir el nombre de nuestro dominio (www.midominio.com) o nuestra IP, o localhost:

```
ServerName localhost
```

DocumentRoot "C:/WAMP/Apache22/htdocs"

La carpeta raíz de los archivos del sitio web. Esta carpeta es donde se deben colocar las páginas y archivos que queremos que sirva nuestro servidor web. Cambiamos la ubicación de dicha carpeta, por ejemplo:

```
DocumentRoot "C:/WAMP/www"
```

Habrás observado que en determinados sitios web, para acceder al contenido, basta con teclear el nombre del dominio (por ejemplo http://www.google.es nos sirve la pagina principal del buscador), mientras que en otros casos hemos de teclear asimismo el nombre de la página (por ejemplo http://www.mipagina/index.html).

Tambien habrás observado que en ocasiones al teclear solo el nombre del dominio, sin la página, obtenemos un listado del directorio de archivos disponibles, o sencillamente obtienes un mensaje de error.

Todos estos comportamientos obedecen a concretas configuraciones de servidor, que ahora vamos a ver:

Indexes

Si incluimos esta opción, todo aquel que teclee solo nuestro nombre de dominio obtendrá un listado de los archivos disponibles, en lugar de la página principal. Por defecto Apache establece la opcion Indexes para el directorio htdocs, que como sabemos es el raiz de nuestro servidor:

```
Options indexes FollowSymLinks MultiViews
```

Si no queremos que salga este listado, sencillamente borramos la palabra Indexes. Ahora, quien escriba nuestro nombre de dominio, por ejemplo http://www.irvnet.com/ no obtendrá el listado de archivos, obtendrá un mensaje de error.

DirectoryIndex

Con esta opción evitamos ese error, indicando al servidor la página que ha de servir si el usuario no escribe ninguna. Por defecto es la siguiente:

```
DirectoryIndex index.html
```

y nosotros podemos añadir varias separandolas con un espacio. La preferencia la determina el orden de izquierda a derecha:

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

Manejando el servidor

Como hemos dicho, Apache se ejecuta en modo consola, por lo que hemos de teclear las instrucciones siguientes para controlar su funcionamiento si no se ha configurado como un servicio.

Los comandos básicos son:

- apache -s *inicia el servidor*
- apache -k start *inicia el servidor*
- apache -k shutdown *apaga el servidor*
- apache -k restart *reinicia el servidor*
- apahce –k install *instala el servidor como un servicio*

Cuando no se ejecuta como servicio el principal inconveniente es el de dejar abierta la ventana de DOS donde se está ejecutando Apache, y la necesidad de abrir una

sesion de DOS nueva para reiniciar o cerrar el servidor. En un momento dado, es fácil que la barra de tareas de tu pantalla quede llena de ventanas de aplicaciones.

Se pueden manejar el servidor con un pequeño icono situado en el systray denominado Apache Monitor, desde el cual ponerlo en marcha, apagarlo o reiniciarlo, y lo que es más importante, haciendo desaparecer las ventanas DOS del escritorio.

Con esto ya hemos cubierto los pasos básicos y tenemos el servidor operativo. No olvides cambiar la página de bienvenida de apache en *C:\Archivos de programa\Apache Group\Apache\htdocs* (*C:\WAMP\Apache22\htdocs* en nuestro curso) por tu propio contenido.

1.5. Instalación de PHP

Obteniendo los archivos

Las diferentes versiones de PHP se suceden sin parar. La que nosotros usamos aqui es la última de Mayo de 2008, PHP 5.2.6, que obtienes en el archivo *php-5.2.6-Win32.zip*, de aprox. 9,29 megas, directamente de php.net.

Instalando PHP

Con un programa descompresor, como winzip, ejecutamos el archivo obtenido, descomprimiendo su contenido a un nuevo directorio que podemos llamar, por ejemplo, *C:\WAMP\PHP526*.

Accedemos a ese directorio y hacemos dos operaciones:

- Copiar el archivo *php5ts.dll* desde el directorio de instalación de PHP al directorio de windows (c:\windows o c:\winnt o c:\win2000 o c:\winxp).
- Renombramos el archivo *php.ini-recommended* a *php.ini*.

Preparando Apache para PHP5

Podemos instalar PHP como módulo de Apache o como CGI. La instalación como CGI se considera más estable y más segura, aunque como módulo se supone más rápida. Veremos en este manual las dos formas:

PHP como CGI/FastCGI

Recuperamos y editamos el archivo ***httpd.conf***, que por defecto se halla en *C:\Archivos de programa\Apache Group\Apache\conf* y en nuestro curso en *C:\WAMP\Apache22\conf*

Dentro del apartado "**IfModule alias_module**", vemos que apache se ocupa de los ScriptAlias. Introducimos la siguiente línea nueva:

```
ScriptAlias /php/ "C:/WAMP/PHP526"
```

A continuacion buscamos otra sección que comienza diciendo "**IfModule mime_module"** y escribimos:

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

Y una última modificación justo a continuación.

```
Action application/x-httpd-php /php/php-cgi.exe
```

Dado que Apache debe poder acceder al directorio donde se haya instalado PHP, debemos dotar, en el archivo de configuración de Apache, a dicho directorio de los permisos pertinentes. Para ello:

```
<Directory "C:/WAMP/PHP526">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

Partimos de que has instalado php en c:\WAMP\PHP526; si no fuera asi, sustituye la ruta por la tuya propia (advierte que en el archivo de configuracion de Apache las barras son siempre invertidas /)

Probando la instalación

Abrimos nuestro editor de texto y escribimos:

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

guardamos este archivo como *prueba.php* dentro de *C:\Archivos de programa\Apache Group\Apache\htdocs* (con una configuración por defecto) o en nuestro caso en *C:\WAMP\www*, que como sabemos es donde apache guarda los documentos a servir.

Arrancamos Apache, y abrimos el navegador. Escribimos la dirección: *http://localhost/prueba.php*. Si todo funciona correctamente se abrirá una página con información de los parámetros de php. Si el navegador abre la dirección, pero en lugar de visualizar la página, ofrece guardarla en disco, revisa de nuevo que hayas modificado correctamente el archivo *httpd.conf*.

PHP como módulo apache

La instalación varía ligeramente. Los pasos son los siguientes:

- movemos la libreria *php5ts.dll* desde el directorio de instalación de PHP al directorio de windows. (Este paso ya se hizo en la instalación como CGI)
- en lugar de las líneas que habiamos añadido a nuestro archivo *httpd.conf* de Apache en la instalación como CGI (que eliminamos pues ya no son necesarias, añadimos las siguientes:

Dentro del apartado "**Dynamic Shared Object (DSO) Support**", añadimos el módulo de PHP5 para Apache 2.2, poniendo especial cuidado en poner la ruta a la DLL correctamente.

```
LoadModule php5_module "c:/wamp/php526/php5apache2_2.dll
```

A continuacion buscamos otra sección que comienza diciendo "**IfModule mime_module"** y escribimos:

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

Y por último para que se carge el fichero de configuración de PHP, al final del archivo *httpd.conf* ponemos su ruta:

```
PHPIniDir "C:/wamp/php526"
```

Para verificar que todo esta correcto, ejecutamos de nuevo nuestro archivo de prueba (phpinfo) y veremos al comienzo el nuevo dato: Server API Apache 2.0 Handler

Añadiendo módulos a php

La distribución de php para windows viene con alguna de las extensiones necesarias ya incorporadas, por lo que normalmente no necesitaremos añadir ninguna extensión más. Pero en caso de necesitarlo, veremos como es el proceso para activarlas.

Vamos a poner como ejemplo la extensión **gd**, que no viene instalada por defecto, y es necesaria para que php pueda generar gráficos "al vuelo":

- Editamos nuestro php.ini, y buscamos en la sección de extensiones (**Dynamic Extensions**). Veremos que hay una lista de las disponibles, todas ellas comentadas (línea comenzando por ;). Buscamos la que nos interesa: *;extension=php_gd.dll*, eliminamos el punto y coma inicial, guardamos el archivo.

```
;extension=php_gd.dll,
```

- Copiamos la libreria *php_gd.dll* desde el directorio donde se encuentre (en la instalación por defecto, se encuentra en el directorio *ext* dentro de la instalación de PHP) al directorio que hayamos especificado en la directiva ***extension_dir*** que en nuestro php.ini). Otra opción muy recomendable es apuntar dicha directiva (*extensión_dir*) a la carpeta donde vienen instaladas las extensiones por defecto:

```
extension_dir = "c:/wamp/php526/ext/"
```

- Si la instalación de PHP es como módulo de Apache es necesario reiniciar el servidor web para que los cambios en el archivo *php.ini* tengan efecto. Si la instalación es como CGI no es necesario.

¿Que es el archivo php.ini?

El archivo php.ini es el que contiene nuestra configuración de PHP, con el que podemos controlar muchos aspectos de su funcionamiento. A continuación se explica

para que sirve cada una de sus instrucciones y cual es la mejor forma de configurarlo. La explicación sigue el mismo orden interior de php.ini, aunque puede que haya ligeras diferencias con tu copia, debidas a pequeños cambios entre versiones. La configuración aqui contemplada es la que corresponde a las versiones php 5.2.x.

Este archivo sirve para indicar una serie de valores que determinan el comportamiento del intérprete PHP. Lo encontramos dentro de la distribución php en el directorio raiz bajo el nombre php.ini-recommended o php.ini-dist. Se trata de un archivo de texto, que podemos abrir con cualquier editor que trabaje con texto simple (*.txt).

Lo primero que debemos hacer es en efecto editar una de las dos versiones disponibles, configurarla de acuerdo a nuestras necesidades, y guardarla con el nombre php.ini. ¿Cual escoger? las dos son el mismo archivo, con distintos valores por defecto.

Dentro de este archivo, todo lo que comienza con un punto y coma es un comentario, y es ignorado. El texto marcado con corchetes, como *[PHP]* indica una cabecera de sección.

Las instrucciones se llaman directivas, y estan formadas por una pareja compuesta por la clave y su valor, por ejemplo: *asp_tags = Off*. Y ten cuidado, porque diferencia entre mayusculas y minusculas. No es lo mismo asp_tags que Asp_tags. También verás que algunas directivas comienzan con *;* lo que quiere decir que estan comentadas ya que no son necesarias por defecto. Debes desactivarlas sin necesitas esa funcionalidad.

Otro dato más a tener en cuenta. En windows las rutas o paths se escriben con la barra invertida (c:\windows) mientras que unix utiliza la barra (/usr/local/bin/...). En php.ini deberas indicar algunas rutas. Los formatos admisibles son:

- C:\directorio\directorio
- \directorio\directorio
- /directorio/directorio/

Si no especificas la letra de la unidad, se presupone que es la unidad actual (esto es, donde tengas php.exe).

¿Como trabaja el archivo php.ini?

Antes que nada aclarar que el proceso de instalación de PHP en tu ordenador NO crea el archivo php.ini. Una vez instalado PHP debes escoger uno de los archivos proporcionados como ejemplo y renombrarlos a php.ini

Si tenemos PHP como módulo del servidor, el archivo php.ini se lee cada vez que se reinicia. Por lo tanto tienes que reiniciar para que actualice los cambios. Si PHP está instalado como CGI se leerá el php.ini en cada llamada a PHP. En ambos casos, lo primero a tener en cuenta será, pues, donde archivar php.ini. El servidor lo buscará sucesivamente -y por este orden- en el propio directorio php (c:/wamp/php526 si usas la instalacion por defecto). Si no lo encuentra alli lo buscará en la ruta definida como variable de entorno y finalmente en el directorio de sistema (c:/windows)

Lo aconsejado es mover php.ini a tu directorio de sistema (c:\windows habitualmente). Cuida no dejar ninguna version antigua de php.ini en el directorio php, porque podría ser leida con preferencia a la que hayas movido a /windows/. Y *recuerda* que para que cualquier cambio que realices en el php.ini surta efecto, debes reiniciar tu servidor.

Los problemas mas comunes que encontrarás con PHP pasan casi siempre por una incorrecta configuración de php.ini, y en muchos casos, por tener el archivo mal ubicado o duplicado, leyendose un archivo distinto del que tu estas configurando. Si haces un cambio en php.ini y este no se refleja en el funcionamiento de PHP, comprueba la sintaxis que has usado; que has reiniciado el servidor correctamente y que este lee el php.ini deseado. Controla siempre tus copias de php.ini

Es altamente recomendable que tengas preparada una pagina con la función phpinfo() para ver como queda la configuración de tu php:

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

Guarda esta página como info.php o como se te ocurra, y tenla a mano para comprobar la configuración en cuanto tengas tu php listo.

Los dos php.ini

En la carpeta PHP verás que hay dos archivos php.ini: uno php.ini-recommended y otro php.ini-dist. Los dos tienen las mismas directivas, pero configuradas de distinta forma. La version recomendada es mas exigente en cuestiones de seguridad (esencialmente la directiva registrar globales esta off y mostrar errores tambien off) mientras que dist, aunque menos segura, posiblemente permitirá funcionar a la mayoria de los scripts que puedas bajarte de internet, sin necesidad de adaptaciones.

Las directivas

Veremos a continuación cada una de las directivas y su significado, siguiendo el orden que podriamos ver en nuestro php.ini. Muchas directivas vienen con valores por defecto, o sin valor determinado, o comentadas (inactivas). Una buena política es dejarlas como estan, salvo que sepas exactamente que estas haciendo.

Los valores que indicamos en esta página son indicativos. Lo que pretendemos es explicar el valor de cada directiva (al menos las que conocemos), no proponer un php.ini modélico.

Opciones de lenguaje

En esta primera sección encontramos algunas instrucciones generales sobre el funcionamiento de PHP:

- ***engine = On*** **-** activa la interpretación de scripts php (si php está cargado como módulo de apache). Esta directiva, en unión de httpd.conf, permite habilitar o deshabilitar php en directorios determinados.

- ***zend.ze1_compatibility_mode = Off*** -Habilita el modo de compatibilidad con el Motor Zend 1 (PHP 4). Afecta el modo de trabajar con los objetos.

- ***short_open_tag = Off*** - Permite usar en tus scripts etiquetas php abreviadas <? ... ?>, y el atajo para imprimir variables <%= $valor %>. Si el valor es off, deberas usar la forma <?php ... ?> o <script>. Se recomienda ponerlo a off para mayor portabilidad del código

- ***asp_tags = Off*** - Permite usar etiquetas estilo asp <% ... %>. Deshabilitado por defecto

- ***precision = 14*** - número máximo de decimales visualizados

- ***y2k_compliance = On*** - Forzar compatibilidad con el año 2000.

- ***output_buffering = 4096*** - permite enviar cabeceras http (cookies por ejemplo) desde puntos distintos al inicio del script. Además de valores on | off puedes fijar aqui el tamaño maximo (en bytes) de las lineas http permitidas, por ejemplo: output_buffering = 4096

  Puedes deshabilitar esta función con carácter general aqui, y habilitarla en partes concretas de tus scripts utilizando las funciones de buffer correspondientes (por ejemplo *ob_start()*).

  Cuando output buffering esta activado, PHP no lanza las cabeceras HTTP al inicio de la ejecución del script, sino que las almacena temporalmente en un buffer de memoria, lo que te permitirá modificar o añadir instrucciones HTTP durante la ejecución del script, que se enviarán solo cuando este finalice.

  Esta posibilidad esta penalizada por una disminución del rendimiento.

- ***output_handler =*** - Con esta directiva puedes redirigir toda la salida de tus scripts a una función PHP. Es preferible no habilitar esta opción y establecerla si es preciso en cada uno de tus scripts.

- ***zlib.output_compression = Off*** - habilita la libreria zlib de forma que los datos de salida del script se envian comprimidos. Puedes indicar valores off|on o precisar el tamaño del buffer (por defecto es de 4 KB).

- ***;zlib.output_handler =*** - Si tienes activada la opción anterior, no puedes usar la directiva output_handler; con similar funcionalidad tienes zlib.output_handler.

- ***implicit_flush = Off*** - Intenta enviar al cliente el contenido de la memoria intermedia de salida. O dicho coloquialmente, "envia lo que tengas hasta ahora, en lugar de esperar a completarlo". Equivale a llamar la función *flush()* despues de cada llamada echo o print y cada segmento html. Es desaconsejable su activación, siendo preferido usar la función *flush()* cuando sea necesario.

- ***unserialize_callback_func*= -** relacionado con las funciones *serialize()*.

- ***allow_call_time_pass_reference* = *Off* -** Uno más de los cambios en PHP ... tradicionalmente podias construir una función y al usarla, decidir si pasabas o no el valor de una variable por referencia (&$var). Ahora esto es desaconsejado y se recomienda especificar que los valores serán pasados por referencia en la propia declaración de la funcion (function blah (&$var))

- ***safe_mode* = *Off*** - Para activar el modo seguro de PHP.

  Si usas PHP como CGI, "debes" activar safe_mode y especificar el valor de safe_mode_exec_dir, con lo cual aseguras que el usuario solo pueda acceder a la información existente en las carpetas especificadas.

- ***safe_mode_gid* = *Off*** - Por defecto, con safe_mode On PHP hace un chequeo UID del fichero al abrirlo. Con esta directiva puedes especificar en su lugar un chequeo GID

- ***safe_mode_include_dir* = -** Los archivos que esten en este directorio podrán ser utilizados con include/require en safe_mode On sin necesidad de chequeos UID/GID

- ***safe_mode_exec_dir* = -** Si el PHP se utiliza en modo seguro, la función *system()* y el resto de funciones que ejecutan programas del sistema solo actuaran sobre archivos ejecutables que esten en el directorio indicado.

- ***safe_mode_allowed_env_vars* = *PHP_*** - Puedes proporcionar aqui una serie de prefijos (separados por ;). Si indicas estos prefijos, en safe_mode los usuarios solo podrán alterar variables de entorno cuyo nombre comience con ese prefijo. Si esta directiva esta vacia, en safe_mode podrán modificarse todas las variables de entorno.

- ***safe_mode_protected_env_vars* = *LD_LIBRARY_PATH* -** una lista de variables de entorno (separadas por ;) que no pueden variarse via *putenv()* incluso aunque safe_mode_allowed_env_vars lo permita

- ***open_basedir* = -** Limita los archivos que se pueden abrir por PHP al árbol de directorios especificado.

  Cuando un script intenta abrir un archivo con, por ejemplo, *fopen()*, se comprueba su localización. Si el fichero está fuera del árbol de directorios especificado, PHP se negará a abrirlo. Todos los enlaces simbólicos son resueltos, de modo que no es posible evitar esta limitación usando uno de ellos.

  El valor especial indica que el directorio base será aquel en el que reside el script.

  Bajo Windows, los directorios se separan mediante punto y coma. En el resto de sistemas, con dos puntos ":". Como módulo de Apache, los senderos para open_basedir de los directorios padre se heredan ahora automáticamente.

El valor por defecto es permitir abrir todos los archivos.

Esta directiva es independiente de Safe Mode.

- ***disable_functions*** **=** - Con esta directiva puedes inhabilitar con carácter general determinadas funciones PHP. Basta con incluirlas separadas por punto y coma (";"). Al igual que la anterior, es independiente de Safe Mode.

- ***disable_classes*** **=** - Con esta directiva puedes inhabilitar con carácter general determinadas clases. Basta con incluirlas separadas por punto y coma (";"). Al igual que la anterior, es independiente de Safe Mode.

- ***highlight****...* - permite especificar los colores a utilizar por el coloreador de sintaxis interno de PHP

- ***ignore_user_abort = on*** **-** permite al script seguir ejecutandose aun cuando se haya detectado que el usuario ha abortado la petición.

- ***realpath_cache_size=16k*** - Determina el tamaño del caché realpath a ser usado por PHP. Este valor debe ser incrementado en sistemas en los que PHP abre muchos archivos, para reflejar la cantidad de operaciones de archivo realizadas.

- ***realpath_cache_ttl=120*** - Duración de tiempo (en segundos) para mantener en caché la información de realpath para un archivo o directorio dado. En sistemas en los que raramente cambian los archivos, considere incrementar este valor.

- ***expose_php = On*** **-** Permite controlar si PHP debe o no revelar su presencia en el servidor, por ejemplo incluyendose en las cabeceras http del servidor.

Limites al empleo de recursos

- ***max_execution_time = 30*** - Fija el tiempo máximo en segundos que se le permite usar a un script antes de ser finalizado por el intérprete. Así se evita que scripts mal escritos puedan bloquear el servidor.

- ***max_input_time = 60*** - Tiempo máximo en segundos que el script puede invertir en analizar datos recibidos

- ***;max_input_nesting_level = 64*** - Establece el mayor nivel de anidamiento de las variables de entrada.

- ***memory_limit = 128M*** - Fija el tamaño máximo de memoria en bytes que se permite reclamar a un script. Así se evita que script mal escritos se coman toda la memoria disponible de un servidor.

Gestion y archivo de errores

- ***error_reporting = E_ALL*** - Fija el nivel (detalle) con el que PHP te informa de errores. Esta directiva vuelca el informe de errores en la pantalla, y su uso está desaconsejado en páginas en producción, ya que el error puede revelar información sensible. Lo recomendado es permitir mostrar errores, con el máximo detalle posible, mientras desarrollas el script PHP; y cuando está terminado y en

producción, deshabilitar el mostrado de errores en pantalla y activar en su lugar el archivo de errores.

Como cada nivel de informe de error está representado por un número, puedes designar el nivel deseado sumando valores:

1 errores normales
2 avisos normales
4 errores del parser (error de sintaxis)
8 avisos de estilo no críticos

El valor por defecto para esta directiva es 7 (se muestran los errores normales, avisos normales y errores de parser).

Tambien puedes designar el nivel de error nominativamente:

Algunas combinaciones son:
error_reporting = E_ALL & ~E_NOTICE muestra todos los errores criticos, excluyendo advertencias que pueden indicar mal funcionamiento del código pero no impiden la ejecución del intérprete.

error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR muestra solo errores.

error_reporting = E_ALL muestra todos los errores y advertencias.

- ***display_errors = Off*** **-** determina si los errores se visualizan en pantalla como parte de la salida en HTML o no.

  Como queda dicho, es desaconsejado mostrar errores en pantalla en páginas visibles al público.

- ***display_startup_errors = Off*** - Incluso con display_errors on, por defecto PHP no muestra los errores que detecta en la secuencia de encendido. Con esta directiva puedes mostrar estos errores. Desaconsejado activarla.

- ***log_errors = On*** - Guarda los mensajes de error en un archivo. Normalmente el registro del servidor. Esta opción, por tanto, es específica del mismo.

- ***log_errors_max_len = 1024*** - Especifica el tamaño del archivo error_log. Si tiene un valor 0 significa que no hay restricción de tamaño

- ***ignore_repeated_errors = Off*** - Si está activado, no archiva mensajes repetidos. No se consideran mensajes repetidos aquellos que no provienen de la misma linea.

- ***ignore_repeated_source = Off*** - Si está activado, considera repetidos los mensajes de error iguales, aunque provengan de distinta linea / script

- ***report_memleaks = On*** - Mostrar o no. memory leak se refiere a cuando (por error) el script no libera la memoria usada cuando ya no la necesita, y en consecuencia usa cada vez mas hasta llegar a agotarla.

- ***track_errors = Off*** - Si lo activamos, tendremos el último mensaje de error/advertencia almacenado en la variable $php_errormsg

- ***html_errors = Off*** - Si activo, incluye etiquetas HTML en los mensajes de error.

- ***docref_root = /phpmanual/*** y ***docref_ext = .html*** - Si tienes html_errors activado, PHP automaticamente incluye enlaces en el mensaje de error que te dirigen a la página del manual que explica la función implicada. Puedes bajarte una copia del manual y indicar su ubicación (y extensión del archivo) usando estas directivas.

- ***error_prepend_string = ""*** - Cadena a añadir antes de cada mensaje de error.

- ***error_append_string = ""*** - cadena a añadir despues del mensaje de error.

- ***;error_log = filename*** - Nombre del fichero para registrar los errores de un script. Si se utiliza el valor especial syslog, los errores se envían al registro de errores del sistema. Como verás, esta comentado (inhabilitado) por defecto.

Gestion de datos

- ***track_vars*** - Esta directiva crea arrays $HTTP_GET_VARS, $HTTP_POST_VARS y $HTTP_COOKIE_VARS con los datos introducidos con los métodos GET, POST y con cookies. Desde PHP 4.0.3 está siempre activada.

- ***;arg_separator.output = "&"*** - El carácter que se empleará en las urls generadas por PHP para separar argumentos (valores pasados via url). & es el separador por defecto.

- ***;arg_separator.input = ";&"*** - separadores que usará PHP cuando analice una url suministrada para almacenarla en variables

- ***variables_order = "GPCS"*** - Esta directiva fija el orden (precedencia) en que PHP registrará y interpretará las variables de entorno (de izquierda a derecha en el orden indicado). Los valores posibles se toman con las iniciales del método usado para asignar el valor a la variable: Get, Post, Cookie, Enviroment y Server. Fijando por ejempo el valor a "GP", hará que el PHP ignore por completo las cookies y que sobreescriba las variables recibidas por GET con las que tengan el mismo nombre y vengan por POST.

  En php.ini encontrarás una directiva semejante en desuso (no recomendada) que es gpc_order

- ***register_globals = Off*** - Permite registrar autómaticamente (o no) las variables EGPCS como globales. Por razones de seguridad se recomienda desactivar el registro.

- ***register_long_arrays = Off*** - Permite registrar autómaticamente (o no) las variables EGPCS como en su formato largo (HTTP_ _VARS). Por razones de rendimientose recomienda desactivar el registro.

- ***register_argc_argv = Off*** - Esta directiva instruye a PHP si debe declarar las variables argv&argc (arrays predefinidos que almacenan los parámetros pasados (argv) y su numero (argc).

- ***auto_globals_jit = On*** - Cuando es habilitado el parámetro, las variables SERVER y ENV son creadas al usarse por primera vez ("Just In Time", o justo a tiempo) en lugar de hacerlo cuando el script inicia. Si estas variables no son usadas en el interior del script, esta directiva resultará en una ganancia en rendimiento.

- ***post_max_size = 8M*** - Tamaño máximo de los datos que PHP aceptará por el método POST

Magic quotes

- ***magic_quotes_gpc = Off*** - Fija el estado magic_quotes para operaciones GPC (Get/Post/Cookie). Si magic_quotes vale on, todas las ' (comilla sencilla), " (comilla doble), \ (barra invertida) y los NUL son automáticamente marcados con una barra invertida. Si además magic_quotes_sybase vale on, la comilla sencilla es marcada con otra comilla sencilla en lugar de la barra invertida.

- ***magic_quotes_runtime = Off*** - Si se habilita magic_quotes_runtime, muchas de las funciones que devuelven datos de algún tipo de fuente externa incluyendo bases de datos y archivos de texto devolverán las comillas marcadas con una barra invertida. Si también está activo magic_quotes_sybase, la comilla simple es marcada con una comilla simple en lugar de la barra invertida.

- ***magic_quotes_sybase = Off*** - Si magic_quotes_sybase está a on, la comilla simple es marcada con una comilla simple en lugar de la barra invertida cuando están habilitados magic_quotes_gpc o magic_quotes_runtime.

Más directivas de Gestion de datos

- ***auto_prepend_file =*** y ***auto_append_file =*** permiten indicar la ruta y nombre de un archivo que se añadirán antes o despues (respectivamente) de todos los archivos php que se ejecuten.
  El valor especial none desconecta la adición automática de archivos.
  Si el script es terminado con exit(), no tendrá lugar la adición automática señalada con auto_append_file.
  Los archivos indicados con estas directivas se incluirán como si fuesen llamados mediante la función include(), así que se utiliza include_path.

- ***;default_charset = "iso-8859-1"*** - Por defecto, el código de caracteres indicado por PHP en la cabecera de salida.

- ***default_mimetype = "text/html"*** -Por defecto, el tipo mime de salida de datos. Cada MIMETYPE define el formato de los datos (por ejemplo, texto/hmtl, jpg, gif ....)

- ***;always_populate_raw_post_data = On*** - PHP crea la variable $HTTP_RAW_POST_DATA cuando recibe datos via POST cuyo tipo MIME no reconoce (almacena los datos en esta variable sin analizarlos). Con esta directiva se

ordena que se cree siempre la variable $HTTP_RAW_POST_DATA, aunque el tipo MIME sea conocido.

Rutas y directorios

- ***include_path = ".;c:\php\includes"*** - Permite especificar una lista de directorios en los que las funciones *require(), include() y fopen_with_path()* buscaran los archivos requeridos. El formato es similar a la variable de entorno de sistema PATH: una lista de directorios separados por dos puntos en UNIX o por punto y coma en Windows. Ejemplo unix seria *include_path=.:/home/httpd/php-lib* y en windows *include_path=".;c:\www\phplib"*.

  El valor por defecto para esta directiva es . (sólo el directorio actual).

- ***doc_root =*** - Indica el "Directorio raiz" donde estan nuestras paginas php en el servidor. Sólo se usa si no está vacío. Si PHP se configura con safe mode, no se interpretaran las páginas php situadas fuera de este directorio. Ojo con los servidores virtuales que apuntan a zonas distintas del servidor.

- ***user_dir =*** - El directorio raiz para los archivos PHP bajo el directorio inicial de un usuario (/~usuario). Normalmente se deja vacio

- ***extension_dir = ./*** - En qué directorio debe buscar el PHP las extensiones dinámicas a cargar. Deberemos modificar este valor para que apunte donde tenemos situadas las dll de las extensiones de PHP.

- ***enable_dl = On*** - Esta directiva sólo es útil en la versión del PHP como módulo del Apache. Puede habilitar o deshabilitar para un servidor virtual o para un directorio la carga dinámica de extensiones de PHP mediante dl().

  La razón principal para desactivar la carga dinámica es la seguridad. Con la carga dinámica es posible ignorar las restricciones para abrir archivos establecidas con open_basedir.

  El valor por defecto es permitir la carga dinámica, excepto cuando se usa safe_mode. En modo seguro, es imposible usar dl().

- ***cgi.force_redirect = 1*** - Por defecto se activa. Es una directiva importante de seguridad que "debes" activar si ejecutas en tu apache PHP como cgi (no es necesaria si tienes PHP como modulo, o si usas como servidor el IIS de microsoft).

- ***;cgi.redirect_status_env = ;*** - En conjunción con cgi.force_redirect y servidores distintos de Apache o iPlanet.

- ***; fastcgi.impersonate = 1;*** - En conjunción con IIS y FastCGI

- ***cgi.rfc2616_headers= 0*** - Le dice a PHP qué tipo de cabeceras usar cuando envíe los códigos de respuesta HTTP. Si su valor es 0, PHP envía una cabecera Status: que es soportada por Apache y otros servidores web.

Subir ficheros

- ***file_uploads = On*** - Permitir o no subir (upload) ficheros via HTTP.

- ***upload_tmp_dir =*** - Carpeta o directorio utilizable para guardar temporalmente archivos subidos por PHP. Si no se especifica, usará el designado por defecto por el servidor. El usuario que esté ejecutando el script debe tener permiso de escritura en ese directorio.

- ***upload_max_filesize = 2M*** - Tamaño máximo de archivos que pueden subirse.

Directivas relacionadas con fopen

- ***allow_url_fopen = On*** - Permite pasar urls (http, ftp) a la función fopen(), en lugar de la ubicacion fisica del archivo

- ***allow_url_include = On*** - Permite pasar urls (http, ftp) a las funciones *incluye(), require(), incluye_once()* y *require_once()*, en lugar de la ubicacion fisica del archivo

- ***;from="john@doe.com"*** - define el email a usar como contraseña para ftp anonimo

- ***;user_agent="PHP"*** - define la "firma" que dejará PHP en el servidor remoto de donde coge los archivos

- ***default_socket_timeout = 60*** - timeout en segundos para la apertura de sockets

- ***; auto_detect_line_endings = Off*** - Si activo, PHP detectara automaticamente el carácter que indica fin de linea (distinto en windows, linux y windows)

Extensiones dinamicas

- ***extension=*** - Qué extensiones dinámicas debe cargar el PHP cuando arranca. Debes elegir el archivo que corresponde a tu sistema operativo: por ejemplo *extension=msql.dll* para windows, *extension=msql.so* para linux.

  Ojo, aqui solo indicamos la extension de los archivos, no su ubicación. Los archivos DEBEN estar en el directorio especificado mas arriba con *extension_dir*.

Configuracion de modulos de PHP

- ***define_syslog_variables = Off*** - Permite definir variables del sistema. Recomendado Off.

- ***;browscap = extra/browscap.ini*** - El archivo browscap.ini es un archivo de texto que contiene información sobre las cadenas de identificación que usa cada navegador. Mediante esta directiva indicas a PHP donde tienes browscap.ini; se usa conjuntamente con la función get_browser().

Directivas de Configuración de Correo

Si usas PHP bajo linux, puedes enviar correo usando tu propio PC con sendmail; con windows no tienes esa posibilidad, por lo que para enviar correos desde un script PHP con la funcion mail() tienes que delegar en tu configuración de correo ordinaria, la que usas por ejemplo con outlook para enviar y recibir correo.

Este seria un ejemplo bajo windows:

- ***SMTP = mailhost@teleline.es*** - Este seria el caso si tu conexion a internet te la proporciona telefonica. Especificamos la direccion del servidor smtp (correo saliente).

- ***sendmail_from= webmaster@misitio.com*** - La dirección del remitente ("De:") para los correos enviados desde PHP bajo Windows.

2. Características del lenguaje

- *HTML y PHP*
- *Comentarios*
- *Instrucciones*

2.1. HTML y PHP

Para interpretar un archivo, PHP simplemente interpreta el texto del archivo hasta que encuentra uno de los carácteres especiales que delimitan el inicio de código PHP. El intérprete ejecuta entonces todo el código que encuentra, hasta que encuentra una etiqueta de fin de código, que le dice al intérprete que siga ignorando el código siguiente. Este mecanismo permite embeber código PHP dentro de HTML: todo lo que está fuera de las etiquetas PHP se deja tal como está, mientras que el resto se interpreta como código.

Hay cuatro conjuntos de etiquetas que pueden ser usadas para denotar bloques de código PHP. De estas cuatro, sólo 2 *(<**?php** ... **?>*** y ***<script language="php">*** ... ***</script>***) están siempre disponibles; el resto pueden ser configuradas en el fichero de php.ini para ser o no aceptadas por el intérprete. Mientras que el formato corto de etiquetas (short-form tags) y el estilo ASP (ASP-style tags) pueden ser convenientes, no son portables como la versión de formato largo de etiquetas. Además, si se pretende embeber código PHP en XML o XHTML, será obligatorio el uso del formato ***<?php*** ... ***?>*** para la compatibilidad con XML.

Las etiquetas soportadas por PHP son:

Formas de escapar de HTML

```
<?php
echo("si quieres servir documentos XHTML o XML, haz  como   aquí<BR>");
?>

<?
echo ("esta es la más simple, una instrucción de procesado   SGML <BR>");
?>
<?= expression ?> Esto es una abreviatura de "<? echo expression ?>"<BR>

<script language="php">
echo ("muchos editores (como FrontPage) no aceptan instrucciones de
procesado<BR>");
</script>

<%
echo ("Opcionalmente, puedes usar las etiquetas ASP<BR>");
%>
<%= $variable; # Esto es una abreviatura de "<% echo . . ." %>
```

El método primero, ***<?php*** ... ***?>***, es el más conveniente, ya que permite el uso de PHP en código XML como XHTML.

El método segundo no siempre está disponible. El formato corto de etiquetas está disponible con la función **short_tags()** (sólo PHP 3), activando el parámetro del fichero de configuración de PHP short_open_tag, o compilando PHP con la opción --enable-short-tags del comando **configure**. Está activa por defecto en php.ini-dist.

El método cuarto sólo está disponible si se han activado las etiquetas ASP en el fichero de configuración: asp_tags.

> ***Nota:*** *El soporte de etiquetas ASP se añadió en la versión 3.0.4.*
> ***Nota:*** *No se debe usar el formato corto de etiquetas cuando se desarrollen aplicaciones o librerías con intención de redistribuirlas, o cuando se desarrolle para servidores que no están bajo nuestro control, porque puede ser que el formato corto de etiquetas no esté soportado en el servidor. Para generar código portable y redistribuíble, asegúrate de no usar el formato corto de etiquetas.*

La etiqueta de fin de bloque incluirá tras ella la siguiente línea si hay alguna presente. Además, la etiqueta de fin de bloque lleva implícito el punto y coma; no necesitas por lo tanto añadir el punto y coma final de la última línea del bloque PHP.

PHP permite estructurar bloques como:

```
<?php
if ($expression) {
?>
    <strong>This is true.</strong>
<?php
} else {
?>
    <strong>This is false.</strong>
<?php
}
?>
```

Este ejemplo realiza lo esperado, ya que cuando PHP encuentra las etiquetas **?>** de fin de bloque, empieza a escribir lo que encuentra tal cual hasta que encuentra otra etiqueta de inicio de bloque. El ejemplo anterior es, por supuesto, inventado. Para escribir bloques grandes de texto generamente es más eficiente separalos del código PHP que enviar todo el texto mediante las funciones **echo()**, **print()** o similares.

2.2. Comentarios

Una vez visto como se introduce código PHP dentro del código HTML, lo siguiente es ver como se puede comentar el código PHP. PHP soporta el estilo de comentarios de 'C', 'C++' y de la interfaz de comandos de Unix. Por ejemplo:

```
<?php
    echo "Esto es una prueba"; // Comentario estilo c++
    /* Comentario multi-linea
       con varias lineas de comentario */
    echo "Otra prueba";
    echo "Prueba final"; # Comentario estilo shell de Unix
?>
```

Los estilos de comentarios de una línea (es decir, // y #) actualmente sólo comentan hasta el final de la línea o el bloque actual de código PHP, lo primero que ocurra.

```
<h1>Esto es un <?php # echo "simple";?> ejemplo.</h1>
<p>La cabecera de arriba dice 'Esto es un ejemplo.'.
```

Hay que tener cuidado con no anidar comentarios de estilo 'C', algo que puede ocurrir al comentar bloques largos de código.

```
<?php
 /*
   echo "Esto es una prueba"; /* Este comentario causa problemas*/
 */
?>
```

Los estilos de comentarios de una linea actualmente sólo comentan hasta el final de la linea o del bloque actual de código PHP, lo primero que ocurra. Esto implica que el código HTML tras *// ?>* será impreso: *?>* sale del modo PHP, retornando al modo HTML, el comentario *//* no le influye.

2.3. Instrucciones

Un fragmento de código PHP va a estar compuesto por una o varias instrucciones. En PHP la separación de instrucciones se hace de la misma manera que en C o Perl - terminando cada instrucción con un punto y coma.

La etiqueta de fin de bloque (*?>*) implica el fin de la instrucción, por lo tanto no es necesario un punto y coma después de la última instrucción. Como se ve en el ejemplo donde los dos fragmentos siguientes son equivalentes:

```
<?php
    echo "Esto es una prueba";
?>

<?php echo "Esto es una preba" ?>
```

Todo script PHP se compone de una serie de sentencias o instrucciones. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma como se ha dicho anteriormente. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia.

Para construir las sentencias o instrucciones se necesitan distintos elementos: variables y constantes, operadores, estructuras de control, funciones, ... A continuación se irán viendo cada uno de estos elementos que permiten construir los scripts de PHP.

3. Variables y Constantes

- *Conceptos Básicos*
- *Variables predefinidas*
- *Ambito de las variables*
- *Existencia y tipo de las variables*
- *Variables variables*
- *Variables externas a PHP*
- *Constantes*

3.1. Conceptos Básicos

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable valido tiene que empezar con una letra o una raya (underscore), seguido de cualquier número de letras, números y rayas. Como expresión regular se podría expresar como: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

> ***Nota:*** *En nuestro caso, una letra es a-z, A-Z, y los caracteres ASCII del 127 al 255 (0x7f-0xff).*

```
<?php
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // imprime "Bob, Joe"

$4site = 'not yet';     // inválido; comienza con un número
$_4site = 'not yet';    // valido; comienza con un subrayado
$täyte = 'mansikka';    // valido; 'ä' es ASCII 228 (Extendido)
?>
```

En PHP, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor íntegro de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después de asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra.

PHP ofrece otra forma de asignar valores a las variables: *asignar por referencia*. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes matrices u objetos.

Para asignar por referencia, simplemente se antepone un signo ampersand "&" al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida 'Mi nombre es Bob' dos veces:

```
<?php
$foo = 'Bob';              // Asigna el valor 'Bob' a $foo
```

```
$bar = &$foo;              // Referencia $foo vía $bar.
$bar = "Mi nombre es $bar";  // Modifica $bar...
echo $foo;                 // $foo también se modifica.
echo $bar;
?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

```
<?php
$foo = 25;
$bar = &$foo;      // Esta es una asignación válida.
$bar = &(24 * 7);  // Inválida; referencia una expresión sin nombre.

function test() {
   return 20;
}

$bar = &test();
?>
```

No es necesario inicializar variables en PHP, sin embargo, es una muy buena práctica. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo - FALSE, cero, una cadena vacía o una matriz vacía. La función **isset()** se puede usar para detectar si una variable ya ha sido inicializada.

3.2. Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. De todas formas, muchas de esas variables no pueden ser completamente documentadas ya que dependen de sobre qué servidor se esté ejecutando, la versión y configuración de dicho servidor, y otros factores. Algunas de estas variables no estarán disponibles cuando se ejecute PHP desde la línea de comandos. Para obtener una lista de estas variables se puede consultar la sección *Variables predefinidas reservadas* de la ayuda de PHP.

A partir de PHP 4.1.0, PHP ofrece un conjunto adicional de matrices predefinidas, conteniendo variables del servidor web, el entorno y entradas del usuario. Estas nuevas matrices son un poco especiales porque son automáticamente globales. Por esta razón, son conocidas a menudo como "autoglobales" ó "superglobales". Las superglobales existentes más usadas se detallan a continuación; sin embargo para una lista de sus contenidos y más información sobre variables dichas predefinidas en PHP, se pueden consultar la sección *Variables predefinidas reservadas* de la ayuda de PHP.

- **$GLOBALS :** Contiene una referencia a cada variable disponible en el espectro de las variables del script. Las claves de esta matriz son los nombres de las variables globales.

```
<?php
function test() {
    $foo = "variable local";

    echo '$foo en el ámbito global: ' . $GLOBALS["foo"] . "\n";
    echo '$foo en el ámbito actual: ' . $foo . "\n";
}
$foo = "Contenido de ejemplo";
test();
?>
```

- **$_SERVER :** Variables definidas por el servidor web o directamente relacionadas con el entorno en donde el script se esta ejecutando. Análoga a la antigua matriz $HTTP_SERVER_VARS (la cual está todavía disponible, aunque no se use).

```
echo $_SERVER['SERVER_NAME'];
```

- **$_GET :** Variables proporcionadas al script por medio de HTTP GET. Análoga a la antigua matriz $HTTP_GET_VARS (la cual está todavía disponible, aunque no se use). Si el usuario ha visitado la siguiente dirección *http://example.com/?nombre=Juan*

```
echo '¡Hola ' .  $_GET["nombre"] . '!';
```

- **$_POST :** Variables proporcionadas al script por medio de HTTP POST. Análoga a la antigua matriz $HTTP_POST_VARS (la cual está todavía disponible, aunque no se use).
- **$_COOKIE :** Variables proporcionadas al script por medio de HTTP cookies. Análoga a la antigua matriz $HTTP_COOKIE_VARS (la cual está todavía disponible, aunque no se use).

```
echo '¡Hola ' .  $_COOKIE["nombre"] . '!';
```

- **$_FILES :** Variables proporcionadas al script por medio de la subida de ficheros via HTTP . Análoga a la antigua matriz $HTTP_POST_FILES (la cual está todavía disponible, aunque no se use).
- **$_ENV :** Variables proporcionadas al script por medio del entorno. Análoga a la antigua matriz $HTTP_ENV_VARS (la cual está todavía disponible, aunque no se use).

```
echo '¡Mi nombre de usuario es ' . $_ENV["USER"] . '!';
```

- **$_REQUEST :** Variables proporcionadas al script por medio de cualquier mecanismo de entrada del usuario ($_GET, $_POST, $_COOKIE) y por lo tanto no es recomendable su uso. La presencia y el orden en que aparecen las variables en esta matriz es definido por la directiva de configuración *variables_order*. Esta matriz no tiene un análogo en versiones anteriores a PHP 4.1.0.
- **$_SESSION :** Variables registradas en la sesión del script. Análoga a la antigua matriz $HTTP_SESSION_VARS (la cual está todavía disponible, aunque no se use).

3.3. Ambito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
<?php
$a = 1;
include "b.inc";
?>
```

Aquí, la variable *$a* será visible dentro del script incluido *b.inc*. De todas formas, dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variable que se use dentro de una función está, por defecto, limitada al ámbito local de la función. Por ejemplo:

```
<?php
$a = 1; /* global scope */

function Test()
{
    echo $a; /* referencia a una variable de ámbito local */
}

Test();
?>
```

Este script no producirá salida, ya que la orden **echo** utiliza una versión local de la variable *$a*, a la que no se ha asignado ningún valor en su ámbito. Puede que se note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobreescritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función mediante el modificador **global**. Veamos un ejemplo:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
}

Sum();
echo $b;
?>
```

El script anterior producirá la salida *"3"*. Al declarar *$a* y *$b* globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando la matriz **$GLOBALS**. El ejemplo anterior se puede reescribir así:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
```

```
Sum();
echo $b;
?>
```

La matriz $**GLOBALS** es una matriz asociativa con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento de la matriz. **$GLOBALS** existe en cualquier ámbito, esto pasa porque **$GLOBALS** es una superglobal. A continuación un ejemplo que demuestra el poder de las superglobales:

```
<?php
function test_global()
{

    // La mayoría de variables predefinidas no son "super" y requieren
    // 'global' para estar disponibles al ámbito local de las funciones.
    global $HTTP_GET_VARS;

    print $HTTP_GET_VARS['name'];

    // Las superglobales están disponibles en cualquier ámbito y no
    // requieren 'global'. Las superglobales están disponibles desde
    // PHP 4.1.0, y ahora HTTP_GET_VARS se considera obsoleta.
    print $_GET['name'];
}
?>
```

Otra característica importante del ámbito de las variables es el modificador **static**. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```
<?php
function Test ()
{
    $a = 0;
    echo $a;
    $a++;
}
?>
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a *$a* el valor *0* y representa un *"0"*. La sentencia *$a++*, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina la variable *$a* desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable *$a* debe declararse como estática:

```
<?php
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Ahora, cada vez que se llame a la función *Test()*, se representará el valor de *$a* y se incrementará.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma

indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta *10*, usando la variable estática *$count* para saber cuándo parar:

```
<?php
function Test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
?>
```

Las variables estáticas pueden ser declaradas como se ha visto en los ejemplos anteriores. Al tratar de asignar valores a estas variables que sean el resultado de expresiones, causará un error de análisis sintáctico.

```
<?php
function foo(){
    static $int = 0;          // correcto
    static $int = 1+2;        // incorrecto  (ya que es una expresión)
    static $int = sqrt(121);  // incorrecto  (es una expresión)

    $int++;
    echo $int;
}
?>
```

3.4. Existencia y tipo de una variable

Hemos visto que en PHP el tratamiento de las variables es muy flexible, a la vez que impreciso. Su tiempo de vida también es breve. En ocasiones, antes de volver a usarla, es necesario conocer si una variable está definida y, si lo está, qué valor tiene. Puede ser que en el transcurso de un programa que llame a varias páginas web la variable haya sido eliminada de la memoria o que haya adquirido el valor nulo. Para conocer estos dos datos, PHP dispone de las dos funciones siguientes:

1. **empty(),** que devuelve el valor *verdadero* si la variable no está definida o lo está, pero tiene un valor nulo. Por ejemplo, si a la variable *$num* le asignamos el valor *10*, las instrucciones

```
if (!empty($num))
    echo "La variable \$num contiene $num.";
else
    echo "La variable \$num no está definida.";
```

genera la página

```
La variable $num contiene 10.
```

En cambio, si no hubiéramos definido la variable, devolvería

```
La variable $num no está definida.
```

2. **isset()**, que devuelve el valor *verdadero* si la variable ha sido inicializada con un valor y *falso* si no lo ha sido. Por ejemplo, si inicializamos la variable *$num* con valor *10*, las instrucciones

```
if (isset($num))
   echo "La variable \$num ha sido inicializada y contiene $num.";
else
   echo "La variable \$num no ha sido inicializada o definida.";
```

genera

```
La variable $num ha sido inicializada y contiene 10.
```

En cambio, si no hubiéramos definido la variable, devolvería

```
La variable $num no ha sido inicializada o definida.
```

Cómo obtener información sobre el tipo de una variable

A veces, también necesitamos saber de qué tipo son las variables. PHP dispone con este fin de un conjunto de funciones que empiezan por **is_** y que devuelven verdadero si el tipo de variable es el que se indique y falso si no lo es. Son las siguientes:

1. ***is_array():*** Devuelve verdadero si el tipo de la variable es una matriz y falso si no lo es.

2. ***is_bool():*** Devuelve verdadero si el tipo de la variable es booleano y falso si no lo es.

3. ***is_float()*** o ***is_double()*** o ***is_real()***: Devuelve verdadero si el tipo de la variable es un número en coma flotante y falso si no lo es.

4. ***is_int()*** o ***is_integer()*** o ***is_long()***: Devuelve verdadero si el tipo de la variable es un número entero y falso si no lo es.

5. ***is_object()***: Devuelve verdadero si el tipo de la variable es un objeto y falso si no lo es.

6. ***is_resource()***: Devuelve verdadero si el tipo de la variable es un recurso y falso si no lo es.

7. ***is_scalar()***: Devuelve verdadero si el tipo de la variable es un escalar (integer, float, string o booleano) y falso si no lo es.

8. ***is_string()***: Devuelve verdadero si el tipo de la variable es una cadena de caracteres y falso si no lo es.

9. ***is_null():***Devuelve verdadero si el tipo de la variable es el valor NULL y falso si no lo es.

10. ***is_numeric():*** Devuelve verdadero si la variable es un número o una cadena numérica o falso si no lo es.

11. ***gettype()***: Esta última función, más directa que las anteriores, devuelve directamente el tipo de la variable que se le pasa como argumento. Los tipos posibles son integer, double, string, array, object y unknown (desconocido).

Si sólo se necesita saber de qué tipo es una variable, esta última función es la más directa y recomendable. En cambio, si se necesita que una variable sea de un determinado tipo para operar con ella, es más directo y recomendable utilizar la correspondiente anterior.

Cómo modificar una variable

PHP dispone también de funciones para realizar estas operaciones:

1. Eliminar una variable de la memoria. Recordamos que todas las variables se eliminan por sí mismas al acabar la interpretación de un programa, por lo cual no es necesario eliminarlas explícitamente. Esta operación sólo es recomendable cuando se comprueba que dentro de un mismo script php se consumen innecesariamente muchos recursos, como cuando hay muchas variables definidas que ya no se usan.

   En este caso, podemos utilizar la función **unset()**, que elimina una variable definida previamente. El nombre de la variable que debe eliminarse se pasa a la función como parámetro. Por ejemplo, si queremos que *$casa* ya no esté definida, podemos escribir *unset($casa).* Si la operación se ha llevado a cabo con éxito, la función devuelve el valor verdadero; de lo contrario, devuelve falso.

```
<?php
// destruir una variable sencilla
unset($foo);

// destruir un elemento de una matriz
unset($bar['quux']);

// destruir más de una variable
unset($foo1, $foo2, $foo3);
?>
```

2. Asignar un tipo concreto a una variable. La función ***settype()*** permite, igualmente, forzar que una variable sea del tipo especificado.

   Por ejemplo, si queremos que la variable *$cosa* sea de tipo *cadena*, debemos escribir: *settype($cosa, "string");* . Como puede observarse, el primer argumento es el nombre de la variable y el segundo (entre comillas dobles) es el tipo de dato que debe tener: boolean, integer, float, string, array, object, null.

```
<?php
$foo = "5bar"; // string
$bar = true;   // boolean

settype($foo, "integer"); // $foo es ahora 5   (integer)
```

```
settype($bar, "string");  // $bar es ahora "1" (string)
?>
```

Separar el contenido de una variable

El contenido de una variable de cadena puede contener letras y números. PHP dispone de diferentes funciones que permiten separar los números del texto y asignarlos a las variables correspondientes de su tipo.

Por ejemplo, si tenemos la cadena *"89.53Pepe es madrileño"*, es posible separar el contenido numérico de este texto: *89.53*. Veamos cómo se hace estudiando cada una de las funciones siguientes:

1. ***floatval():*** Devuelve el número de coma flotante que haya en la frase. Conviene advertir que esto sólo ocurre si el número de coma flotante aparece al principio de la frase. De no se así, se devuelve 0.

2. ***intval():*** Devuelve el número entero que haya en la cadena o 0 si no lo hay.

3. ***strval():*** Devuelve una cadena de caracteres utilizando el contenido de la variable.

Ver el contenido de variables no escalares

Si se desea ver el contenido de una variable no escalar (arrays, objetos), no se puede hacer uso de las funciones **echo()** y **print()**. Para ver el contenido de variables de estos tipos podemos usar las funciones **print_r()** y **var_dump()**.

```
$a = array ('a' => 'manzana', 'b' => 'banano', 'c' => array ('x', 'y', 'z
'));
print_r ($a);
$a = array(1, 2, array("a", "b", "c"));
var_dump ($a);
```

3.5. Variables variables

A veces es conveniente tener nombres de variables variables. Dicho de otro modo, son nombres de variables que se pueden establecer y usar dinámicamente. Una variable normal se establece con una sentencia como:

```
<?php
$a = "hello";
?>
```

Una variable variable toma el valor de una variable y lo trata como el nombre de una variable. En el ejemplo anterior, *hello*, se puede usar como el nombre de una variable utilizando dos signos de dólar. p.ej.

```
<?php
$$a = "world";
?>
```

En este momento se han definido y almacenado dos variables en el árbol de símbolos de PHP: *$a*, que contiene *"hello"*, y *$hello*, que contiene *"world"*. Es más, esta sentencia:

```
<?php
echo "$a ${$a}";
?>
```

produce el mismo resultado que:

```
<?php
echo "$a $hello";
?>
```

p.ej. ambas producen el resultado: *hello world*.

Para usar variables variables con matrices, hay que resolver un problema de ambigüedad. Si se escribe *$$a[1]* el intérprete necesita saber si nos referimos a utilizar *$a[1]* como una variable, o si se pretendía utilizar *$$a* como variable y el índice *[1]* como índice de dicha variable. La sintaxis para resolver esta ambiguedad es: *${$a[1]}* para el primer caso y *${$a}[1]* para el segundo.

Aviso
Hay que tener en cuenta que variables variables no pueden usarse con Matrices superglobales. Esto significa que no se pueden hacer cosas como ${$_GET}.

3.6. Constantes

Una constante es un identificador para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. (Las constantes especiales **__FILE__** y **__LINE__** son una excepción a esto, ya que actualmente no lo son). Una constante es sensible a mayúsculas por defecto. Por convención, los identificadores de constantes suelen declararse en mayúsculas.

El nombre de una constante sigue las mismas reglas que cualquier etiqueta en PHP. Un nombre de constante válido empieza con una letra o un caracter de subrayado, seguido por cualquier número de letras, números, o subrayados. Se podrían expresar mediante la siguiente expresión regular: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*

El alcance de una constante es global, es decir, es posible acceder a ellas sin preocuparse por el ámbito de alcance.

Se puede definir una constante usando la función **define()**. Una vez definida, no puede ser modificada ni eliminada. La sintaxis de **define()** es la siguiente:

```
int define(string nombre, mixed valor [, int noMayusculas])
```

Solo se puede definir como constantes valores escalares (**boolean**, **integer**, **float** y **string**).

Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, no se tiene que especificar el prefijo $. También se puede utilizar la función **constant()**, para obtener el valor de una constante, en el caso de que queramos expresarla de forma dinámica se debe usar la función ***get_defined_constants()*** parar obtener una lista de todas las constantes definidas.

> ***Nota:*** *Las constantes y las variables (globales) se encuentran en un espacio de nombres distinto. Esto implica que por ejemplo* ***TRUE*** *y $TRUE son diferentes.*

Si se usa una constante todavía no definida, PHP asume que se está refiriendo al nombre de la constante en si. Se lanzará un aviso si esto sucede. Se debe usar la función **defined()** para comprobar la existencia de dicha constante.

Estas son las diferencias entre constantes y variables:

- Las constantes no van precedidas por un símbolo de dolar ($)
- Las contantes solo pueden ser definidas usando la función **define()** , nunca por simple asignación
- Las constantes pueden ser definidas y se accede a ellas sin tener en cuenta las reglas de alcance del ámbito.
- Las constantes no pueden ser redefinidas o eliminadas después de establecerse; y
- Las constantes solo puede albergar valores escalares

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // Imprime "Hello world."
echo Constant; // Imprime "Constant" y imprime una alerta.
echo MAXSIZE;
echo constant("MAXSIZE"); // Imprime lo mismo que la anterior
?>
```

4. Tipos de Datos

- *Booleanos*
- *Enteros*
- *Números en punto flotante*
- *Cadenas*
- *Arrays*
- *Objetos*
- *Recursos*
- *NULL*
- *Forzado de tipos*

PHP soporta ocho tipos primitivos:

- booleanos
- enteros
- números en punto flotante
- cadenas
- matrices
- objetos
- recursos (resource)
- NULL

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución dependiendo del contexto en el que se utilice esa variable.

Si desea conocer el tipo y valor de una expresión, se puede usar la función **var_dump()**. Para obtener una representación legible para humanos del tipo de una variable para propósitos de depuración, se puede usar la función **gettype()**. Para comprobar si una variable es de un cierto tiepo, *no* se debe usar **gettype()**, si no las funciones **is_tipo**.

```
<?php
$un_bool = TRUE;   // un valor booleano
$un_str  = "foo";  // una cadena
$un_str2 = 'foo';  // una cadena
$un_int  = 12;     // un entero

echo gettype($un_bool); // imprime: boolean
echo gettype($un_str);  // imprime: string

// Si este valor es un entero, incrementarlo en cuatro
if (is_int($un_int)) {
    $un_int += 4;
}

// Si $bool es una cadena, imprimirla
// (no imprime nada)
if (is_string($un_bool)) {
    echo "Cadena: $un_bool";
}
?>
```

Si se quisiese obligar a que una variable se convierta a un tipo concreto, se podría forzar la variable o usar la función **settype()** para ello.

Nótese que una variable se puede comportar de formas diferentes en ciertas situaciones, dependiendo de qué tipo sea en ese momento.

4.1. Booleanos

Este es el tipo más simple. Un booleano expresa un valor de verdad. Puede ser **TRUE** o **FALSE**. Para especificar un literal booleano, use alguna de las palabras clave **TRUE** o **FALSE**. Ambas son insensibles a mayúsculas y minúsculas.

```
$foo = True; // asigna el valor TRUE a $foo
```

4.2. Enteros

Los enteros se pueden especificar usando una de las siguientes sintaxis:

```
$a = 1234; # número decimal
$a = -123; # un número negativo
$a = 0123; # número octal (equivalente al 83 decimal)
$a = 0x12; # número hexadecimal (equivalente al 18 decimal)
```

4.3. Números en punto flotante

Los números en punto flotante (double) se pueden especificar utilizando cualquiera de las siguientes sintaxis:

```
$a = 1.234;
$a = 1.2e3;
$a = 7E-10;
```

4.4. Cadenas

Las cadenas de caracteres se pueden especificar usando uno de dos tipos de delimitadores.

Si la cadena está encerrada entre dobles comillas ("), las variables que estén dentro de la cadena serán expandidas (sujetas a ciertas limitaciones de interpretación). Como en C y en Perl, el carácter de barra invertida ("\") se puede usar para especificar caracteres especiales:

secuencia	significado
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\\	Barra invertida
\$	Signo del dólar
\"	Comillas dobles
\[0-7]{1,3}	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal
\x[0-9A-Fa-f]{1,2}	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal

Se puede proteger cualquier otro carácter, pero se producirá una advertencia en el nivel de depuración más alto.

La segunda forma de delimitar una cadena de caracteres usa el carácter de comilla simple (‘). Cuando una cadena va encerrada entre comillas simples, los únicos caracteres de escape que serán comprendidos son \\ y \'. Esto es por convenio, así que se pueden tener comillas simples y barras invertidas en una cadena entre comillas simples. Las variables *no* se expandirán dentro de una cadena entre comillas simples.

Las cadenas se pueden concatenar usando el operador **.** (punto). Nótese que el operador + (suma) no sirve para esto.

Se puede acceder a los caracteres dentro de una cadena tratándola como un array de caracteres indexado numéricamente, usando una sintaxis similar a la de C. Vea un ejemplo más abajo.

```
<?php
/* Asignando una cadena. */
$str = "Esto es una cadena";

/* Añadiendo a la cadena. */
$str = $str . " con algo más de texto";

/* Otra forma de añadir, incluye un carácter de nueva línea protegido. */
$str .= " Y un carácter de nueva línea al final.\n";

/* Esta cadena terminará siendo '<p>Número: 9</p>' */
$num = 9;
$str = "<p>Número: $num</p>";

/* Esta será '<p>Número: $num</p>' */
$num = 9;
$str = '<p>Número: $num</p>';

/* Obtener el primer carácter de una cadena  */
$str = 'Esto es una prueba.';
$first = $str[0];

/* Obtener el último carácter de una cadena. */
$str = 'Esto es aún una prueba.';
$last = $str[strlen($str)-1];
?>
```

4.4.1. Conversión a cadenas

Un valor puede ser convertido a cadena usando el moldeamiento ***(string)*** o la función **strval()**.

Un valor booleano ***TRUE*** es convertido a la cadena *"1"*, el valor ***FALSE*** se representa como *""* (la cadena vacía). De esta forma, se puede convertir de ida y vuelta entre valores booleanos y de cadena.

Un número **integer** o de punto flotante (float) es convertido a una cadena que representa el número textualmente (incluyendo la parte del exponente para los números de punto flotante). Los números de punto flotante pueden ser convertidos usando la notación exponencial (*4.1E+6*).

Las **matrices** son siempre convertidas a la cadena *"Array"*; por esta razón, **echo()** y **print()** no pueden por su cuenta mostrar los contenidos de un valor array. Para ver un elemento sencillo, use una construcción como *echo $arr['foo']*.

Los **objetos** a partir de PHP 4 son convertidos siempre a la cadena *"Object"*. Por eso, al igual que pasa con las matrices **echo()** y **print()** no pueden por su cuenta mostrar los contenidos de un valor objecto.

Los recursos son siempre convertidos a cadenas con la estructura *"Resource id #1"* en donde *1* es el número único del valor recurso asignado por PHP en tiempo de ejecución. No se debe escribir código que dependa de esta estructura; está sujeta a cambios.

NULL se convierte siempre a una cadena vacía.

Como se ha indicado anteriormente, convertir directamente un valor tipo array, object, o resource a un string no ofrece información útil sobre los valores más allá de su tipo. Si desea inspeccionar el contenido de estos tipos debe usar las funciones **print_r()** y **var_dump()**.

La mayoría de valores PHP pueden ser convertidos también a string para su almacenamiento permanentemente. Este método es conocido como serialización (seriación), y es efectuado por la función **serialize().**

4.4.2. Conversión de cadenas a números

Cuando una cadena se evalúa como un valor numérico, el valor resultante y el tipo se determinan como sigue.

La cadena se evaluará como un doble si contiene cualquiera de los caracteres '.', 'e', o 'E'. En caso contrario, se evaluará como un entero.

El valor viene dado por la porción inicial de la cadena. Si la cadena comienza con datos de valor numérico, este será el valor usado. En caso contrario, el valor será 0 (cero). Los datos numéricos válidos son un signo opcional, seguido por uno o más dígitos (que opcionalmente contengan un punto decimal), seguidos por un exponente opcional. El exponente es una 'e' o una 'E' seguidos por uno o más dígitos.

Cuando la primera expresión es una cadena, el tipo de la variable dependerá de la segunda expresión.

```
$foo = 1 + "10.5";                // $foo es doble (11.5)
$foo = 1 + "-1.3e3";              // $foo es doble (-1299)
$foo = 1 + "bob-1.3e3";           // $foo es entero (1)
$foo = 1 + "bob3";                // $foo es entero (1)
$foo = 1 + "10 Cerditos";         // $foo es entero (11)
$foo = 1 + "10.0 Cerditos";       // $foo es double (11)
$foo = "10.0 cerdos " + 1;        // $foo es double (11)
$foo = "10.0 cerdos " + 1.0;      // $foo es double (11)
```

Si quisiera probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

4.5. Arrays

Los arrays actualmente actúan tanto como tablas hash (arrays asociativos) como arrays indexados (vectores).

4.5.1. Arrays unidimensionales

PHP soporta tanto arrays escalares como asociativos. De hecho, no hay diferencias entre los dos. Se puede crear una array usando las funciones **list()** o **array()**, o se puede asignar el valor de cada elemento del array de manera explícita.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
$matriz = array("foo" => "bar", 12 => true);
```

También se puede crear un array simplemente añadiendo valores al array. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

```
$a[] = "hola"; // $a[2] == "hola"
$a[] = "mundo"; // $a[3] == "mundo"
```

Los arrays se pueden ordenar usando las funciones **asort()**, **arsort()**, **ksort()**, **rsort()**, **sort()**, **uasort()**, **usort()**, y **uksort()** dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función **count()**.

Se puede recorrer un array usando las funciones **next()** y **prev()**. Otra forma habitual de recorrer un array es usando la función **each()**.

4.5.2. Arrays Multidimensionales

Los arrays multidimensionales son bastante simples actualmente. Para cada dimensión del array, se puede añadir otro valor [clave] al final:

```
$a[1]      = $f;           # ejemplos de una sola dimensión
$a["foo"]  = $f;

$a[1][0]     = $f;         # bidimensional
$a["foo"][2] = $f;         # (se pueden mezclar índices numéricos y asociativos)
$a[3]["bar"] = $f;         # (se pueden mezclar índices numéricos y asociativos)

$a["foo"][4]["bar"][0] = $f;   # tetradimensional!
```

En PHP3 no es posible referirse a arrays multidimensionales directamente dentro de cadenas. Por ejemplo, lo siguiente no tendrá el resultado deseado:

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: $a[3][bar]";
```

En PHP3, lo anterior tendrá la salida *Esto no va a funcionar: Array[bar]*. De todas formas, el operador de concatenación de cadenas se puede usar para solucionar esto:

```
$a[3]['bar'] = 'Bob';
echo "Esto va a funcionar: " . $a[3][‘bar’];
```

A partir de PHP4, sin embargo, todo el problema se puede solucionar encerrando la referencia al array (dentro de la cadena) entre llaves:

```
$a[3]['bar'] = 'Bob';
echo "Esto va a funcionar: {$a[3][‘bar’]}";
```

Se pueden "rellenar" arrays multidimensionales de muchas formas, pero la más difícil de comprender es cómo usar el comando **array()** para arrays asociativos. Estos dos trozos de código rellenarán el array unidimensional de la misma manera:

```
# Ejemplo 1:

$a["color"]  = "rojo";
$a["sabor"]  = "dulce";
$a["forma"]  = "redondeada";
$a["nombre"] = "manzana";
$a[3]        = 4;

# Ejemplo 2:
$a = array(
     "color" => "rojo",
     "sabor" => "dulce",
     "forma" => "redondeada",
     "nombre"  => "manzana",
     3       => 4
);
```

La función **array()** se puede anidar para arrays multidimensionales:

```
<?php
$a = array(
     "manzana"  => array(
          "color"  => "rojo",
          "sabor"  => "dulce",
          "forma"  => "redondeada"
     ),
     "naranja"  => array(
          "color"  => "naranja",
          "sabor"  => "ácido",
          "forma"  => "redondeada"
     ),
     "plátano"  => array(
          "color"  => "amarillo",
          "sabor"  => "paste-y",
          "forma"  => "aplatanada"
     )
);

echo $a["manzana"]["sabor"];    # devolverá "dulce"
?>
```

Siempre deben usarse comillas alrededor de un índice de matriz tipo cadena literal. Por ejemplo, *$foo['bar']* es correcto, mientras que *$foo[bar]* no lo es. Funciona pero no es correcto. Funciona porque PHP automáticamente convierte una *cadena pura* (una cadena sin comillas que no corresponda con símbolo conocido alguno) en una cadena que contiene la cadena pura. Por ejemplo, si no se ha definido una constante llamada *bar*, entonces PHP reemplazará su valor por la cadena *'bar'* y usará ésta última.

4.6. Objetos

Para inicializar un objeto, se usa la sentencia *new* para instanciar el objeto a una variable.

```
class foo {
    function do_foo () {
        echo "Haciendo foo.";
    }
}

$bar = new foo;
$bar->do_foo();
```

4.7. Recursos

Un valor tipo resource es una variable especial, que contiene una referencia a un recurso externo. Los recursos son creados y usados por funciones especiales. Gracias al sistema de conteo de referencias introducido con el Motor Zend de PHP 4, un recurso que ya no es referenciado es detectado automáticamente, y es liberado por el recolector de basura. Por esta razón, rara vez se necesita liberar la memoria manualmente. Los enlaces persistentes con bases de datos son una excepción a esta regla. Estos *no* son destruidos por el recolector de basura.

4.8. NULL

El valor especial **NULL** representa una variable que no tiene valor. **NULL** es el único valor posible del tipo NULL. Una variable es considerada como null si:

- ⇨ se le ha asignado la constante **`NULL`**.
- ⇨ no ha sido definida con valor alguno.
- ⇨ ha sido eliminada con **unset().**

Existe un solo valor de tipo null, y ese es la palabra clave **NULL**, insensible a mayúsculas y minúsculas.

```
$var = NULL;
```

Hay varias funciones interesantes para trabajar con el valor null: **is_null()** y **unset().**

4.9. Forzado de tipos

PHP no requiere (o soporta) la declaración explícita del tipo en la declaración de variables; el tipo de una variable se determina por el contexto en el que se usa esa variable. Esto quiere decir que si se asigna un valor de cadena a la variable *var*, *var* se convierte en una cadena. Si después se asigna un valor entero a la variable *var*, se convierte en una variable entera.

Un ejemplo de conversión de tipo automática en PHP es el operador suma '+'. Si cualquiera de los operandos es un flotante, entonces todos los operandos se evalúan como flotantes, y el resultado será un flotante. En caso contrario, los operandos se interpretarán como enteros, y el resultado será también un entero. Nótese que esto NO cambia los tipos de los operandos propiamente dichos; el único cambio está en cómo se evalúan los operandos.

```
$foo = "0";  // $foo es una cadena (ASCII 48)
$foo++;      // $foo es la cadena "1" (ASCII 49) convertido a entero (1)
$foo += 1;   // $foo ahora es un entero (2)
$foo = $foo + 1.3;  // $foo ahora es un doble (3.3)
$foo = 5 + "10 Cerditos Pequeñitos"; // $foo es entero (15)
$foo = 5 + "10 Cerditos";     // $foo es entero (15)
```

Si se desea obligar a que una variable sea evaluada con un tipo concreto se puede realizar un forzado de tipos. Si se desea cambiar el tipo de una variable se puede usar la función **settype()**.

Si quisiese probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo o usar la función **var_dump()**:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

> ***Nota:*** *La posibilidad de una conversión automática a array no está definida actualmente.*

```
$a = 1;       // $a es un entero
$a[0] = "f";  // $a se convierte en un array, en el que $a[0] vale "f"
```

Aunque el ejemplo anterior puede parecer que claramente debería resultar en que *$a* se convierta en un array, el primer elemento del cual es *'f'*, consideremos esto:

```
$a = "1";     // $a es una cadena
$a[0] = "f";  // ¿Qué pasa con los índices de las cadenas? ¿Qué ocurre?
```

Dado que PHP soporta indexación en las cadenas vía offsets usando la misma sintaxis que la indexación de arrays, el ejemplo anterior nos conduce a un problema: ¿debería convertirse $a en un array cuyo primer elemento sea "f", o debería convertirse "f" en el primer carácter de la cadena $a?

El forzado de tipos en PHP funciona como en C: el nombre del tipo deseado se escribe entre paréntesis antes de la variable a la que se pretende forzar.

```
$foo = 10;   // $foo es un entero
$bar = (double) $foo;   // $bar es un doble
```

Los forzados de tipo permitidos son:
(int), (integer) - fuerza a entero (integer)
(bool),(bolean) – fuerza a booleano
(real), (double), (float) - fuerza a doble (double)
(string) - fuerza a cadena (string)
(array) - fuerza a array (array)
(object) - fuerza a objeto (object)

Nótese que las tabulaciones y espacios se permiten dentro de los paréntesis, así que los siguientes ejemplos son funcionalmente equivalentes:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Puede no ser obvio que ocurrirá cuando se fuerce entre ciertos tipos. Por ejemplo, lo siguiente debería ser tenido en cuenta.

Cuando se fuerza el cambio de un escalar o una variable de cadena a un array, la variable se convertirá en el primer elemento del array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0];  // produce la salida 'ciao'
```

Cuando se fuerza el tipo de una variable escalar o de una cadena a un objeto, la variable se convertirá en un atributo del objeto; el nombre del atributo será 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar;  // produce la salida 'ciao'
```

Cuando se fuerza el tipo de una variable a booleano, el valor de la variable será **FALSE** si: es el valor booleano *FALSE*, si es el entero *0*, si es el flotante *0.0* si es la cadena vacía o la cadena *"0"*, si es un array con cero elementos, si es *NULL*; en cualquier otro caso el valor será **TRUE**.

5. Operadores

- *Operadores Aritméticos*
- *Operadores de Asignación*
- *Operadores de Bits*
- *Operadores de Comparación*
- *Operadores de Control de Errores*
- *Operador de Ejecución*
- *Operadores de Incremento/decremento*
- *Operadores Lógicos*
- *Operador de cadenas de texto*
- *Operadores de Matrices*
- *Operadores de Tipo*
- *Precedencia y asociatividad de operandos*

5.1. Operadores Aritméticos

Disponemos de los clásicos operadores aritméticos:

Operación	Nombre	Resultado
$a + $b	Suma	Suma de $a y $b.
$a - $b	Resta	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de la operación $a / $b.
-$a	Negación	El opuesto de $a

5.2. Operadores de Asignación

El operador básico de asignación es "=". A primera vista podrías pensar que es el operador de comparación "igual que". Pero no. Realmente significa que el operando de la izquierda toma el valor de la expresión a la derecha, (esto es, "toma el valor de").

El valor de una expresión de asignación es el propio valor asignado. Esto es, el valor de "$a = 3" es 3. Esto permite hacer cosas curiosas como

```
$a = ($b = 4) + 5; // ahora $a es igual a 9, y $b vale 4.
```

Además del operador básico de asignación, existen los "operadores combinados" para todas las operaciones aritméticas y de cadenas que sean binarias. Este operador combinado permite, de una sola vez, usar una variable en una expresión y luego establecer el valor de esa variable al resultado de la expresión. Por ejemplo:

```
$a = 3;
$a += 5; // establece $a a 8, como si hubiésemos escrito: $a = $a + 5;
$b = "Hola ";
$b .= "Ahí!"; // establece $b a "Hola Ahí!", igual que si hiciésemos $b = $b . "Ahí!";
```

Al igual que en C, C++ y Java, en PHP se dispone de estos operadores combinados para, en una sola operación, operar sobre una variable y asignarle a esa misma variable el resultado.

Las operaciones susceptibles de ser usadas con estos operadores son:

```
+  -  *  /  %  &  ^  .  >>  y  <<
```

resultando en los nuevos signos de operación-asignación:

```
+=  -=  *=  /=  %=  &=  ^=  .=  >>=  y  <<=
```

Ejemplos de uso:

```
$var1 += 3;          // $var1 = $var1 + 3;
$var2 /= 2;          // $var2 = $var2 / 2;
$var3 >>= 1;         // $var3 = $var3 >> 1;
```

Fíjese en que la asignación realiza una nueva copia de la variable original (asignación por valor), por lo que cambios a la variable original no afectan a la copia. Esto puede tener interés si necesitas copiar algo como un array con muchos elementos dentro de un bucle que se repita muchas veces (cada vez se realizará una nueva copia del array). PHP soporta asignación por referencia, usando la sintaxis *$var = &$othervar;* , pero esto no es posible en PHP3. 'Asignación por referencia' quiere decir que ambas variables acabarán apuntando al mismo dato y que nada es realmente copiado. A partir de PHP5 los objetos son asignados por referencia a menos que explícitamente se diga lo contrario con la nueva palabra clave **clone.**

5.3. Operadores de bits

Veamos ahora los operadores BIT a BIT de que dispone PHP:

Operación	Nombre	Resultado
$a & $b	Y	Se ponen a 1 los bits que están a 1 en $a y $b.
$a | $b	O	Se ponen a 1 los bits que están a 1 en $a o $b.
$a ^ $b	O Exclusivo (XOR)	Se ponen a 1 los bits que están a 1 en $a o $b, pero no en ambos.
~ $a	No	Se invierten los bits (se cambian 1 por 0 y viceversa.)
$a << $b	Desp. Izq.	Desplaza $b posiciones a la izquierda todos los bits de $a (cada posición supone multiplicar por dos).
$a >> $b	Desp. Drch.	Desplaza $b posiciones a la derecha todos los bits de $a (cada posición supone dividir por dos).

5.4. Operadores de Comparación

Los operadores de comparación, como su nombre indica, permiten comparar dos valores.

Operación	Nombre	Resultado
$a == $b	Igualdad	Compara si el valor de los dos operandos es el mismo.
$a === $b	Identidad	Compara si el valor es el mismo y, además, el tipo coincide (A partir de PHP4).
$a != $b	No igual	Cierto si el valor de $a no es igual al de $b.
$a !== $b	No idéntico	Cierto si $a no es igual a $b, o si no tienen el mismo tipo.
$a < $b	Menor que	Cierto si $a es estrictamente menor que $b.

Operación	Nombre	Resultado
$a > $b	Mayor que	Cierto si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	Cierto si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	Cierto si $a es mayor o igual que $b.
$a <> $b	No igual	Cierto si el valor de $a no es igual al de $b

Si se compara un entero con una cadena, la cadena es convertida a un número. Si se compara dos cadenas numéricas, son comparadas como enteros. Estas reglas también se aplican a la sentencia **switch**.

Otro operador condicional es el operador "**?:**" (o ternario), que funciona como en C y otros muchos lenguajes.

```
(expr1) ? (expr2) : (expr3);
```

La expresión toma el valor *expr2* si *expr1* se evalúa a cierto, y *expr3* si *expr1* se evalúa a falso.

```
$cad = $a > $b ? “a es mayor que b” : “a no es mayor que b”;
```

5.5. Operadores de Control de Errores

PHP ofrece soporte para un operador de control de errores: el signo de arroba (**@**). Cuando es colocado al comienzo de una expresión en PHP, cualquier mensaje de error que pudiera generarse a causa de esa expresión será ignorado.

Si la característica **track_errors** está habilitada, cualquier mensaje de error generado por la expresión será almacenado en la variable **$php_errormsg**. La variable será sobrescrita en cada instancia de error, así que realice sus chequeos de forma temprana si quiere usarla.

```
<?php
/* Error intencionado de archivo */
$mi_archivo = @file ('archivo_que_no_existe')
     or die ("La apertura de archivo ha fallado: el error fue
             '$php_errormsg'");

// esto funciona con cualquier expresión, no solo con funciones:
$valor = @$cache[$llave];
// no producirá una anotación si el índice $llave no existe.

?>
```

Puede usar el operado @ al inicio de variables, llamadas a funciones y sentencias include(), constantes. No puede usarlo sobre definiciones de función o clase, ni sobre estructuras condicionales como if y foreach.

5.6. Operador de ejecución

PHP soporta un operador de ejecución: la comilla invertida (`). ¡Fíjese que no son comillas normales! PHP intentará ejecutar la instrucción contenida dentro de las

comillas invertidas como si fuera un comando del shell; y su salida devuelta como el valor de esta expresión (i.e., no tiene por qué ser simplemente volcada como salida; puede asignarse a una variable). El uso de este operados es idéntico a la función **shell_exec()**.

```
$output = `dir /a-d`;
echo "<pre>$output</pre>";
```

5.7. Operadores de Incremento/decremento

PHP soporta los operadores de pre y post decremento y incremento al estilo de C. Estos operadores no afectan a valores booléanos y incrementar NULL resulta en 1.

Operación	Nombre	Resultado
++$a	Pre-incremento	Incrementa $a en 1, y devuelve $a (ya incrementado)
$a++	Post-incremento	Devuelve $a (sin incrementar), y después lo incrementa en 1.
--$a	Pre-decremento	Decrementa $a en 1, y después lo devuelve.
$a--	Post-decremento	Devuelve $a, y después lo incrementa en 1.

He aquí un listado de ejemplo:

```
<?php
echo "<h3>Postincremento</h3>";
$a = 5;
echo "Debería ser 5: " . $a++ . "<br>\n";
echo "Debería ser 6: " . $a . "<br>\n";

echo "<h3>Preincremento</h3>";
$a = 5;
echo "Debería ser 6: " . ++$a . "<br>\n";
echo "Debería ser 6: " . $a . "<br>\n";

echo "<h3>Postdecremento</h3>";
$a = 5;
echo "Debería ser 5: " . $a-- . "<br>\n";
echo "Debería ser 4: " . $a . "<br>\n";

echo "<h3>Predecremento</h3>";
$a = 5;
echo "Debería ser 4: " . --$a . "<br>\n";
echo "Debería ser 4: " . $a . "<br>\n";
?>
```

5.8. Operadores lógicos

Los operadores lógicos realizan operaciones dependiendo del valor booleano de los operandos.

Operación	Nombre	Resultado
$a and $b	Y	Cierto si $a y $b son ciertos.
$a or $b	O	Cierto si $a o $b es cierto.
$a xor $b	O Exclusivo.	Cierto si $a o $b es cierto, pero no ambos.
! $a	No	Cierto si $a es falso.
$a && $b	Y	Cierto si $a y $b son ciertos.
$a || $b	O	Cierto si $a o $b es cierto.

La razón de que haya dos operadores distintos para las operaciones *Y* y *O* lógicas es que tienen distinta precedencia.

5.9. Operador de cadenas de texto

Para operar con cadenas sólo disponemos de un operador: la concatenación de cadenas representada por el punto '**.**'.

Por ejemplo:

```
$a = 1;
$b = 2;
$c = “El resultado de “ . $a . “ + “ . $b . “ es “ . ($a + $b);
```

Que dejaría en *$c* la cadena "*El resultado de 1 + 2 es 3*". Antes de cada concatenación se realizarán las conversiones de tipo que fueran necesarias (en el ejemplo, los enteros se convierten a cadenas.)

5.10. Operadores de Matrices

Ejemplo	Nombre	Resultado
$a + $b	Unión	Unión de $a y $b.
$a == $b	Igualdad	Cierto si $a y $b tienen las mismas parejas llave/valor.
$a=== $b	Identidad	Cierto si $a y $b tienen las mismas parejas llave/valor en el mismo orden y de los mismos tipos.
$a != $b	No-igualdad	Cierto si $a no es igual a $b.
$a <> $b	No-igualdad	Cierto si $a no es igual a $b.
$a!== $b	No-identidad	Cierto si $a no es idéntico a $b.

El operador **+** adiciona elementos de las claves restantes de la matriz del lado derecho a aquella al lado izquierdo, al mismo tiempo que cualquier clave duplicada NO es sobrescrita.

```
<?php
$a = array("a" => "manzana", "b" => "banano");
$b = array("a" => "pera", "b" => "fresa", "c" => "cereza");

$c = $a + $b; // Unión de $a y $b
echo "Unión de \$a y \$b: \n";
var_dump($c);

$c = $b + $a; // Unión de $b y $a
echo "Unión de \$b y \$a: \n";
var_dump($c);
?>
```

Cuando sea ejecutado, este script producirá la siguiente salida:

```
Unión de $a y $b:
array(3) {
  ["a"]=>
  string(7) "manzana"
  ["b"]=>
```

```
  string(6) "banano"
  ["c"]=>
  string(6) "cereza"
}
Unión de $b y $a:
array(3) {
  ["a"]=>
  string(4) "pera"
  ["b"]=>
  string(5) "fresa"
  ["c"]=>
  string(6) "cereza"
}
```

Los elementos de las matrices son considerados equivalentes en la comparación si éstos tienen la misma clave y valor.

```
<?php
$a = array("manzana", "banano");
$b = array(1 => "banano", "0" => "manzana");

var_dump($a == $b); // bool(true) tiene las mismas clave-valor
var_dump($a === $b); // bool(false) pero en orden distinto
?>
```

5.11. Operadores de Tipo

instanceof es usado para determinar si una variable PHP es una instancia de objeto de cierta clase:

```
<?php
class MiClase
{
}

class NoMiClase
{
}
$a = new MiClase;

var_dump($a instanceof MiClase);
var_dump($a instanceof NoMiClase);
?>
```

El resultado del ejemplo sería:

```
bool(true)
bool(false)
```

El operador **instanceof** fue introducido en PHP5. Anteriormente se usaba la función **is_a()** pero desde la introdución del operados instanceof, la función anterior se considera obsoleta.

5.12. Precedencia y asociatividad de operandos

La precedencia de los operandos resuelve el orden en el que se evalúa una expresión múltiple que no ha sido delimitada con paréntesis. Por ejemplo, *1 + 5 * 3* en PHP daría como resultado *1 + (5 * 3) = 16* y no *(1 + 5) * 3 = 18* ya que el producto tiene mayor precedencia que la suma.

La tabla muestra la asociatividad de los operandos en PHP, y está ordenada en orden creciente de precedencia (los más prioritarios al final):

Asociatividad	Operandos
izquierda	,
izquierda	or
izquierda	xor
izquierda	and
Derecha	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
izquierda	? :
izquierda	||
izquierda	&&
izquierda	|
izquierda	^
izquierda	&
no-asociativo	== != === !==
no-asociativo	< <= > >=
izquierda	<< >>
izquierda	+ - .
izquierda	* / %
derecha	!
no-asociativo	~ - (int) (float) (string) (array) (object) (bool) @
no-asociativo	++ --
izquierda	[
no-asociativo	clone new

La asociatividad de izquierda quiere decir que la expresión es evaluada desde la izquierda a la derecha, la asociatividad de derecha quiere decir lo contrario.

```
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

6. Estructuras de Control

- *if*
- *else*
- *elseif*
- *while*
- *do..while*
- *for*
- *foreach*
- *break*
- *continue*
- *switch*
- *declare*
- *return*
- *require()*
- *include()*
- *require_once()*
- *include_once()*
- *Sintaxis Alternativa de Estructuras de Control*

Todo script PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia. En este capítulo se describen los diferentes tipos de sentencias.

6.1. if

La construcción **if** es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura **if** que es similar a la de C:

```
<?php
if (expr)
    sentencia
?>
```

Como se describe en la sección sobre expresiones, *expr* se evalúa a su valor condicional (boolean). Si *expr* se evalúa como **TRUE**, PHP ejecutará la *sentencia*, y si se evalúa como **FALSE** - la ignorará. Se puede encontrar más información sobre los valores evaluados como **FALSE** en la sección sobre el tipo de datos booleano.

El siguiente ejemplo mostraría *a es mayor que b* si *$a* fuera mayor que *$b*:

```
<?php
if ($a > $b)
   print "a es mayor que b";?>
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula **if**. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este

código mostraría *a es mayor que b* si *$a* fuera mayor que *$b*, y entonces asignaría el valor de *$a* a *$b*:

```
<?php
 if ($a > $b) {
     print "a es mayor que b";
     $b = $a;
 }
?>
```

Las sentencias **if** se pueden anidar indefinidamente dentro de otras sentencias **if**, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

6.2. else

A menudo queremos ejecutar una sentencia si se cumple una cierta condicion, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve **else**. **else** extiende una sentencia **if** para ejecutar una sentencia en caso de que la expresión en la sentencia **if** se evalúe como **FALSE**. Por ejemplo, el siguiente código mostraría *a es mayor que b* si *$a* fuera mayor que *$b*, y *a NO es mayor que b* en cualquier otro caso:

```
<?php
 if ($a > $b) {
     print "a es mayor que b";
 } else {
     print "a NO es mayor que b";
 }
?>
```

La sentencia **else** se ejecuta solamente si la expresión **if** se evalúa como **FALSE**, y si hubiera alguna expresión **elseif** - sólo si se evaluaron también a **FALSE** (Ver elseif).

6.3. elseif

elseif, como su nombre sugiere, es una combinación de **if** y **else**. Como **else**, extiende una sentencia **if** para ejecutar una sentencia diferente en caso de que la expresión **if** original se evalúa como **FALSE**. No obstante, a diferencia de **else**, ejecutará esa expresión alternativa solamente si la expresión condicional **elseif** se evalúa como **TRUE**. Por ejemplo, el siguiente código mostraría *a es mayor que b*, *a es igual a b* o *a es menor que b*:

```
<?php
 if ($a > $b) {
     print "a es mayor que b";
 } elseif ($a == $b) {
     print "a es igual que b";
 } else {
     print "a es mayor que b";
 }
?>
```

Puede haber varios **elseif** dentro de la misma sentencia **if**. La primera expresión **elseif** (si hay alguna) que se evalúe como **TRUE** se ejecutaría. En PHP, también se puede escribir 'else if' (con dos palabras) y el comportamiento sería idéntico al de un 'elseif' (una sola palabra). El significado sintáctico es ligeramente distinto (si estas

familiarizado con C, es el mismo comportamiento) pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia **elseif** se ejecuta sólo si la expresión **if** precedente y cualquier expresión **elseif** precedente se evalúan como **FALSE**, y la expresión elseif actual se evalúa como **TRUE**

6.4. while

Los bucles **while** son los tipos de bucle más simples en PHP. Se comportan como su contrapartida en C. La forma básica de una sentencia **while** es:

```
while (expr) sentencia
```

El significado de una sentencia **while** es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión while se evalúe como **TRUE**. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión while se evalúa como **FALSE** desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.

Como con la sentencia **if**, se pueden agrupar multiples sentencias dentro del mismo bucle **while** encerrando un grupo de sentencias con llaves.

El siguiente ejemplo imprime números del 1 al 10:

```
<?php
$i = 1;
 while ($i <= 10) {
     print $i++;  /* el valor impreso sería
                     $i antes del incremento
                     (post-incremento) */
 }

?>
```

6.5. do..while

Los bucles **do..while** son muy similares a los bucles while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares **while** es que se garantiza la ejecución de la primera iteración de un bucle **do..while** (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle **while** regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como **FALSE** desde el principio la ejecución del bucle finalizará inmediatamente).

Hay una sola sintaxis para los bucles **do..while**:

```
<?php
 $i = 0;
 do {
     print $i;
 } while ($i>0);?>
```

El bucle de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como **FALSE** (*$i no es más grande que 0*) y la ejecución del bucle finaliza.

Los usuarios avanzados de C pueden estar familiarizados con un uso distinto del bucle **do..while**, para permitir parar la ejecución en medio de los bloques de código, encapsulandolos con *do..while(0),* y usando la sentencia break. El siguiente fragmento de código demuestra esto:

```
<?php
do {
     if ($i < 5) {
         print "i no es lo suficientemente grande";
         break;
     }
     $i *= $factor;
     if ($i < $minimum_limit) {
         break;
     }
     print "i es correcto";
     /* procesa i */
 } while(0);
?>
```

6.6. for

Los bucles **for** son los bucles más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un bucle **for** es:

```
for (expr1; expr2; expr3) sentencia
```

La primera expresión (*expr1*) se evalúa (ejecuta) incondicionalmente una vez al principio del bucle.

Al comienzo de cada iteración, se evalúa *expr2* . Si se evalúa como **TRUE**, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como **FALSE**, la ejecución del bucle finaliza.

Al final de cada iteración, se evalúa (ejecuta) *expr3*.

Cada una de las expresiones puede estar vacía. Que *expr2* esté vacía significa que el bucle debería correr indefinidamente (PHP implicitamente lo considera como **TRUE**, al igual que C). Esto puede que no sea tan inútil como se podría pensar, puesto que a menudo se quiere salir de un bucle usando una sentencia break condicional en vez de usar la condición de for.

Considere los siguientes ejemplos. Todos ellos muestran números del 1 al 10:

```
<?php
 /* ejemplo 1 */

 for ($i = 1; $i <= 10; $i++) {
     print $i;
 }
```

```
 /* ejemplo 2 */

 for ($i = 1; ;$i++) {
     if ($i > 10) {
         break;
     }
     print $i;
 }

 /* ejemplo 3 */

 $i = 1;
 for (;;) {
     if ($i > 10) {
         break;
     }
     print $i;
     $i++;
 }

 /* ejemplo 4 */

 for ($i = 1; $i <= 10; print $i, $i++) ;
?>
```

Por supuesto, el primer ejemplo parece ser el mas elegante (o quizás el cuarto), pero uno puede descubrir que ser capaz de usar expresiones vacías en bucles **for** resulta útil en muchas ocasiones.

Otros lenguajes poseen una sentencia foreach para traducir un array o una tabla hash. PHP3 no posee tal construcción; PHP4 sí (ver foreach). En PHP3, se puede combinar while con las funciones **list()** y **each()** para conseguir el mismo efecto.

6.7. foreach

PHP 4 (PHP3 no) incluye una construcción **foreach**, tal como perl y algunos otros lenguajes. Esto simplemente da un modo fácil de iterar sobre matrices. **foreach** funciona solamente con matrices y devolverá un error si se intenta utilizar con otro tipo de datos ó variables no inicializadas. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

```
foreach(expresion_array as $value) sentencia
foreach(expresion_array as $key => $value) sentencia
```

La primera forma recorre el array dado por expresion_array. En cada iteración, el valor del elemento actual se asigna a $value y el puntero interno del array se avanza en una unidad (así en el siguiente paso, se estará mirando el elemento siguiente).

La segunda manera hace lo mismo, salvo que la clave del elemento actual será asignada a la variable $key en cada iteración.

> ***Nota:*** *Cuando foreach comienza su primera ejecución, el puntero interno a la matriz se reinicia automáticamente al primer elemento de la matriz. Esto significa que no se necesita llamar a* ***reset()*** *antes de un bucle foreach.*
>
> ***Nota:*** *Hay que tener en cuenta que foreach trabaja con una copia de la matriz especificada y no la lista en si, por ello el puntero de la lista no es modificado como en la función* ***each()****, y los cambios en el elemento de la matriz retornado no afectan a la matriz original. De todas maneras el puntero interno a la matriz original avanza al procesar la matriz.*

Suponiendo que bucle foreach se ejecuta hasta el final, el puntero interno a la matriz estará al final de la matriz.

Nota: *foreach no soporta la caracteristica de suprimir mensajes de error con '@'.*

Puede haber observado que las siguientes son funcionalidades idénticas:

```
<?php
$arr = array("one", "two", "three");
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}

foreach ($arr as $value) {
    echo "Value: $value<br>\n";
}
?>
```

Las siguientes también son funcionalidades idénticas:

```
<?php
reset( $arr );
while( list( $key, $value ) = each( $arr ) ) {
   echo "Key: $key; Valor: $value<br>\n";
}

foreach( $arr as $key => $value ) {
   echo "Key: $key; Valor: $value<br>\n";
}
?>
```

Algunos ejemplos más para demostrar su uso:

```
<?php
/* foreach ejemplo 1: sólo valor*/
$a = array(1, 2, 3, 17);

foreach($a as $v) {
   print "Valor actual de \$a: $v.<br>\n";
}

/* foreach ejemplo 2: valor (con clave impresa para ilustrar) */
$a = array(1, 2, 3, 17);

$i = 0; /* sólo para propósitos demostrativos */

foreach($a as $v) {
   print "\$a[$i] => $v.<br>\n";
   $i++;
}

/* foreach ejemplo 3: clave y valor */
$a = array(
   "uno" => 1,
   "dos" => 2,
   "tres" => 3,
   "diecisiete" => 17
);

foreach($a as $k => $v) {
   print "\$a[$k] => $v.<br>\n";
}

/* foreach ejemplo 4: matriz multi-dimensional */
unset($a);
$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
```

```
$a[1][1] = "z";

foreach($a as $k1 => $v1) {
    foreach ($v1 as $k2 => $v2) {
        print "$k1,$k2 => $v2<br>\n";
    }
}

/* foreach ejemplo 5: matriz dinámica */

foreach(array(1, 2, 3, 4, 5) as $v) {
    print "$v<br>\n";
}
?>
```

6.8. break

break sale de la estructuras de control iterante (bucle) actuales **for**, **while**, o **switch**.

break accepta un parámetro opcional, el cual determina de cuantas estructuras de control hay que salir.

```
<?php
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
    if ($val == 'stop') {
        break;    /* Tambien se podría poner break 1; */
    }
    echo "$val<br>\n";
}

/* Usando el argumento opcional. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "En el 5<br>\n";
        break 1;  /* Sale solo del switch. */
    case 10:
        echo "En el 10; saliendo<br>\n";
        break 2;  /* Sale del switch y del while. */
    default:
        break;
    }
}
?>
```

6.9. continue

continue se usa dentro de la estructura del bucle para saltar el resto de la iteración actual del bucle y continuar la ejecución al comienzo de la siguiente iteración.

> ***Nota:*** *Tener en cuenta que en PHP la declaración switch es considerada una estructura de bucle por continue.*

continue accepta un parámetro opcional, el cual determina cuantos niveles (bluces) hay que saltar antes de continuar con la ejecución.

```
<?php
$arr = array ('one', 'two', 'three', 'four', 'five', 'six');
while (list ($key, $value) = each ($arr)) {
    if (!($key % 2)) { // Salta los elementos pares
        continue;
```

```
    }
    echo "$value<br>\n";
}
$i = 0;
while ($i++ < 5) {
    echo "Externo<br>\n";
    while (1) {
        echo "  Medio<br>\n";
        while (1) {
            echo "    Interno<br>\n";
            continue 3;
        }
        echo "Esto nunca se imprime.<br>\n";
    }
    echo "Y esto tampoco.<br>\n";
}
?>
```

6.10. switch

La sentencia **switch** es similar a una serie de sentencias **if** en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con nuchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia **switch**.

> ***Nota:*** *Tener en cuenta que al contrario que otros lenguajes de programación, continue se aplica a switch y funciona de manera similar a break. Si teneis un switch dentro de un bucle y deseais continuar con el paso siguiente en el bucle externo, usar continue 2.*

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias if, y el otro usa la sentencia switch:

```
<?php
$i=2;
if ($i == 0) {
    print "i es igual a 0<br>\n";
} elseif ($i == 1) {
    print "i es igual a 1<br>\n";
} elseif ($i == 2) {
    print "i es igual a 2<br>\n";
}

switch ($i) {
    case 0:
        print "i es igual a 0<br>\n";
        break;
    case 1:
        print "i es igual a 1<br>\n";
        break;
    case 2:
        print "i es igual a 2<br>\n";
        break;
}
?>
```

Es importante entender cómo se ejecuta la sentencia **switch** para evitar errores. La sentencia **switch** ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia **case** con un valor que coincide con el valor de la expresión switch PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o la primera vez que vea una sentencia **break**. Si no se escribe una sentencia **break** al final de una lista de sentencias **case**, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
<?php
 switch ($i) {
     case 0:
         print "i es igual a 0";
     case 1:
         print "i es igual a 1";
     case 2:
         print "i es igual a 2";
 }
?>
```

Aquí, si *$i es igual a 0*, ¡PHP ejecutaría todas las sentecias print! Si *$i es igual a 1*, PHP ejecutaría las últimas dos sentencias print y sólo si *$i es igual a 2*, se obtendría la conducta 'esperada' y solamente se mostraría *'i es igual a 2'*. Así, es importante no olvidar las sentencias **break** (incluso aunque pueda querer evitar escribirlas intencionadamente en ciertas circunstancias).

En una sentencia **switch**, la condición se evalúa sólo una vez y el resultado se compara a cada sentencia **case**. En una sentencia **elseif**, la condición se evalúa otra vez. Si tu condición es más complicada que una comparación simple y/o está en un bucle estrecho, un switch puede ser más rápido.

La lista de sentencias de un **case** puede también estar vacía, lo cual simplemente pasa el control a la lista de sentencias del siguiente case.

```
<?php
switch ($i) {
     case 0:
     case 1:
     case 2:
         print "i es menor que 3, pero no negativo";
         break;
     case 3:
         print "i es 3";
 }
?>
```

Un caso especial es el **default** case. Este case coincide con todo lo que no coincidan los otros case. Por ejemplo:

```
<?php
 switch ($i) {
     case 0:
         print "i es igual a 0";
         break;
     case 1:
         print "i es igual a 1";
         break;
     case 2:
         print "i es igual a 2";
         break;
     default:
         print "i no es igual a 0, 1 o 2";
 }
?>
```

La expresión **case** puede ser cualquier expresión que se evalúe a un tipo simple, es decir, números enteros o de punto flotante y cadenas de texto. No se pueden usar aquí ni arrays ni objetos a menos que se conviertan a un tipo simple.

6.11. declare

La construcción **declare** es usada para definir directivas de ejecución para un bloque de código. La sintaxis de **declare** es similar a la de las otras estructuras de control:

```
declare (directiva) sentencia
```

Directiva permite asignar el comportamiento del bloque **declare**. Actualmente una sola directiva es reconocida: la directiva *ticks*

La *sentencia* es lo que se ejecuta -- Como se ejecuta y que efectos secundarios tiene depende de la directiva definida en la *directiva*.

El constructor *declare* se puede usar tambien globalmente, afectando a todo el código que le sigue.

```
<?php
// Estos son lo mismo:

// se puede usar este:
declare(ticks=1) {
    // script completo aqui
}

// o este:
declare(ticks=1);
// script completo aqui
?>
```

Un "**tick**" es un evento que ocurre por cada N sentencias de bajo nivel ejecutadas dentro del bloque *declare*. El valor de N es especificado por *ticks=N* como *directiva* dentro de *declare*.

El evento que ocurre en cada "tick" es especificado usando la función **register_tick_function()**.

```
<?php
// Una función que registra el momento en que se llama
function profile ($dump = FALSE)
{
    static $profile;

    // Devuelve los tiempos almacenado en el perfil, despues lo borra
    if ($dump) {
        $temp = $profile;
        unset ($profile);
        return ($temp);
    }

    $profile[] = microtime();
}

// Establace cual es la función que ocurre en el tick
register_tick_function("profile");

// Inicializa la función antes de declarar el bloque
profile ();

// Ejecuta el bloque y lanza el tick cada 2 sentencias.
declare (ticks=2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text (md5($x), md5($x*$x)), "<br />;";
```

```
    }
}

// Muestra el contenido del perfil
print_r (profile (TRUE));
?>
```

Este ejemplo perfila el código PHP dentro del bloque 'declare', grabando la hora, una sentencia si y otra no, cuando fue ejecutada. Esta información puede ser usada para encontrar areas en donde el codigo es lento. Este proceso se puede implementar de diferentes maneras: usando "ticks" es más conveniente y facil de implementar.

"Ticks" es una manera muy buena de eliminar errores, implementando simples trabajos en paralelo, I/O en modo desatendido y otras tareas.

6.12. return

Si se llama desde una función, **return()** termina inmediatamente la ejecución de la función y retorna su argumento como valor de la función. **return()** tambien terminará la ejecución de una sentencia **eval()** ó un script PHP.

Si el script actual ha sido incluido ó requerido con **include()** ó **require()**, el control es transferido al script que llamo al script incluido. Además, si el script actual fue incluido, el valor dado a **return()** será retornado como el valor de la llamada **include()**. Si **return()** es invocado desde el script principal, la ejecución terminara inmediatamente.

> ***Nota:*** *Tener en cuenta que ya que* ***return()*** *es un constructor del lenguaje y no una función, los paréntesis alrededor de sus argumentos,* no *son necesarios, es más no se suelen utilizar tan a menudo, aunque funciona igual de una manera u otra.*

6.13. require()

La sentencia **require()** incluye y evalua el archivo especificado.

require() incluye y evalua el archivo especificado. Información detallada de como esta inclusión funciona se puede encontrar en la documentación de la función **include()**.

require() y **include()** son idénticas en todos los aspectos excepto en el modo de actuar ante un error. **include()** produce un Warning mientras que **require()** produce un Error Fatal. En otras palabras, no dude en utilizar **require()** si quiere que un fichero no encontrado cuelgue el procesamiento de la página. **include()** no se comporta de esta manera, el script seguirá funcionando de todas maneras. Asegurarse que include_path este configurado bien.

```
<?php
require 'prepend.php';
require $somefile;
require ('somefile.txt');
?>
```

> ***Nota:*** *Con anterioridad a PHP 4.0.2, se aplica lo siguiente:* ***require()*** *siempre intentará leer el fichero a incluir, incluso si la linea donde se encuentra* ***require()*** *nunca es ejecutada. Sin embargo, si la linea donde se encuentra* ***require()*** *no es ejecutada, tampoco lo hará el código incluido.*

6.14. include()

La sentencia **include()** incluye y evalúa el archivo especificado.

Esta documentación también se aplica a la función **require()**. **require()** y **include()** son idénticas en todos los aspectos excepto en el modo de actuar ante un error. **include()** produce un Warning mientras que **require()** produce un Error Fatal. En otras palabras, no dude en utilizar **require()** si quiere que un fichero no encontrado cuelgue el procesamiento de la página. **include()** no se comporta de esta manera, el script seguirá funcionando de todas maneras. Asegurarse que include_path este configurado bien.

Cuando un fichero es incluido, el código que contiene hereda el ámbito de variables de la línea en donde el include ocurre. Cualquier variable disponible en esa línea en el fichero desde donde se hace la inclusión estará disponible en el fichero incluido a partir de ese momento.

apar614_02.php
```
<?php

$color = 'green';
$fruit = 'apple';

?>
```

apar614_01.php
```
<?php

echo "A $color $fruit"; // A

include 'apar614_02.php';

echo "A $color $fruit"; // A green apple

?>
```

Si la inclusión ocurre dentro de una función en el fichero donde se incluye, todo el código contenido en el fichero incluido se comportará como si hubiese sido definido dentro de esta función.

```
<?php

function foo()
{
    global $color;

    include 'apar614_02.php';

    echo "A $color $fruit";
}

/* apar614_02.php esta en el ambito de foo() asi que *
 * $fruit NO está disponible fuera de este     *
 * ámbito.  $color si porque se ha declarado   *
 * como global.                                */

foo();                    // A green apple
echo "A $color $fruit";   // A green
```

```
?>
```

Cuando un fichero es incluido, el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

Si "URL fopen wrappers" esta activada en PHP (como está en la configuración inicial), se puede especificar el fichero que se va a incluir usando una URL (via HTTP u otro mecanismo soportado) en vez de un fichero local. Si el servidor destino interpreta el fichero destino como código PHP, se puede mandar variables al fichero incluido usando una cadena URL de petición, tal como se hace con HTTP GET. Esto no es lo mismo que incluir un fichero y que este fichero herede las variables del fichero padre; el script es ejecutado en el servidor remoto y el resultado es incluido en en script local.

Versiones de PHP para Windows anteriores a la versión 4.3.0, no soportan el acceso remoto a ficheros en esta función, no funcionará incluso activando allow_url_fopen.

```
<?php

/* Este ejemplo asume que www.example.com esta configurado para
interpretar ficheros .php y no ficheros .txt . Asi mismo, las variables
$foo y $bar están disponibles al incluir el fichero.*/

// No funciona; file.txt no es manejado por example.com como PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// No funciona; busca un fichero llamado 'file.php?foo=1&bar=2' // en
local.
include 'file.php?foo=1&bar=2';

// Funciona.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt';  // Funciona.
include 'file.php';  // Funciona.

?>
```

Ya que **include()** y **require()** son contructores especiales del lenguaje, se deben de incluir dentro del bloque de una sentencia, si están dentro de un bloque condicional.

```
<?php

// Esto está MAL y no funcionará como se desea.
if ($condition)
    include $file;
else
    include $other;


// Esto es CORRECTO.
if ($condition) {
    include $file;
} else {
    include $other;
}

?>
```

Es posible ejecutar una sentencia **return** dentro de un archivo incluido para terminar el procesado de ese archivo y volver al archivo de comandos que lo llamó. Tambien es posible retornar valores de ficheros incluidos. Se puede coger el valor de la llamada "**include**" como se haría con una función normal.

> ***Nota:*** *En PHP3, return no puede aparecer dentro de un bloque a menos que sea un bloque de función, en el cual return se aplica a esa función y no al archivo completo.*

```
return.php
<?php
$var = 'PHP';
return $var;
?>

noreturn.php
<?php
$var = 'PHP';
?>

apar614_05.php
<?php
$foo = include 'return.php';
echo $foo; // imprime 'PHP'
$bar = include 'noreturn.php';
echo $bar; // imprime 1

?>
```

$bar es igual a *1* porque la inclusión salio bien. Se debe prestar atención a la diferencia entre los dos ejemplos anteriores. El primero usa **return()** dentro del fichero incluido y el segundo no. Otras maneras de incluir ficheros en variables es con **fopen()**, **file()** ó usando **include()** con Funciones de control de salida.

6.15. require_once()

La función **require_once()** incluye y evalua el fichero especificado durante la ejecución del script. Se comporta de manera similar a **require()**, con la única diferencia que si el código ha sido ya incluido, no se volverá a incluir. Consultar la documentación de la función **require()** para obtener más información.

require_once() debería de usarse en casos en los que un mismo fichero puede ser incluido y evaluado más de una vez durante la ejecución de un script, y se quiere estar seguro que se incluye una sola vez para evitar problemas con redefiniciones de funciones, valores de funciones, etc.

> ***Nota:*** *El comportamiento de de **require_once()** y **include_once()** puede que no sea el esperado en sistemas operativos en los que mayúsculas y minúsculas se traten igual (como en Windows)*

```
<?php
require_once("a.php"); // esto incluye a.php
require_once("A.php"); // esto no incluye a.php de nuevo!
?>
```

6.16. include_once()

La función **include_once()** incluye y evalua el fichero especificado durante la ejecución del script. Se comporta de manera similar a **include()**, con la única diferencia que si el código ha sido ya incluido, no se volverá a incluir.

include_once() debería de usarse en casos en los que, un mismo fichero puede ser incluido y evaluado más de una vez durante la ejecución de un script, y se quiere estar seguro que se incluye una sola vez para evitar problemas con redefiniciones de funciones, valores de funciones, etc.

> ***Nota:*** *El comportamiento de de* ***include_once()*** *y* ***require_once()*** *puede que no sea el esperado en sistemas operativos en los que mayúsculas y minúsculas se traten igual (como en Windows)*

```
<?php
include_once("a.php"); // esto incluye a.php
include_once("A.php"); // esto no incluye a.php de nuevo!
?>
```

6.17. Sintaxis Alternativa de Estructuras de Control

PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, **if**, **while**, **for**, y **switch**. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:) y cerrar-llave por **endif;**, **endwhile;**, **endfor;**, or **endswitch;**, respectivamente.

```
<?php if ($a==5): ?>
 A es igual a 5
 <?php endif; ?>
```

En el ejemplo de arriba, el bloque HTML "*A es igual 5*" se anida dentro de una sentencia if escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si *$a fuera igual a 5*.

La sintaxis alternativa se aplica a **else** y también a **elseif**. La siguiente es una estructura **if** con **elseif** y **else** en el formato alternativo:

```
<?php
 if ($a == 5):
     print "a es igual a 5";
     print "...";
 elseif ($a == 6):
     print "a es igual a 6";
     print "!!!";
 else:
     print "a no es ni 5 ni 6";
 endif;
?>
```

Como con la sentencia **if**, se pueden agrupar multiples sentencias dentro del mismo bucle **while** encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa:

```
while (expr): sentencia ... endwhile;
```

Los siguientes ejemplos son idénticos, y ambos imprimen números del 1 al 10:

```
<?php
/* ejemplo 1 */
  $i = 1;
 while ($i <= 10) {
     print $i++;  /* el valor impreso sería
                     $i antes del incremento
                     (post-incremento) */
 }

 /* ejemplo 2 */
  $i = 1;
 while ($i <= 10):
     print $i;
     $i++;
 endwhile;
?>
```

PHP también soporta la "sintaxis de dos puntos" alternativa para bucles **for**.

```
for (expr1; expr2; expr3): sentencia; ...; endfor;
```

La sintaxis alternativa para las estructuras de control está también soportada con **switch**.

```
<?php
switch ($i):
     case 0:
         print "i es igual 0";
         break;
     case 1:
         print "i es igual a 1";
         break;
     case 2:
         print "i es igual a 2";
         break;
     default:
         print "i no es igual a 0, 1 o 2";
 endswitch;
?>
```

7. Funciones

- *Funciones definidas por el usuario*
- *Parámetros de las funciones*
- *Devolviendo valores*
- *Funciones variables*
- *Funciones de tratamiento de cadenas de caracteres*

7.1. Funciones definidas por el usuario

Las funciones de usuario son bloques de código que pueden ser utilizados en cualquier momento dentro del código PHP.

Se define una función con la sentencia *function* seguida del nombre de la función y de las líneas de instrucciones. Estas últimas deben ir entre las llaves *{...}*. Si la función tiene que devolver un valor a la línea de código donde se invocó, como última instrucción hay que poner la orden *return* seguida del valor que se devuelve. Si no se devuelve un valor, sino que sólo se ejecutan instrucciones dentro del código de la propia función, puede prescindirse de la sentencia *return.*

Las funciones, en la mayoría de los casos, tienen parámetros o argumentos entre paréntesis, a través de los cuales se pasan los datos con los que hay que realizar operaciones.

Una función se puede definir con la siguiente sintaxis:

```
<?php
function foo ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Función de ejemplo.\n";
    return $retval;
}
?>
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funciones y definiciones de clases.

En PHP3, las funciones deben definirse antes de que se referencien. A partir de PHP4 no existe tal requerimiento. Excepto cuando una función es definida condicionalmente como en los ejemplos siguientes.

Cuando una función es definida condicionalmente como se puede ver en estos dos ejemplos, su definición debe ser procesada *antes* que sea llamada.

```
<?php

$makefoo = true;

/* No se puede llamar a foo() desde aquí porque todavía no existe, pero
si se puede llamar a bar() */

bar();

if ($makefoo) {
  function foo ()
```

```
  {
    echo "No existo hasta que la ejecución del programa llegue hasta
mi.\n";
  }
}

/* Ahora se puede llamar con seguridad a foo()
   porque $makefoo se a evaluado a true */

if ($makefoo) foo();

function bar()
{
  echo "Existo desde que el programa comienza.\n";
}

?>
```

```
<?php
function foo()
{
  function bar()
  {
    echo "No existo hasta que se llame a foo().\n";
  }
}

/* No se puede llamar a bar() todavía
   porque no existe. */

foo();

/* Ahora se puede llamar a bar(),
   porque el procesamiento de foo() la ha hecho accesible. */

bar();

?>
```

PHP no soporta la redefinición de funciones previamente declaradas.

> ***Nota:*** *Los nombres de funciones se pueden llamar con mayusculas o minusculas, aunque es una buena costumbre el llamar a las funciones tal y como aparecen en su definición.*

PHP3 no soporta un número variable de parámetros, aunque sí soporta parámetros por defecto. A partir de PHP4 se soporta ambos.

7.2. Parámetros de las funciones

La información puede suministrarse a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.

PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto. Las listas de longitud variable de parámetros sólo están soportadas en PHP4 y posteriores. Un efecto similar puede conseguirse en PHP3 pasando un array de parámetros a la función:

```
<?php
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

7.3. Pasar parámetros por referencia

Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si deseas permitir a una función modificar sus parámetros, debes pasarlos por referencia.

Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un ampersand (&) al nombre del parámetro en la definición de la función:

```
<?php
function add_some_extra(&$string)
{
    $string .= ' y algo más.';
}
$str = 'Esto es una cadena, ';
add_some_extra($str);
echo $str;    // Saca 'Esto es una cadena, y algo más.'
?>
```

7.4. Parámetros por defecto

Una función puede definir valores por defecto para los parámetros escalares estilo C++:

```
<?php
function makecoffee ($type = "cappucino")
{
    return "Hacer una taza de $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
?>
```

La salida del fragmento anterior es:

```
Hacer una taza de cappucino.
Hacer una taza de espresso.
```

El valor por defecto tiene que ser una expresión constante, y no una variable, un miembro de una clase ó una llamada a una función.

Destacar que cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto; de otra manera las cosas no funcionarán de la forma esperada. Considera el siguiente fragmento de código donde se usa incorrectamente parámetros por defecto en funciones:

```
<?php
function makeyogurt ($type = "acidophilus", $flavour)
{
    return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora");   // No funcionará de la manera esperada
?>
```

La salida del ejemplo anterior es:

```
Warning: Missing argument 2 in call to makeyogurt()called in
C:\WAMP\www\Capitulo_07\apar74_02.php on line 6 and defined in
C:\WAMP\www\Capitulo_07\apar74_02.php on line 2
```

Y ahora, compárese con:

```
<?php
function makeyogurt ($flavour, $type = "acidophilus")
{
    return "Haciendo un bol de $type $flavour.\n";
}

echo makeyogurt ("mora");   // funciona como se esperaba
?>
```

La salida de este ejemplo es:

```
Haciendo un bol de acidophilus mora.
```

7.5. Lista de longitud variable de parámetros

A partir de PHP4 se soportan las listas de longitud variable de parámetros en las funciones definidas por el usuario.

No se necesita ninguna sintaxis especial, y las listas de parámetros pueden ser escritas en la llamada a la función y se comportarán de la manera esperada.

Es realmente fácil, usando las funciones proporcionadas por PHP **func_num_args()**, **func_get_arg()**, y **func_get_args()**:

- **Func_num_args():** Devuelve el número de parámetros pasados a una función de parámetros variable. (El índice de un parámetro dentro de una función va desde 0 hasta n-1, donde n representa el número de parámetros de la función)
- **Func_get_arg():** Devuelve el valor de un parámetro en particular. Por ejemplo, para retornar el valor del primer parámetro de una función de parámetros variable, podemos escribir:

```
$arg1 = func_get_arg(0);
```

- **Func_get_args():** Retorna todos los parámetros pasados a la función en forma de array.

Un ejemplo del uso de funciones de parámetros variable es:

```
<?php
   showtitles('Titulo1');
   showtitles('Titulo1', 'Titulo2');
   showtitles('Titulo1', 'Titulo2', 'Titulo3');

   function showtitles()
   {
```

```
        for($i=0;$i<func_num_args();$i++)
           echo (func_get_arg($i) . "\n");
    }
?>
```

7.6. Devolviendo valores

Los valores se retornan usando la instrucción opcional **return**. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

```
<?php
function square ($num)
{
    return $num * $num;
}
echo square (4);   // saca '16'.
?>
```

No puedes devolver múltiples valores desde una función, pero un efecto similar se puede conseguir devolviendo un array.

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

Para retornar una referencia desde una función, se tiene que usar el operador de referencias & tanto en la declaración de la función como en la asignación del valor de retorno a una variable;

```
<?php
function &returns_reference()
{
    return $someref;
}

$newref = &returns_reference();
?>
```

7.7. Funciones variables

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que la evaluación de la variable, e intentará ejecutarla. Entre otras cosas, esto te permite implementar retrollamadas (callbacks), tablas de funciones y demás.

Las funciones variables no funcionarán con construcciones del lenguaje, tal como **echo()**, **print()**, **unset()**, **isset()**, **empty()**, **include()**, **require()** y derivados. Se necesitará usar una función propia para utilizar cualquiera de estos constructores como funciones variables.

```
<?php
function foo()
{
    echo "En foo()<br>\n";
}

function bar($arg = '')
{
    echo "En bar(); el argumento es '$arg'.<br>\n";
}

function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // Llama a foo()

$func = 'bar';
$func('test');  // Llama a bar()

$func = 'echoit';
$func('test');  // Llama a echoit()
?>
```

También se puede llamar a un método de un objeto usando la característica variable de las funciones.

```
<?php
class Foo
{
    function Var()
    {
        $name = 'Bar';
        $this->$name(); // Esto llama al método Bar()
    }

    function Bar()
    {
        echo "Esto es Bar";
    }
}

$foo = new Foo();
$funcname = "Var";
$foo->$funcname();  // Esto llama a $foo->Var()

?>
```

7.8. Funciones de tratamiento de cadenas de caracteres

En este apartado vamos a abordar con mayor detalle las diferentes operaciones que se pueden llevar a cabo con este tipo de datos estudiando las principales funciones que permiten hacerlo.

Algunas de las funciones más usadas para mostrar información son:

7.8.1. echo

Ya hemos visto varios ejemplos de este comando. Tiene esta sintaxis:

```
echo argumento
```

donde la información que se muestra (*argumento*) puede ser una cadena de caracteres entre comillas o una variable. También puede usarse la sintaxis

```
echo (argumento1, argumento2,...)
```

Si el argumento es una cadena entre comillas dobles, debemos tener en cuenta que las variables incluidas en ese texto se expanden, es decir, muestran su contenido, no su nombre. Sin embargo, las funciones no se expanden. ¿Cómo podemos conseguir que las variables no se expandan y las funciones sí lo hagan? Veamos el ejemplo siguiente donde mostramos en letras mayúsculas el contenido de una variable:

```
$x = "el amor es libre";
echo "La variable $x contiene el texto strtoupper($x) en
      letras mayúsculas.";
```

Ejecutando estas dos instrucciones aparece en la pantalla:

```
La variable el amor es libre contiene el texto strtoupper(el amor es
libre) en letras mayúsculas.
```

Observamos que, aunque esté dentro de una cadena, la variable $x se ha expandido, es decir, ha mostrado su texto; en cambio, la función strtoupper (*string to upper*, convertir el texto de la cadena en letras mayúsculas) no se ha expandido. Pero nosotros queremos que en la página *web* aparezca:

```
La variable $x contiene el texto EL AMOR ES LIBRE en letras mayúsculas.
```

Para conseguirlo, hay que utilizar esta otra sintaxis:

```
$x = “el amor es libre”;
echo “La variable \$x contiene el texto”.strtoupper($x). “en letras
mayúsculas.”;
```

Si nos fijamos en esta última sintaxis, podemos observar que hemos utilizado la barra hacia la izquierda (\) delante de la variable, para que no expanda su contenido. Además, hemos sacado la función de las comillas y la hemos concatenado con el texto anterior y posterior usando el operador de concatenación (**.**).

7.8.2. print

Esta orden es prácticamente igual que **echo**, hasta el punto de que se pueden intercambiar sin problemas.

Veamos un ejemplo en que se usa la orden **print** en lugar de **echo**.

```
<HTML>
<HEAD><TITLE>Curso de PHP</TITLE></HEAD>
<BODY>

<? // Aquí se inicia el código PHP.

  print "<H1><CENTER>Uso de la orden print. Funciona como echo.</H1>
            </CENTER><H2>";
    /* Mostramos el texto anterior con un tamaño de letra grande
       y centrado.*/
  $pares=0;
```

```
    /* Abrimos un bucle para repetir la operación 20 veces.
    En la Unidad 2 se estudia esta estructura de control.*/
    for (;$pares<40;)
    {
      print " -- $pares "; // Observa que la variable se expande.
      if ($pares==40)      // Si la variable $pares vale 40...
        break;           // Salimos del bucle.
      $pares+=2;      // Sumamos 2 al valor actual de la variable.
     }

     print "</H2><P><H3>     Relación de los 20
                 primeros números pares.</H3>";
       /* Mostramos el texto anterior con un tamaño de letra pequeño.
       Como HTML sólo respeta un espacio en blanco, con el comando
         hacemos que la frase deje 5 espacios a la izquierda. */

     /* Aquí acaba el código PHP. No podemos poner este comentario a
     continuación del signo ?>, pues estaría fuera del código PHP.*/
  ?>

  </BODY>
  </HTML>
```

7.8.3. printf

Es una función que permite mostrar información formateada según los valores que aparezcan en una cadena de formato integrada por caracteres fijos y por una directiva %.

Su sintaxis es:

```
printf ("cadena de formato", argumento1, argumento2,...)
```

En los argumentos se incluye la información que se quiere mostrar, como con **echo** o con **print**. En la cadena de formato, entre comillas, puede ponerse un texto, que aparece tal cual lo escribamos, con expansión de las variables si no usamos el signo \, y el valor de una variable según se establezca en la directiva %.

Veamos en un ejemplo sencillo cómo se comporta esta función:

```
<HTML>
<HEAD></HEAD>
<BODY>
<?
    $i=15;
    printf ("Valor de la variable \$i=    %d \n", $i);
    echo "<P>Nueva línea ";
?>
</BODY>
</HTML>
```

Si ejecutamos este código, en la pantalla aparece la página web siguiente:

```
Valor de la variable $i= 15

Nueva línea
```

Si editamos el código HTML que ha recibido el usuario, vemos cómo es:

```
<HTML>
<HEAD></HEAD>
<BODY>
```

```
Valor de la variable $i=    15
<P>Nueva línea</BODY>
</HTML>
```

En este sencillo ejemplo podemos apreciar varias cosas que debemos conocer. Son las siguientes:

- El segundo argumento de la función printf(), que es $i, aparece con la información que tiene esta variable.
- El primer argumento determina el formato de salida de esa información y está integrado por un texto literal (*Valor de la variable \$i=*), que se muestra tal como lo hemos escrito, por una directiva (*%d*), que establece que el valor de *$i* ha de aparecer como número decimal (en base 10) entero. Y así lo hace.

En este ejemplo hay que observar también varias cosas que debemos tener en cuenta:

1. Conviene recordar que en el texto literal hemos tenido que utilizar el signo \ delante de *$i* para que la variable no se expandiera. No confundirlo con el comando \n que sirve para que se produzca un salto de línea.

2. La directiva *%d* lleva detrás el comando del código PHP \n. Puede llevar también otro literal. El comando \n sirve para que, una vez mostrado el resultado de la variable *$i*, se produzca un salto de línea dentro del código HTML, como ya hemos dicho.

3. En el código fuente PHP hemos dejado hasta cinco espacios en blanco después del primer literal y antes de la directiva, pero en la página *web* sólo aparece uno. Hay que tener en cuenta que HTML sólo muestra un espacio, no todos los que haya repetidos en el código.

Dentro de la cadena de formato nos interesa ver cómo se configura una directiva y qué significa cada uno de los caracteres que la integran.

Toda directiva debe iniciarse por un signo **%**. Si se quiere anular el efecto de % de forma que la directiva se vea como un literal, hay que poner delante otro signo %. Por ejemplo, la orden

```
printf("Valor de la variable \$i= %%d \n", $i);
```

muestra el texto *Valor de la variable $i= %d* sin el valor de $i ya que no hemos precisado su formato.

Así pues, si queremos que en la página web aparezca la frase

```
La variable $i con la directiva %d aparece como 15
```

debemos escribir la función de esta forma:

```
printf("La variable \$i con la directiva %%d aparece como %d \n", $i);
```

La directiva % puede estar integrada hasta por cinco elementos:

1. Un carácter de relleno, para que ocupe las posiciones vacías cuando lo que ha de mostrarse no completa todas. Si no se pone, se supone que las posiciones vacías han de ser espacios en blanco. En el caso de números, puede indicarse que se rellenen con ceros o con otro carácter (escribir entre comillas simples) como primer integrante de la directiva.
2. El carácter opcional '-', para alinear un número por la izquierda.
3. El número mínimo de posiciones que ha de ocupar la salida. Si su contenido no ocupa tantas posiciones, los espacios vacíos se completan con el carácter que se haya indicado en el componente primero. Por ejemplo, la instrucción

```
printf("La variable \$i con la directiva %%05d aparece como %05d \n", $i);
```

muestra en la página el texto

```
La variable $i con la directiva %05d aparece como 00015
```

donde el *0* primero indica el carácter de relleno, el *5* es el número de posiciones que debe ocupar la salida y la *d*, como veremos, indica que ha de mostrarse en formato decimal (base 10).

Cuando la salida es un número, este guarismo sólo se refiere a la parte entera, no a la decimal, cuya longitud de salida se fija con el elemento siguiente, separando ambos elementos por un punto.

4. El número de decimales que deben mostrarse cuando los haya, es decir, cuando trabajemos con números de coma flotante. Este número debe ir separado del número entero anterior por un punto (**.**).

Por ejemplo, si la varible *$i* vale *11.3456*, la instrucción

```
printf("La variable \$i con la directiva %%09.5f aparece como %09.5f \n",
$i);
```

muestra en la página el texto

```
La variable $i con la directiva %09.5f aparece como 011.34560
```

5. Especificación del tipo de salida. Este último es el más importante, hasta el punto de que la mayoría de las veces es el único que sigue al signo de directiva %.

Puede ser una sola de las letras que aparecen en la tabla siguiente:

LETRA	FORMATO DE SALIDA. LA VARIABLE $I CONTIENE 15.
b	La salida numérica aparece en formato binario (base 2). Orden: printf("\$i con %%b aparece como %b", $i); Muestra: $i con %b aparece como 1111
c	La salida completa aparece como un carácter que se indica detrás de la c. Orden: printf("\$i con %%cZ aparece como %cZ", $i); Muestra: $i con %CZ aparece como Z
d	La salida numérica aparece en formato decimal (base 10). Orden: printf("\$i con %%d aparece como %d", $i); Muestra: $i con %b aparece como 15
f	La salida numérica aparece en formato de coma flotante.

	Orden: printf("\$i con %%f aparece como %f", $i); Muestra: $i con %b aparece como 15.000000
o	La salida numérica aparece en formato octal (base 8). Orden: printf("\$i con %%o aparece como %o", $i); Muestra: $i con %b aparece como 17
s	La salida, aunque sea numérica, aparece como una cadena. Orden: printf("\$i con %%s aparece como %s cadena con números", $i); Muestra: $i con %s aparece como 15 cadena con números
x	La salida numérica aparece en formato hexadecimal (base 16) con las letras en minúscula. Orden: printf("\$i con %%04x aparece como %04x", $i); Muestra: $i con %04x aparece como 000f
X	La salida numérica aparece en formato hexadecimal (base 16) con las letras en mayúscula. Orden: printf("\$i con %%04X aparece como %04X", $i); Muestra: $i con %04x aparece como 000F

En el ejemplo siguiente se muestra cómo usar la orden **printf()**.

```
<HTML>
<HEAD><TITLE>Curso de PHP</TITLE></HEAD>
<BODY>

<CENTER>
<H3>TABLA DE LOS 20 PRIMEROS NÚMEROS EN DIFERENTES
        BASES</H3>
<TABLE BORDER=2>
    <TH>   Decimal  </TH>
    <TH>   Binario   </TH>
    <TH>   Octal   </TH>
    <TH>   Hexadecimal  </TH>

<?   // Aquí se inicia el código PHP.

  echo "<BR>"; // Salto de línea para dejar un espacio entre
              // el título y la tabla.

  /* Abrimos un bucle para repetir la operación 20 veces.
   En la Unidad 2 se estudia esta estructura de control.*/
  for ($a=0;$a<21;$a++)
  {
      echo "<TR>";   // Nueva fila de tabla.
      printf("<TD> %02d</TD><TD> %06b</TD>
                <TD> %02o</TD><TD> %02X</TD>"
                ,$a,$a,$a,$a);
          /* Observa que en la cadena de formato, entre paréntesis,
          hemos incluido los códigos HTML (<TD> ...</TD>)
          y la directiva (%02d, %06b, %02o y %02X) de cada una
          de las formas de presentar el valor de la varible $a. */

      print "</TR>";
  }

   /* Aquí acaba el código PHP. No podemos poner este comentario a
   continuación del signo ?>, pues estaría fuera del código PHP.*/
?>

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

7.8.4. sprintf

Esta función nos permite dar formato a una cadena de texto exactamente igual que con **printf**, y guardar, además el resultado en una variable, para utilizarlo posteriormente.

Su sintaxis es la siguiente:

```
$cadena=sprintf(“cadena de formato”, argumento 1, argumento 2,...)
```

Siguiendo el último ejemplo de la tabla anterior, podíamos escribir:

```
<HTML>
<HEAD></HEAD>
<BODY>
<?
    $i=15;
    $cadena=sprintf("\$i con %%04X aparece como %04X", $i);
    echo $cadena;
    echo "<P>Nueva línea";
?>
</BODY>
</HTML>
```

El resultado es la página *web* siguiente:

```
$i con %04X aparece como 000F

Nueva línea
```

Aunque la cadena de formato parezca inicialmente algo complicada, con la práctica resultará de gran utilidad.

Una vez vistas las funciones necesarias para mostrar información en forma de cadena de caracteres veamos algunas de las funciones de cadenas más frecuentes y necesarias. Por otra parte, en el **Capítulo CLXI. Funciones de Cadenas** del **Manual de PHP** aparecen todas las funciones de cadena de caracteres. En este apartado nos vamos a limitar a explicar brevemente las más frecuentes y necesarias.

7.8.5. Adaptar las cadenas al contexto

Al mezclarse el código HTML y PHP es preciso ocasionalmente que el intérprete sepa distinguir los signos propios de cada lenguaje que puedan aparecer en el interior de una cadena. Las funciones que permiten adaptar el contenido de una cadena al contexto de cada código son las siguientes:

La función **addSlashes()** devuelve una cadena con barras invertidas delante de los caracteres que necesitan marcarse en consultas de bases de datos y en otras operaciones en las que intervienen esos caracteres. Nos estamos refiriendo a la comilla simple ('), a la comilla doble ("), a la barra invertida (\) y a NUL (byte nulo).

La forma de proteger estos caracteres, para que no sean interpretados por el Shell o por PHP como caracteres significativos, es poner una barra invertida (*Slash*) delante de los mismos.

Por ejemplo, si a la variable *`$consulta`* le asignamos el valor "*WHERE nom = ‘Ana’*", las comillas simples interiores serán interpretadas como caracteres significativos, pero necesitamos que sean interpretados como comillas. Así pues, con la función **addSlashes()** lo conseguimos de esta forma:

```
$consulta="WHERE nom = 'Ana'";
```

```
// Así la variable $consulta contendrá WHERE nom = ‘Ana’
$consulta=addSlashes($consulta);
// Así la variable $consulta contendrá WHERE nom = \‘Ana\’
```

La función **stripSlashes()** devuelve una cadena sin barras invertidas delante de la comilla simple ('), la comilla doble ("), la barra invertida (\) y el valor NUL. Esta función, pues, realiza la función inversa de la función anterior.

Siguiendo con el ejemplo anterior podemos usarla de esta forma:

```
$consulta="WHERE nom = 'Ana'";
// Así la variable $consulta contendrá WHERE nom = 'Ana'
$consulta=addSlashes($consulta);
// Así la variable $consulta contendrá WHERE nom = \'Ana\'
$consulta=stripSlashes($consulta);
// Así la variable $consulta contendrá WHERE nom = 'Ana'
```

La función **urlencode()** devuelve una cadena en la que todos los caracteres no alfanuméricos, excepto -, _, y., son reemplazados por un signo de porcentaje (%) seguido por dos dígitos hexadecimales. Los espacios son sustituidos por signos positivos (+). Es conveniente usar esta función para escribir una cadena de texto que va a ser usada dentro del código HTML como parte de una consulta de una URL. También es una forma adecuada de pasar variables a la página siguiente. Lo más frecuente es que la usemos para que HTML respete los espacios en blanco que haya dentro del texto.

Veamos en un ejemplo cómo se producen las conversiones:

```
$b='<A HREF = _ -  "Poñw?.jura 1 ) # & //';
echo $b."<P>";
/* Aquí se muestra la misma cadena que hemos
   escrito entre comillas simples. */
echo urlencode($b);
/* Aquí se muestra la cadena %3CA+HREF+%3D+_+-
++%22Po%F1w%3F.jura+1+%29+%23+%26+%2F%2F  */
```

Conviene observar que los espacios se representan con el signo +, las letras, excepto la ñ, y los números se dejan igual, y los demás signos, como <, =,), etcétera, son sustituidos por el signo % seguido de dos dígitos en base hexadecimal.

La función **urldecode()** realiza la operación inversa de la anterior. Por ejemplo, si escribimos las instrucciones

```
$c='SELECT+ALL+FROM+tabla+WHERE+nombre%3D%22JUANA%22';
$d=urldecode($c);
echo $d."<P>";
```

la página muestra la cadena original

```
SELECT ALL FROM tabla WHERE nombre="JUANA"
```

Las funciones **rawurlencode()** y **rawurldecode()** (código puro para URL) se comportan prácticamente igual que las dos anteriores, con la diferencia de que éstas no convierten en código HTML las letras, los números y el carácter de subrayado (_), pero sí codifican en el mismo los signos -, y. Usando las primeras se consiguen efectos más amplios, por lo que recomendamos su uso preferente.

La función **nl2br()** se utiliza también para dar formato a las salidas en HTML. Esta función transforma los códigos de salto de línea de los ficheros de texto en códigos

de salto de línea propios de HTML conservando los anteriores. Se utiliza principalmente para organizar el texto introducido en TEXTAREA de un formulario. Cambia, pues, la orden \n (salto de línea en un fichero de texto) en
 (salto de línea en código XHTML) produciendo un salto de línea real, tanto en el código HTML devuelto al interpretarse el código PHP como en la página web que se muestra.

Por ejemplo, si escribimos las siguientes instrucciones

```
$frase="Primero \n Segundo \n Tercero";
echo "<br> \n";
echo "Sin aplicar saldo de línea HTML: \"$frase\"";
echo "<br> \n";
$frase=nl2br($frase);
echo "Aplicando salto de línea HTML: \"$frase\"";
```

se devuelve al siguiente código HTML

```
<HTML>
<HEAD></HEAD>
<BODY>
<br>
Sin aplicar salto de línea HTML: "Primero
 Segundo
 Tercero"<br>
Aplicando salto de línea HTML: "Primero <br />
 Segundo <br />
 Tercero"
</BODY>
</HTML>
```

y se genera la siguiente página:

```
Sin aplicar saldo de línea HTML: "Primero Segundo Tercero"
Aplicando salto de línea HTML: "Primero
Segundo
Tercero"
```

Se puede observar que los saltos de línea del fichero de texto *n* mediante la función **nl2br()** han sido sustituidos en el código XHTML por saltos de línea <*br /*>.

7.8.6. Limpiar cadenas de caracteres

La función **chop()** elimina los espacios en blanco que haya al final de una cadena de caracteres, incluyendo los códigos de fin de línea si los hubiere.

Por ejemplo, las instrucciones

```
$f="La ciencia es un conjunto de verdades    \n  ";
$g=chop($f);
echo "La variable \$f contiene ".strlen($f)."
      caracteres de los que ".(strlen($f)-strlen($g)).
     "son espacios en blanco o saltos de línea.<P>";
echo "En cambio, si usamos la función chop()
      la variable \$f contiene sólo ".strlen($g).
     "caracteres.<P>";
```

generan la página siguiente:

```
La variable $f contiene 44 caracteres de los que 7 son espacios en
blanco o saltos de línea.
En cambio, si usamos la función chop() la variable $f contiene sólo 37
caracteres.
```

Algo más adelante se verá que la función **strlen()** devuelve el número de caracteres que tiene una cadena.

La función **ltrim()** elimina los espacios en blanco que haya al principio de una cadena de caracteres.

Por ejemplo, las instrucciones

```
$h="     La ciencia es un conjunto de verdades";
$i=ltrim($h);
echo "La variable \$h contiene ".strlen($h)." caracteres
      de los que ".(strlen($h)-strlen($i))."son espacios
      en blanco.<P>";
echo "En cambio, si usamos la función ltrim() la variable
      \$i contiene sólo ".strlen($i)." caracteres.<P>
      Ha perdido los espacios en blanco iniciales.<P>";
```

generan la página siguiente:

```
La variable $h contiene 42 caracteres de los que 5 son espacios en
blanco.
En cambio, si usamos la función ltrim() la variable $h contiene sólo 37
caracteres.
Ha perdido los espacios en blanco iniciales.
```

La función **trim()** es una combinación de las dos anteriores: elimina los espacios en blanco que haya al principio y al final de una cadena.

Por ejemplo, si *$limpia* contiene " *casa* ", la instrucción *echo trim($limpia)* devuelve "*casa*".

La función **strip_tags()** elimina las etiquetas HTML de una cadena. Por ejemplo, las instrucciones siguientes

```
$quita_html="<H1>Texto grande</H1> <B>Negrita</B><P>";
echo $quita_html."<P>";
echo strip_tags($quita_html);
```

generan esta página

```
Texto grande
Negrita
Texto grande Negrita
```

Puede observarse que en el primer **echo** se ejecutan los códigos HTML. En cambio, en el segundo, al eliminarse estos controles con **strip_tags()**, se muestra el texto sin ellos.

7.8.7. Letras mayúsculas y letras minúsculas

Algunas funciones permiten convertir todos o sólo algunos caracteres de una cadena en mayúsculas o minúsculas según nos convenga. Son muy fáciles de usar. Resumimos las más importantes.

1. Función **strtoupper()*:*** devuelve la misma cadena convirtiendo todos sus caracteres en mayúsculas.

Ejemplo: si *$a* contiene "*Pepe*", ***echo strtoupper($a);*** devuelve "*PEPE*".

2. Función **strtolower()*:*** devuelve la misma cadena convirtiendo todos sus caracteres en minúsculas.

Ejemplo: si *$M* contiene "*PEPe*", ***echo strtolower($M);*** devuelve "*pepe*".

3. Función **ucfirst()*:*** devuelve la misma cadena convirtiendo su primer carácter en mayúscula si es una letra.

Ejemplo: si *$M1* contiene "*diosa*", ***echo ucfirst($M1);*** devuelve "*Diosa*".

Para que esta función tenga en cuenta el alfabeto completo español en mayúsculas, sobre todo si las letras tienen tilde, como "*Águila*", hay que establecer su configuración previamente con la función *setlocale (LC_ALL,"spanish");*. Estos valores son los formatos de hora, moneda, alfabeto, separador de decimales, etcétera.

4. Función **ucwords()*:*** devuelve la misma cadena convirtiendo el primer carácter de todas las palabras en mayúscula.

Ejemplo: si *$M_pal* contiene "*diosa de la antigüedad egipcia*", la instrucción ***echo ucwords($M_pal);*** devuelve "*Diosa De La Antigüedad Egipcia*".

7.8.8. Conocer la longitud de una cadena

Función **strlen()*:*** devuelve la longitud de una cadena expresada en valor numérico entero. Cuenta todas las posiciones, aunque sean espacios en blanco. Ya hemos visto cómo se usa en las funciones **`chop()`**, **`ltrim()`** y **`trim()`**.

Ejemplo: si *$largo* contiene "*diosa de la antigüedad egipcia*", la instrucción ***echo strlen($largo);*** devuelve el valor numérico *30*.

7.8.9. Repetir una cadena

Función **str_repeat():** devuelve la misma cadena que toma como primer argumento repetida tantas veces como se indique en el segundo argumento.

Ejemplo: si $*repe* contiene "*diosa*<P>", ***echo str_repeat($repe,3);*** devuelve la cadena "*diosa*" repetida tres veces en distintas líneas.

7.8.10. Modificar algunos caracteres de una cadena

Función **strtr()*:*** devuelve la misma cadena sustituyendo los caracteres de la misma que se indiquen en el segundo argumento por los que se indiquen en el tercer argumento haciéndolos equivaler uno a uno.

Ejemplo: si *$cambia* contiene "*Los días perdidos en verano*" y necesitamos sustituir las *a*s por *u*s y las *o*s por *a*s, podemos escribir la sentencia siguiente: ***echo strtr($cambia,"ao","ua");*** La función devolverá la cadena "*Las díus perdidas en veruna*".

7.8.11. Buscar dentro de las cadenas

Hay funciones que permiten buscar dentro de una cadena. No se usan con frecuencia, por lo cual quien lo necesite puede obtener información en el manual oficial de PHP sobre las mismas. Nos limitamos a enumerar las principales y a indicar para qué sirven:

1. **strcspn()*:*** devuelve el número de caracteres del primer segmento de la primera cadena (primer argumento) que no contiene caracteres de la segunda cadena (segundo argumento).
2. **strspn()*:*** devuelve el número de caracteres del primer segmento de la cadena pasada como primer argumento que aparecen en la cadena pasada como segundo argumento.
3. **strpos()*:*** devuelve el número de la posición del primer carácter de la primera cadena (primer argumento) que coincida con la segunda cadena completa (segundo argumento). Se cuenta desde 0 en adelante. Admite un tercer argumento, que es el número a partir del cual debe buscarse el texto de la segunda cadena en la primera.
4. **strrpos()*:*** devuelve el número de la posición del último carácter de la primera cadena (primer argumento) que coincida con la segunda cadena completa (segundo argumento). Se cuenta desde 0 en adelante.
5. **strrchr()*:*** devuelve la subcadena que comprende el fragmento de la primera cadena (primer argumento) que va desde la última aparición del carácter pasado como segundo argumento hasta el final de la primera cadena.
6. **strstr()*:*** devuelve una subcadena de la cadena pasada como primer argumento que comprende desde la primera aparición de la cadena pasada como segundo argumento hasta el final de la primera cadena.

7.8.12. Operaciones con subcadenas

La función **substr()** devuelve una subcadena de la cadena que se pasa como primer argumento. Esta subcadena contiene los caracteres de la cadena desde la posición que se indica en el segundo argumento, que debe ser un número entero, hasta el número que se indique en el tercer argumento.

Por ejemplo,

```
echo substr("Memorias de África",3,4)
```

devuelve la subcadena "*oria*". Conviene observar que la cuenta empieza desde 0.

La sintaxis de esta función puede ser variada:

- El segundo argumento indica la posición a partir de la cual se empieza a extraer la subcadena. Si el argumento es positivo la extracción empieza en la posición indicada contando a partir del primer carácter de la cadena. Sin embargo si el argumento es negativo se cuenta a partir del último carácter de la cadena.

```
echo substr("Hello”,1);  // retorna ello
echo substr("Hello”,2);  // retorna llo
echo substr("Hello”,-1);  // retorna o
echo substr("Hello”,-2);  // retorna lo
```

- El tercer argumento indica la longitud de la subcadena extraída. Si el argumento es positivo se recupera el número de caracteres especificado a partir de la posición de inicio. Sin embargo si es negativo se recuperan todos los caracteres a partir de la posición de inicio menos los especificados en longitud.

```
echo substr("ABCDE",1,-1)  // retorna BCD
echo substr("ABCDE",2,2)  // retorna CD
echo substr("ABCDE",-3,1)  // retorna C
echo substr("ABCDE",-3,-1)  // retorna CD
```

La función **substr_replace()** sustituye una subcadena de la cadena que se pasa como primer argumento con la subcadena que se pasa como segundo argumento desde la posición de inicio que se pasa como tercer argumento tantos caracteres como se indique en el cuarto argumento. Es una función existente a partir de PHP 4.

Por ejemplo,

```
echo substr_replace("Memorias de África","en",9,2)
```

devuelve la cadena como "*Memorias en África*".

En la sintaxis de esta función también pueden darse las circunstancias explicadas en la funcion **substr**.

La función **strtok()** separa una cadena de texto (primer argumento) en bloques según el separador indicado en el segundo argumento.

La función **parse_str()** trata una cadena de caracteres como si su información se pasase a otra página mediante la URL. Conviene observar que esa cadena debe tener el formato adecuado. Por ejemplo, las sentencias

```
$variables="nom=Nacho&ape1=Roa&ape2=Bastos",
parse_str($variables);
```

define las variables *$nom*, *$ape1* y *$ape2*, y les asigna los valores "*Nacho*", "*Roa*" y "*Bastos*", respectivamente. Así puede pasarlas a otra página.

7.8.13. Invertir el texto de una cadena

La función **strrev()** recibe una cadena de caracteres y la devuelve al revés.

Por ejemplo, la sentencia ***echo strrev("paseo");*** devuelve "*oesap*".

7.8.14. Comparar cadenas

La función **strcasecmp()** compara dos cadenas pasadas como argumentos sin distinguir entre letras mayúsculas y minúsculas y devuelve un número entero, que es un cero si son iguales, un número mayor que cero si la primera es mayor que la segunda y un número menor que cero si la primera es menor que la segunda.

Por ejemplo, la instrucción *strcasecmp("ABC","abc")* devuelve *0*, la instrucción *strcasecmp("CAB","abc")* devuelve *2* y la instrucción *strcasecmp("ABC","cab")* devuelve *-2*.

La función **strcmp()** compara dos cadenas pasadas como argumentos distinguiendo entre letras mayúsculas y minúsculas y devuelve los mismos valores que la función anterior con la salvedad indicada. En el mismo ejemplo anterior, las tres instrucciones siempre devolverían *-1*.

7.8.15. Otras funciones de cadena útiles

La función **chr()** recibe un número entero y devuelve el carácter correspondiente del código ASCII.

```
Por ejemplo, echo chr(65); devuelve la letra A mayúscula.
```

La función **ord()** recibe un carácter del código ASCII y devuelve el número entero correspondiente.

```
Por ejemplo, echo ord("A"); devuelve el número 65.
```

La función **number_format()** permite formatear un número como nos convenga. Recibe un número de coma flotante como primer argumento y devuelve una cadena con un determinado número de decimales fijado en el segundo argumento, separado de la parte entera por un signo especificado en el argumento tercero y un separador de los miles que se fija en el argumento cuarto.

Por ejemplo, si la variable `$numero` contiene el valor *1234.5678* (el punto es el separador decimal), la instrucción `echo number_format($numero,3,",",".");` devuelve la cadena *1.234,567*.

Es importante saber cómo están definidos los valores de configuración local de PHP para los puntos decimales. En la función siguiente se explica este asunto.

La función **setlocale()** establece los valores de distintas categorías que pueden ser diferentes para distintas configuraciones locales, como la conversión y clasificación de caracteres en los diversos idiomas, los separadores de decimales, el formato de fecha y hora, así como la comparación de cadenas y de monedas.

Por ejemplo, la instrucción `setlocale(LC_ALL,"spanish")` fija todas las categorías disponibles (LC_ALL: conversión y clasificación de caracteres en los diferentes idiomas, separadores de decimales y formato de fecha y hora) para el entorno español, que en la mayoría de las categorías es igual al europeo.

7.9. Funciones de tratamiento de arrays

Hemos visto que una variable tiene un nombre que debe iniciarse por el signo dólar (*$*) y un valor de uno de los tipos que soporta PHP. Pero en una variable sólo podemos introducir un valor. ¿Hay alguna forma de asignar más de un valor a un identificador? Sí, pero hay que usar **arrays**, que son conjuntos de datos de diferentes tipos que se asignan a un solo nombre.

Un **array** o **matriz** se caracteriza por tres cosas:

- Está integrado por múltiples elementos (*elements*).

• Tiene un índice (*key* o *index*) para referirse a cada uno de sus elementos. Este índice, también llamado subíndice, puede ser de tipo numérico entero o de tipo cadena. En PHP el índice del primer elemento tiene el valor 0, si es de tipo numérico entero.
• Cada elemento tiene un contenido (*value*), que puede ser de diferente tipo que el resto de los elementos.

7.9.1. Crear e inicializar una matriz

En PHP, como en la mayoría de los lenguajes informáticos, hay varias formas de crear un matriz y de asignar valores a cada uno de sus elementos.

```
$persona["nombre"] = "Josefa";
$persona["apellido1"] = "Pérez";
$persona["apellido2"] = "Rubio";
$persona["edad"] = 60;
```

En las cuatro instrucciones anteriores hemos creado la matriz *$persona*, que tiene cuatro elementos que se referencian por cuatro índices de tipo cadena. Pero también podíamos haber creado e inicializado esta matriz así:

```
$persona[] = "Josefa";
$persona[] = "Pérez";
$persona[] = "Rubio";
$persona[] = 60;
```

En este segundo caso, PHP asigna los índice *0*, *1*, *2* y *3* a cada elemento secuencialmente. Lo mismo hubiéramos conseguido igualmente con la sintaxis:

```
$persona[0] = "Josefa";
$persona[1] = "Pérez";
$persona[2] = "Rubio";
$persona[3] = 60;
```

aunque habría resultado más largo e innecesario. No es preciso asignar índices que sean valores numéricos enteros consecutivos, si bien es preciso conocer el número o el nombre de los índices para poder, después, referirse a los elementos correspondientes.

Otra forma de crear la misma matriz con índices de cadena es la siguiente:

```
$persona = array("nombre"=> "Josefa","apellido1"=> "Pérez",
                 "apellido2"=> "Rubio","edad"=>60);
```

Como puede verse, esta sintaxis utiliza la palabra reservada **array**, pone entre comillas los índices de cadena y usa el signo => para asignar el valor a los elementos.

Para crear una matriz bidimensional con las calificaciones obtenidas por una alumna en las tres evaluaciones de Matemáticas, Lengua y Dibujo usando índices alfanuméricos, podemos escribir lo siguiente:

```
$notas["Mat"]["Pri"] = "Suficiente";
$notas["Mat"]["Seg"] = "Bien";
$notas["Mat"]["Ter"] = "Notable";
$notas["Len"]["Pri"] = "Sobresaliente";
$notas["Len"]["Seg"] = "Sobresaliente";
$notas["Len"]["Ter"] = "Notable";
$notas["Dib"]["Pri"] = "Notable";
$notas["Dib"]["Seg"] = "Bien";
$notas["Dib"]["Ter"] = "Suficiente";
```

Con la misma filosofía de índices, podíamos haberlo hecho de la forma siguiente:

```
$notas = array("Mat"=>array("Pri"=>"Suficiente",
                            "Seg"=>"Bien",
                            "Ter"=>"Notable"),
               "Len"=>array("Pri"=>"Sobresaliente",
                            "Seg"=>"Sobresaliente",
                            "Ter"=>"Notable"),
               "Dib"=>array("Pri"=>"Notable",
                            "Seg"=>"Bien",
                            "Ter"=>"Suficiente"));
```

Igualmente, podíamos haber puesto las calificaciones en número y usado índices numéricos para las evaluaciones y las asignaturas, de 0 a 2 para ambas. En este caso la matriz *$notas* habría quedado así:

```
 $notas[0][0] = 1;
 $notas[0][1] = 2;
 $notas[0][2] = 3;
 $notas[1][0] = 4;
 $notas[1][1] = 5;
 $notas[1][2] = 6;
 $notas[2][0] = 7;
 $notas[2][1] = 8;
 $notas[2][2] = 9;
```

Es más breve, si bien hay que saber a qué asignatura y evaluación pertenece cada índice. En la sintaxis precedente quedaría así:

```
$notas = array(0=>array(0=>1,1=>2,2=>3),
               1=>array(0=>4,1=>5,2=>6),
               2=>array(0=>7,1=>8,2=>9));
```

Como hemos visto, pues, la función **array()** permite crear e inicializar matrices, si bien no es la única forma de hacerlo.

Su sintaxis es

```
array(lista de elementos separados por comas);
```

En la lista de elementos se pueden poner valores de los diferentes tipos de datos que admite PHP. Además, es posible especificar expresamente un índice numérico o de cadena para referirse a cada uno de sus elementos.

La función **range()** devuelve una matriz que sólo contiene los elementos indicados entre dos números, ambos incluidos.

Su sintaxis es

```
range(número entero del límite inferior, número entero del límite superior);
```

Por ejemplo, si creamos la matriz *$numeros=range(2,6)* la matriz resultante tiene cinco elementos: el primero contiene el valor 2 y el último, el valor 6.

7.9.2. Recorrer los elementos de una matriz unidimensional

Las matrices tienen un puntero que apunta a uno de sus elementos. Cuando se accede o usa por primera vez una matriz, el puntero está posicionado sobre el primer elemento. Pero no siempre está sobre este elemento, ya que el puntero puede haberse movido para leer o modificar el contenido de la matriz.

La operación de recorrer los elementos de una matriz se lleva a cabo de forma diferente según se trate de matrices secuenciales, es decir, que tienen ordenados sus elementos por un índice ordinal como 0, 1, 2, 3, etcétera, o de matrices no secuenciales, es decir, que no tienen ordenados sus elementos por un índice ordinal.

Si creamos una matriz sin utilizar expresamente un índice, PHP le asigna un índice ordinal numérico secuencial, que se inicia con el valor 0. Así pues, los elementos quedarán ordenados por su índice desde 0 hasta su elemento n-1.

Pero podemos crear una matriz asignándole un índice numérico que no esté ordenado o incluso un índice de tipo cadena desordenado. Por ejemplo, podíamos haber creado e inicializado así la matriz *$persona*:

```
   $persona[10] = "Josefa";
   $persona[2]  = "Pérez";
   $persona[5]  = "Rubio";
   $persona[1]  = 60;
```

En este caso el primer elemento contendría también "Josefa", pero si queremos acceder al mismo por su índice deberíamos citar el 10. Así pues, esta matriz está desordenada por su índice. Veremos más adelante cómo ordenar matrices por su índice o por el contenido de sus elementos.

La función **reset()** coloca el puntero de una matriz sobre su primer elemento y devuelve su contenido. Por ejemplo, la instrucción *echo reset($persona);* devuelve "*Josefa*".

Cuando conocemos que el índice utilizado en una matriz es de tipo numérico entero secuencial (0, 1, 2,...), la forma más fácil de acceder a cada elemento es usar la función **count()** para saber cuántos elementos tiene. Después, podemos utilizar un bucle **`for`** para recorrer todos sus elementos.

La función **end()** coloca el puntero de una matriz sobre su último elemento y devuelve su contenido. Por ejemplo, la instrucción *echo end($persona);* devuelve *60*.

La función **count()** cuenta los elementos que integran una matriz y devuelve un número entero. Por ejemplo, la instrucción *echo count($persona);* devuelve *4*.

A partir de esta información, podemos recorrer los elementos y mostrar su contenido con un bucle, de esta forma:

```
$numero_elementos=count($persona);
for ($i=0; $i < $numero_elementos; $i++)
    echo $persona[$i]."<P>";
```

Podemos pensar que nos hubiéramos evitado una instrucción si ponemos la función **count()** como parte de la condición dentro de **for**. No es conveniente hacerlo por dos motivos:

1. Evitar que se ejecute la función cada vez que se pase por el bucle con la consiguiente pérdida de tiempo al contar siempre lo mismo.
2. Evitar un mal funcionamiento del bucle si dentro de las instrucciones del bucle se altera el número de elementos de la matriz con la función **unset().**

En cambio, si queremos recorrer una matriz bidimensional, no podemos usar la función **count(),** sino que debemos utilizar dos bucles, uno anidado en el otro de esta forma:

```
for ($i=0; $i < 3; $i++)
{
    for ($j=0; $j < 3; $j++)
        echo "<B>".$notas[$i][$j]." - </B>";
}
```

La función **next()** devuelve el contenido del siguiente elemento según la posición donde esté el puntero y desplaza éste a esa posición nueva. Si se ha alcanzado el último elemento, esta función devuelve *False.*

Por ejemplo, la instrucción *echo next($persona);* devuelve "*Pérez*" si el puntero estaba sobre el elemento primero antes de ejecutarse la función.

La función **prev()** devuelve el contenido del elemento anterior según la posición donde esté el puntero y desplaza éste a esa posición nueva. Si se ha alcanzado el primer elemento, esta función devuelve *False*.

Por ejemplo, la instrucción *echo prev($persona);* devuelve *False* si el puntero estaba sobre el elemento primero antes de ejecutarse la función.

La función **current()** devuelve el contenido de elemento sobre el que está el puntero sin variar su posición. Por ejemplo, *echo current($persona);* devuelve "*Rubio*" si el puntero está sobre el elemento 3. Si se ha sobrepasado el último elemento de la matriz, **current()** devuelve False.

A partir de esta información también podemos recorrer los elementos y mostrar su contenido con el bucle anterior

```
for ($i=0; $i < count($persona); $i++)
{
   echo current($persona)."<P>";
   next($persona);
}
```

La función **key()** devuelve el índice de la posición actual del puntero, se trate de un número entero o de una cadena. Por ejemplo, la instrucción *echo key($persona);* devuelve "*apellido1*" si el puntero está sobre el segundo elemento.

Podemos recorrer los elementos y mostrar su índice y su contenido con las instrucciones siguientes:

```
$datos=array("nombre"=>"Josefa","apellido1"=>"Pérez",
             "apellido2"=>"Rubio","edad"=>60);
$numero_elementos=count($datos);
for ($i=0; $i < $numero_elementos; $i++)
{
   $indice=key($datos);
   $valor=current($datos);
   echo "Indice:<B> $indice. </B>Contenido: <B>$valor</B><BR>";
   next($datos);
}
```

Este código muestra la página siguiente:

```
   Indice: nombre. Contenido: Josefa
   Indice: apellido1. Contenido: Pérez
   Indice: apellido2. Contenido: Rubio
   Indice: edad. Contenido: 60
```

7.9.3. Convertir cadenas de caracteres en matrices y viceversa

La función **explode()** permite convertir una cadena de caracteres en una matriz mediante un separador dado.

Su sintaxis es

```
explode(carácter separador, cadena);
```

La función **implode()** lleva a cabo la operación inversa: lleva los elementos de una matriz a una cadena separándolos como se indique.

Su sintaxis es

```
implode(carácter separador, matriz);
```

7.9.4. Ordenar matrices

Ya hemos comentado que los elementos de las matrices pueden estar desordenados, bien por su índice bien por su contenido. Veamos cómo podemos ordenar los elementos de una matriz de diferentes formas.

La función **arsort()** ordena los elementos de una matriz teniendo en cuenta su contenido (valor) de mayor (en el código ASCII) a menor manteniendo la asociación del contenido de cada elemento con su índice.

Su sintaxis es

```
arsort(matriz);
```

Por ejemplo, si tenemos la matriz

```
$palabras=array("1"=>"amazona","2"=>"león","3"=>"zozobra",
                "4"=>"sabueso", "5"=>"bondad","6"=>"obús");
```

la instrucción *arsort($palabras);* devuelve la matriz

```
$palabras=array("3"=>"zozobra","4"=>"sabueso","6"=>"obús",
                "2"=> "león", "5"=>"bondad","1"=>"amazona");
```

La función **asort()** hace exactamente lo mismo que la función **arsort()**, pero ordenando su contenido de menor a mayor.

Su sintaxis es

```
asort(matriz);
```

Por ejemplo, si tenemos la matriz

```
$palabras=array("1"=>"amazona","2"=>"león","3"=>"zozobra",
                "4"=>"sabueso", "5"=>"bondad","6"=>"obús");
```

la instrucción *asort($palabras);* devuelve la matriz

```
$palabras=array("1"=>"amazona","5"=>"bondad", "2"=> "león",
                "6"=>"obús", "4"=>"sabueso","3"=>"zozobra");
```

La función **rsort()** ordena los elementos de una matriz teniendo en cuenta su contenido (valor) de mayor (en el código ASCII) a menor pero sin mantener la asociación del contenido de cada elemento con su índice.

La función **sort()** hace exactamente lo mismo que la función **rsort()** pero ordenando de menor a mayor.

La función **krsort()** ordena los elementos de una matriz teniendo en cuenta su índice de mayor (en el código ASCII) a menor manteniendo la asociación de cada índice con el contenido de cada elemento.

Su sintaxis es

```
krsort(matriz);
```

Por ejemplo, si tenemos la matriz

```
$palabras=array("1"=>"amazona","2"=>"león","3"=>"zozobra",
                "4"=>"sabueso","5"=>"bondad","6"=>"obús");
```

la instrucción *krsort($palabras);* devuelve la matriz

```
$palabras=array("6"=>"obús","5"=>"bondad","4"=>"sabueso",
                "3"=>"zozobra","2"=>"león","1"=>"amazona");
```

La función **ksort()** ordena los elementos de una matriz teniendo en cuenta su índice de menor (en el código ASCII) a mayor manteniendo la asociación de cada índice con el contenido de cada elemento.

La función **shuffle()** ordena la matriz que se le pasa como parámetro de forma aleatoria.

Su sintaxis es

```
shuffle(matriz);
```

La función **uasort()** ordena los elementos de una matriz teniendo en cuenta su contenido y según una función de comparación definida por el usuario manteniendo la asociación del contenido de cada elemento con su índice. Su sintaxis es

```
uasort(matriz,función de usuario);
```

Por ejemplo, si definimos la función de usuario *compara()* y la usamos como segundo argumento de la función **uasort(),** obtenemos estos resultados:

```
function compara($a,$b)
{
 return (substr($a,3,1)>substr($b,3,1));
}

$colores=array("a"=>"blanco","b"=>"azul","c"=>"rojo",
               "d"=>"amarillo","e"=>"verde");
uasort($colores,"compara");
for (reset($colores);$indice=key($colores);next($colores))
  echo "Indice: <B>$indice</B> Contenido:<B>$colores[$indice]
        </B><P>";
```

En este caso la función **uasort()** ordena los elementos según el cuarto carácter de su contenido de menor a mayor (se cuenta desde 0; por eso hemos puesto 3 en la función de usuario), que es lo que ejecuta la función de usuario *compara()*. Así pues, la matriz queda así:

```
$colores=array("e"=>"verde","b"=>"azul","a"=>"blanco",
               "c"=>"rojo","d"=>"amarillo");
```

La función **usort()** hace exactamente lo mismo que la anterior, suponiendo lógicamente que aplicáramos la misma función de usuario que es la que determina el orden de los elementos según su contenido, pero sin mantener la asociación del índice de cada elemento con su contenido.

La función **uksort()** ordena los elementos de una matriz teniendo en cuenta su índice y según una función de comparación definida por el usuario manteniendo la asociación del índice de cada elemento con su contenido.

Su sintaxis es

```
uksort(matriz,función de usuario);
```

Por ejemplo, si definimos la función de usuario *compara()* y la usamos como segundo argumento de la función **uksort(),** obtenemos estos resultados:

```
function compara($a,$b)
{
   return (substr($a,3,1)>substr($b,3,1));
}
$colores=array("primero"=>"blanco","segundo"=>"azul",
               "tercero"=>"rojo","cuarto"=>"amarillo",
               "quinto"=>"verde");
uasort($colores,"compara");
for (reset($colores);$indice=key($colores);next($colores))
    echo "Indice: <B>$indice</B> Contenido:<B>$colores[$indice]
          </B><P>";
```

En este caso la función **uksort()** ordena los elementos según el cuarto carácter de su índice de mayor a menor (se cuenta desde 0; por eso hemos puesto 3 en la función de usuario), que es lo que ejecuta la función de usuario *compara()*.

Así pues la matriz queda así:

```
$colores=array("segundo"=>"azul","cuarto"=>"amarillo",
               "quinto"=>"verde","primero"=>"blanco",
               "tercero"=>"rojo");
```

7.9.5. Modificar matrices

Las matrices ya creadas e inicializadas pueden modificarse de diferentes formas:

❶ Uniendo los elementos de dos o más matrices.

La función **array_merge(),** con comportamiento diferente a partir de la versión 5 de PHP, recibe como argumentos dos o más matrices y devuelve otra en la que se han unido todas las que se pongan como argumentos anexionando los elementos de unas a continuación de las otras, según el orden en que se pongan los argumentos.

Su sintaxis es

```
array_merge(matriz1,matriz2...);
```

Si en la mezcla algún índice se repite, el último en añadirse elimina el elemento de las matrices anteriores en que se repita.

❷ Completando los elementos de una matriz.

La función **array_pad()** rellena una matriz añadiendo los elementos que se indique con el contenido que se especifique.

Su sintaxis es

```
array_pad(matriz,número de elementos de la matriz resultante,relleno de
los elementos añadidos);
```

donde matriz es la matriz original a la que se añaden elementos; número de elementos es un entero que indica cuántos elementos tendrá la matriz después de ejecutarse la función; relleno es la cadena o valor con el que se completará el contenido de los nuevos elementos.

Si número de elementos es igual o inferior al número de elementos de la matriz original, no se produce ningún cambio en la misma. Si es negativo, los elementos se añaden por el principio de la matriz original. Si es positivo, se añaden por el final.

Por ejemplo, si tenemos la matriz de cinco elementos

```
$sabores=array("dulce","amargo","salado","soso","menta");
```

la instrucción

```
$completa=array_pad($sabores,8,"indeterminado");
```

genera la matriz

```
$completa=array("dulce","amargo","salado","soso","menta",
                  "indeterminado","indeterminado","indeterminado");
```

❸ Invirtiendo el orden de la matriz original.

La función **array_reverse(),** añadida en la versión 4 de PHP, devuelve otra matriz que contiene los mismos elementos ordenados al revés, es decir, invierte el orden de los elementos de forma que el primero pasa a ser el último y el último pasa a ser el primero.

Su sintaxis es

```
array_reverse(matriz);
```

❹ Sustituyendo un fragmento de elementos por otro.

La función **array_splice(),** devuelve la misma matriz sustituyendo varios elementos consecutivos por otros nuevos.

Su sintaxis es

```
array_splice(matriz original,posición inicial del
desplazamiento,longitud del desplazamiento,cadena-valor-matriz que
sustituye el fragmento de la original);
```

donde:

- el argumento posición inicial del desplazamiento es un número entero que indica la posición, contando desde 0, del primer elemento que debe ser sustituido. Si es menor que 0, se cuenta desde el último elemento de la matriz original siendo el último elemento el que ocupa la posición –1.
- el argumento longitud del desplazamiento es un número entero que indica el número de elementos que serán sustituidos a partir de la posición especificada en el argumento anterior. Si se omite, se eliminan todos desde la posición indicada. Si es menor que 0, se eliminan tantos como se indique, pero contando desde el final de la matriz original hasta la posición especificada.
- el argumento cadena, valor o matriz contiene el contenido que ha de asignarse a los elementos eliminados. Puede no existir. En este caso, la matriz original pierde los elementos eliminados sin ser sustituidos por otros.

Veamos algunos ejemplos:

```
   $matriz1=array("muy alto","alto","medio",
                  "bajo","muy bajo");
   $matriz2=array("óptimo","bueno","malo");
   array_splice($matriz1,2,2);
```

genera la matriz

```
$matriz1=array("muy alto","alto","muy bajo");
```

Se han eliminado los elementos tercero y cuarto.

```
array_splice($matriz1,2);
```

genera la matriz

```
$matriz1=array("muy alto","alto");
```

Se han eliminado todos los elementos desde el tercero.

```
array_splice($matriz1,-2);
```

genera la matriz

```
$matriz1=array(("muy alto","alto","medio");
```

Se han eliminado los dos últimos elementos.

```
array_splice($matriz1,2,1);
```

genera la matriz

```
$matriz1=array("muy alto","alto","bajo","muy bajo");
```

Se ha eliminado el elemento tercero.

```
array_splice($matriz1,3,3,$matriz2);
```

genera la matriz

```
$matriz1=array("muy alto","alto","medio", "óptimo","bueno", "malo");
```

Se han sustituido, desde el elemento tercero, tres elementos con el contenido de la matriz *$matriz2*.

```
array_splice($matriz1,count($matriz1),3,$matriz2);
```

genera la matriz

```
$matriz1=array("muy alto","alto","medio","bajo",
               "muy bajo","óptimo","bueno","malo");
```

Se han añadidos los elementos de la segunda matriz a los de la primera.

No conviene confundirse con esta función: si generamos una segunda matriz con esta misma orden, ésta contendrá los elementos eliminados.

Por ejemplo, la orden

```
$nueva=array_splice($matriz1,3,3,$matriz2);
```

genera la matriz

```
$nueva=("bajo","muy bajo");
```

que son los dos elementos eliminados de la matriz original.

❺ Ejecutando una función de usuario sobre cada elemento de la matriz

La función **array_walk(),**devuelve otra matriz en cuyos elementos se ha ejecutado una función de usuario.

Su sintaxis es

```
array_walk(matriz original,función de usuario,prefijo);
```

❻ Juntando variables en una sola matriz

La función **compact()** devuelve otra matriz en la que se han incorporado como elementos las diferentes variables y matrices que se especifiquen como argumento. Como índice se pone el nombre de la variable y como contenido el valor de cada variable.

Su sintaxis es

```
compact(lista de variables o arrays...);
```

7.9.6. Extraer información de las matrices

La función **array_count_values()** recibe como argumento una matriz y devuelve otra cuyos índices son los contenidos de la original y cuyo contenido es la frecuencia con que se repite el mismo contenido en la matriz original.

Su sintaxis es

```
array_count_values(matriz);
```

La función **array_keys()** recibe como argumento una matriz y devuelve otra que contiene sólo los índices de la matriz original como contenidos de los nuevos elementos. Como índices de la nueva matriz se asignan valores numéricos enteros desde 0 en adelante.

Su sintaxis es

```
array_keys(matriz,valor);
```

El segundo argumento es opcional y, si se usa, sirve para decir que pase a la matriz resultante sólo los índices de los elementos que contengan ese valor.

La función **array_slice(),** devuelve otra matriz que contiene una porción de elementos de la matriz original.

Su sintaxis es

```
array_slice(matriz original, posición inicial del desplazamiento,
longitud del desplazamiento);
```

donde

- el argumento posición inicial del desplazamiento es un número entero que indica la posición, contando desde 0, del primer elemento que debe ser extraído. Si es menor que 0, se cuenta desde el último elemento de la matriz original siendo el último elemento el que ocupa la posición –1.
- el argumento longitud del desplazamiento es un número entero que indica el número de elementos que serán extraídos a partir de la posición especificada en el argumento anterior. Si se omite, se extraen todos los elementos desde la posición indicada. Si es menor que 0, se extraen tantos como se indique, pero contando desde el final de la matriz original hasta la posición especificada.

Como se ve, esta función tiene un comportamiento similar a la función **`array_splice(),`** pero aquí se extraen los elementos y en aquélla se eliminan o sustituyen.

```
$altura=array("muy alto","alto","medio","bajo","muy bajo");
$nueva=array_slice($altura,2,2);
```

genera la matriz

```
$nueva=array("medio","bajo");
```

Se han extraído los elementos tercero y cuarto.

La función **array_values()** recibe como argumento una matriz y devuelve otra en la que se han eliminado los índices dejando sólo los contenidos.

Su sintaxis es

```
array_values(matriz original);
```

Lógicamente, como no puede haber una matriz sin índices, a cada elemento se le asignan índices numéricos desde 0 en adelante y se eliminan los que tuviera antes, sean cadenas u otros valores numéricos.

La función **each()** lee el elemento actual de una matriz y avanza el puntero al elemento siguiente. Con el elemento leído fabrica otra matriz de cuatro elementos: dos para el índice (0 y key) y dos para el contenido (1 y value). Generalmente, esta función se utiliza combinada con la función **`list()`** para recorrer los elementos de una matriz con un bucle.

Su sintaxis es

```
each(matriz original);
```

Por ejemplo, si tenemos la matriz

```
$con_indices=array("uno"=>"muy alto","dos"=>"alto",
                   "tres"=>"medio","cuatro"=>"bajo",
                   "cinco"=>"muy bajo");
```

y suponemos que el puntero está sobre el elemento *`"tres"=>"medio"`*, la instrucción *`$nueva=each($con_indices);`* devuelve la matriz de cuatro elementos siguiente

```
$nueva=array(0=>"tres","key"=>"tres", 1=>"medio","value"=>"medio")
```

Los dos primeros elementos de la matriz resultante hacen referencia al índice del elemento actual de la matriz original. De esta forma podemos referirnos a su contenido también de dos formas: *$nueva[0]* o *$nueva("key")*.

Como puede verse, los dos últimos elementos de la matriz resultante hacen referencia al contenido del elemento actual de la matriz original. De esta forma podemos referirnos a su contenido de dos formas: como *$nueva[1]* o como *$nueva("value")*.

La función **extract()** pasa los contenidos de los elementos de una matriz como contenidos de las variables que llevan como nombre el índice de cada elemento de la misma matriz. Como consecuencia, a la tabla de símbolos de variables se incorporan estas nuevas variables.

Su sintaxis es

```
extract(matriz, si hay conflictos tipo de extracción,si hay conflictos
cadena de sustitución);
```

Los dos últimos argumentos son opcionales y sólo se aplican, si existen, cuando al ejecutar la extracción se han producido colisiones, es decir, si la variable que se pretendía crear ya existía e incluso si era de diferente tipo.

En caso de colisión entre variables, el segundo argumento tiene los valores siguientes:

- EXTR_OVERWRITE: se descarta el valor anterior de la variable y se pone el nuevo. Este valor es el que se usa por defecto en caso de colisión, aunque no se indique como argumento.
- EXTR_SKIP: se descarta el valor nuevo de la variable y se pone el anterior.
- EXTR_PREFIX_SAME: al nombre de la variable nueva conflictiva se le pone como prefijo la cadena del tercer argumento seguida de un guión de subrayado (_).
- EXTR_PREFIX_ALL: al nombre de todas las variables nuevas se le pone como prefijo la cadena del tercer argumento seguida de un guión de subrayado (_).
- EXTR_PREFIX_INVALID: solo agrega el prefijo a nombres de variables inválidas.
- EXTR_IF_EXISTS: solo sobrescribe la variable si ya existe.
- EXTR_PREFIX_IF_EXISTES: solo extrae y añade prefijo a las variables si estas xa existen sin prefijo
- EXTR_REFS: extrae las variables como referencias a los elementos originales de la matriz, de forma que si se modifica esta se modifican los valores de las variables.

Esta función comprueba también si el nombre de las nuevas variables es válido y sólo las crea si lo es.

La función **in_array()** devuelve *True* si un valor está contenido en alguno de los elementos de una matriz y *False* si no lo está,

Su sintaxis es

```
in_array(valor buscado, matriz donde se busca);
```

La construcción **list()** asigna en una única operación el valor de los elementos de una matriz a una lista de variables que se usan como argumento. Ya comentamos que la función **`each()`** se usa, por lo general, combinada con la función **`list()`** para recorrer los elementos de una matriz dentro de un bucle.

Su sintaxis es

```
list(lista de variables);
```

Por ejemplo, si tenemos la matriz

```
$comunidad=array("CA1"=>"Aragón","CA2"=>"Andalucía",
                 "CA3"=>"Galicia","CA4"=>"Murcia",
                 "CA5"=>"Valencia","CA6"=>"Canarias");
```

el bucle siguiente recorre y muestra el contenido de todos sus elementos:

```
while (list($indice,$valor)=each($comunidad))
{
   print ("El elemento <B>$indice</B> contiene el valor <B>$valor </B><P>");
}
```

En la condición con la que se establece el bucle combinando **`list()`** y **`each()`** se consigue que **`list()`** tome en sus dos variables el índice y el contenido de cada elemento, que luego se muestran. Además, una vez proporcionada por **`each()`** la información de cada elemento a **`list()`**, hace avanzar el puntero un elemento más, por lo que el bucle se acaba cuando ya no hay elementos que ofrezcan la información necesaria.

7.9.7. Tratar un array como si fuera una pila

La versión 4 de PHP ha incorporado cuatro funciones que permiten tratar las matrices como si fueran pilas de tipo LIFO (*Last In First Out*, Último en Entrar Primero en Salir). El concepto de pila ya es clásico en Informática y se refiere a la forma de almacenar información en forma de montón o pila, Si tenemos muchos platos, podemos almacenarlos apilándolos unos encima de otros. En este caso lo más usual será que el último plato colocado en la pila sea el primero que se retira. Las matrices también pueden considerarse como pilas de datos, que son sus elementos.

Veamos las funciones que nos permiten realizar operaciones de tipo pila con las matrices.

La función **array_pop()** extrae el último elemento de la matriz eliminándolo.

Su sintaxis es

```
array_pop(matriz);
```

La función **array_push()** añade elementos al final de la matriz.

Su sintaxis es

```
array_push(matriz,lista de elementos que se añaden);
```

La función **array_shift()** extrae el primer elemento de la matriz eliminándolo y desplazando todos una posición a la izquierda.

Su sintaxis es

```
array_shift(matriz);
```

La función **array_unshift()** inserta elementos por el principio de la matriz desplazando los demás a la derecha.

Su sintaxis es

```
array_unshift(matriz,lista de elementos que se añaden);
```

Es conveniente observar que, cuando añadimos elementos, PHP crea los índices para cada uno, que son de tipo numérico ordinal.

7.10. Funciones de Tratamientos de Fechas

El uso de fechas es muy frecuente e imprescindible en Internet. PHP dispone también de funciones para tratar y obtener fechas. En este apartado se verán las más importantes y necesarias, así como su aplicación en las páginas web .

En el **Capítulo XXIII. Funciones de fecha y hora** del **Manual Oicial de PHP** aparecen todas las funciones de fecha.

7.10.1. Comprobar la validez de una fecha

La función **checkdate()** nos permite comprobar si una fecha, que se pone como argumento, es válida. Si lo es, la función devuelve el valor *True*. En el caso contrario, devuelve *False*.

Se considera como válida una fecha cuando su año está comprendido entre los años 1 y el 32767, su mes entre 1 y 12, y su día entre 1 y 28 para febrero no bisiesto, 29 para febrero bisiesto, 30 para abril, junio, septiembre y noviembre, y 31 para el resto de los meses. Así pues, esta función controla los años bisiestos al comprobar la validez de una fecha.

La sintaxis de esta función es

```
checkdate(mes,día,año);
```

donde los tres parámetros son números enteros. Conviene observar el orden en que deben introducirse los parámetros, ya que no coinciden con la posición habitual de nuestras fechas (día,mes,año).

Por ejemplo, la instrucción `checkdate(0,25,2000);` devuelve *False*. En cambio la instrucción `checkdate(7,25,2000);` devuelve *True*.

7.10.2. Dar formato a una fecha y una hora

La función **date()** devuelve una cadena de texto que refleja una fecha y una hora formateadas como se indique en el primer parámetro. En el segundo argumento puede indicarse la fecha que se quiere mostrar utilizando un valor de tipo *timestamp* (instante calculado como número de segundos transcurridos desde el 1/1/1970, época UNIX). Si no se especifica este segundo parámetro, se muestra la hora actual.

La sintaxis de esta función es:

```
date(cadena de formato, número entero del instante o nada);
```

Tabla de caracteres y su significado en la cadena de formato

Signo	Descripción de su resultado en la página *web*
a	"am" o "pm"
A	"AM" o "PM"
c	fecha completa ISO 8601
d	día del mes, dos dígitos con cero a la izquierda; es decir, de "01" a "31"
D	día de la semana, en texto, con tres letras; por ejemplo, "Fri"
e	zona horaria; por ejemplo “UTC”,”GMT”
F	mes, en texto, completo; por ejemplo, "January"
h	hora, de "01" a "12"
H	hora, de "00" a "23"
g	hour, sin ceros, de "1" a "12"
G	hour, sin ceros; de "0" a "23"
i	minutos; de "00" a "59"
j	día del mes sin cero inicial; de "1" a "31"
l	día de la semana, en texto, completo; por ejemplo, "Friday"
L	"1" or "0", según si el año es bisiesto o no
m	mes; de "01" a "12"
n	mes sin cero inicial; de "1" a "12"
N	día de la semana: “1” para lunes … “7” para domingo
M	mes, en texto, 3 letras; por ejemplo, "Jan"
s	segundos; de "00" a "59"
S	sufijo ordinal en inglés, en texto, 2 caracteres; por ejemplo, "th", "nd"
t	número de días del mes dado; de "28" a "31"
u	Milisegundos
U	segundos desde el valor de 'epoch' UNIX
w	día de la semana, en número, de "0" (domingo) a "6" (sábado)
W	número de la semana en el año
Y	año, cuatro cifras; por ejemplo, "1999"
y	año, dos cifras; por ejemplo, "99"
z	día del año; de "0" a "365"
Z	diferencia horaria en segundos (de "-43200" a "43200")

Conviene observar que esta cadena no utiliza categorías locales, por lo cual los nombres de los días y los meses aparecen en inglés. Para formatear una fecha y mostrar su resultado, es mejor utilizar la función **strftime()**, que se explica después.

7.10.3. Extraer información de una fecha

La función **getdate()** se usa para extraer información de una fecha dada. Tampoco admite categorías locales y, por tanto, la información literal está en inglés.

La sintaxis de esta función es:

```
getdate(número entero del instante);
```

El instante es un número referido a los segundos transcurridos desde el día 1/1/1970 hasta la fecha. La información obtenida se almacena en un *array* cuyos índices son los siguientes:

1. **"seconds"**: número de segundo
2. **"minutes"**: número de minutos
3. **"hours"**: número de horas
4. **"mday"**: día del mes
5. **"wday"**: día de la semana en numero
6. **"mon"**: número de mes
7. **"year"**: año
8. **"yday"**: día del año
9. **"weekday"**: nombre del día de la semana
10. **"month"**: nombre del mes.
11. **"0"**: segundos desde el comienzo de la época Unix

Por ejemplo, la instrucción `$fecha=getdate();` asigna al array `$fecha` los nueve contenidos indicados antes.

La función **mktime()** se usa para extraer el valor *timestamp* de una fecha. Después, podemos utilizar este valor como argumento de otras funciones. El término *timestamp* se refiere al momento exacto de una fecha concreta expresado en segundos desde el día 1 de enero de 1970 (época UNIX). Esta función nos permite hallar el número de segundos que han transcurrido desde entonces hasta la fecha que especifiquemos.

La sintaxis de esta función es

```
mktime(horas,minutos,segundos,mes,día,año);
```

Por ejemplo, si queremos saber el momento *timestamp* del día 16 de diciembre de 1999, a las 10 horas, 15 minutos y 20 segundos, debemos escribir la instrucción `$momento=mktime(10,15,20,12,16,1999);`. El resultado es `945339320`.

Ahora podemos dar este número como argumento de la función **getdate()** y obtenemos la misma fecha que le dimos a **mktime()**. Esta función es, además, muy útil para comprobar fechas y compararlas.

7.10.4. Dar formato a una fecha traduciendo los nombres

La función **strftime()** se usa para formatear una fecha teniendo en cuenta las especificaciones y categorías locales. Así soslayamos los problemas de este tipo que plantean las funciones **date()** y **getdate()**.

La sintaxis de esta función es

```
strftime(cadena de formato,instante);
```

Si no se proporciona el segundo parámetro, que es el número de segundos, se toma la hora del sistema. Veamos ahora los signos de la cadena de formato y sus resultados en la presentación de una fecha.

Tabla de caracteres y su significado en la cadena de formato

Signo	Descripción de su resultado en la página *web*
%a	nombre del día de la semana abreviado
%A	nombre del día de la semana completo
%b	nombre del mes abreviado
%B	nombre del mes completo
%c	representación de fecha y hora preferidas en el idioma actual
%C	número de la centuria (año dividido entre 100)
%d	día del mes en número (de 00 a 31)
%H	hora como un número de 00 a 23
%I	hora como un número de 01 a 12
%j	día del año como un número de 001 a 366
%m	mes como un número de 01 a 12
%M	minuto en número
%p	'am' o 'pm', según la hora dada, o las cadenas correspondientes en el idioma actual
%r	hora en notación a.m. y p.m.
%R	hora en notación de 24 horas
%S	segundos en número
%U	número de la semana en el año, empezando con el primer domingo como el primer día de la primera semana
%W	número de la semana en el año, empezando con el primer lunes como el primer día de la primera semana
%w	día de la semana en número (el domingo es el 0)
%x	representación preferida de la fecha sin la hora
%X	representación preferida de la hora sin la fecha
%y	año en número de 00 a 99
%Y	año en número de cuatro cifras
%Z	nombre o abreviatura de la zona horaria
%%	carácter '%'

Si queremos que los nombres aparezcan en castellano, aquí sí podemos incluir en el script la instrucción `setlocale(LC_ALL,"spanish");` para traducirlos.

8. Entrada y Salida de Datos

- *Entrada de Datos*
- *Salida de Datos*
- *Consideraciones sobre la E/S*

8.1. Entrada de Datos

Normalmente la entrada de datos desde teclado se realiza a través de formularios HTML (forms). Los formularios nos proporcionan la posibilidad de preguntar al usuario cliente una serie de datos que pueden ser devueltos al servidor. Trabajar en PHP con *forms* es muy fácil, ya que el propio lenguaje se encarga de crear automáticamente las variables necesarias para almacenar los datos del form en la página que los recibe. Vamos a analizar cada uno de los casos:

8.1.1. Valores sencillos

Para los campos de un formulario en los que únicamente se puede elegir o introducir un valor (como campos de texto, desplegables y "radio buttons") PHP crea en el fichero de destino un elemento en la variable superglobal del método que se ha empleado para enviar el formulario (POST, GET,..), con clave igual al nombre del elemento del formulario. Es decir, la clave del elemento de la variable superglobal de PHP ($_GET, $_POST, $_REQUEST) en la página destino es igual al atributo Name del elemento del formulario HTML en la página origen. Por ejemplo, si tenemos este formulario:

```
<form action="accion.php" method="POST">
Su nombre: <input type=text name=nombre><br>
Su edad: <input type=text name=edad><br>
<input type=submit>
</form>
```

En el script de destino *accion.php*, accederíamos a los valores enviados en el form así:

```
Hola <?php echo $_POST['nombre']?>.
Tiene <?php echo $_POST['edad']?> años.
```

Analicemos cada uno de los elementos de los formularios que se encuentran es esta situación:

Cuadros De Texto (Text)

Como hemos visto en el ejemplo anterior se origina un elemento de la matriz supreglobal con nombre igual al atributo Name del elementos *input type='text'* del formulario HTML y con valor igual al contenido introducido por el usuario.

```
<input type="text" name="nombre" size="25" maxlenght="25">

<?php
   echo $_REQUEST['nombre'];
?>
```

Botones De Activación (Radio-Button)

En este caso se origina un elemento de la matriz supreglobal con nombre igual al atributo Name del elementos *input type='radio'* del formulario HTML y con valor igual al atributo Value del botón de activación seleccionado por el usuario.

```
<input type="radio" name="nombre" value="Hombre">
<input type="radio" name="nombre" value="Mujer">

<?php
   echo $_REQUEST['nombre'];
?>
```

Casillas de Verificación (CheckBox)

En este caso se origina un elemento de la matriz supreglobal con nombre igual al atributo Name del elementos *input type='checkbox'* del formulario HTML que hayan sido seleccionados (de los que no se seleccionen no se genera elemento) y con valor igual al atributo Value de la casilla de verificación correspondiente.

```
<input type="checkbox" name="nombre1" value="Hombre">
<input type="checkbox" name="nombre2" value="Mujer">

<?php
   if (!empty($_REQUEST['nombre1']))
   {
      codigo
   }
?>
```

8.1.2. Valores múltiples

Para los campos de selecciones múltiples PHP también se encarga de almacenar los valores marcados por el usuario en un elemento de la matriz supreglobal (array) con el nombre del campo, pero en este caso tendremos que llevar cuidado de dar a las variables nombres de *arrays*, ya que si no sólo tendremos acceso al primer valor seleccionado. Por ejemplo este formulario estaría mal:

```
<form action="accion.php" method="POST">
<select multiple name=menu>
<option>Tortilla <option>Paella
<option>Fabada <option>Lentejas
</select><input type=submit>
</form>
```

Deberíamos haberlo escrito así:

```
<form action="accion.php" method="POST">
<select multiple name=menu[]>
<option>Tortilla <option>Paella
<option>Fabada <option>Lentejas
</select><input type=submit>
</form>
```

Y podemos ver el resultado con este código:

```
<?php
    echo "Su elección:<br>";
    foreach($_POST['menu'] as $plato) {
        echo "$plato<br>\n";
    }
?>
```

8.2. Salida de datos

Normalmente la salida de datos se realiza por pantalla a través de paginas HTML. Para ellos se han de usar alguna de las funciones vistas en el apartado Funciones de Tratamientos de Cadenas del capítulo de Funciones, donde se explican en detalle.

Dichas funciones son **echo()**, **print()**, **printf()** y **sprintf()**. Su sintaxis es:

```
echo (argumento1, argumento2,...)
```

Si el argumento es una cadena entre comillas dobles, debemos tener en cuenta que las variables incluidas en ese texto se expanden, es decir, muestran su contenido, no su nombre, pero las funciones no. Para que las variables no muestren su contenido debemos utilizar la barra hacia la izquierda (\) delante de la variable. Además, si queremos que las funciones se expandan (se ejecuten) debemos sacarlas de las comillas y concatenarlas con el texto anterior y posterior usando el operador de concatenación (**.**).

```
print(argumento1, argumento2,...)
```

Esta orden es prácticamente igual que **echo**, hasta el punto de que se pueden intercambiar sin problemas.

```
printf ("cadena de formato", argumento1, argumento2,...)
```

En los argumentos se incluye la información que se quiere mostrar, como con **echo** o con **print**. En la cadena de formato, entre comillas, puede ponerse un texto, que aparece tal cual lo escribamos, con expansión de las variables si no usamos el signo \, y el valor de una variable según se establezca en la directiva %.

```
$cadena=sprintf("cadena de formato", argumento 1, argumento 2,...)
```

Esta función nos permite dar formato a una cadena de texto exactamente igual que con **printf**, y guardar, además el resultado en una variable, para utilizarlo posteriormente.

8.3. Consideraciones sobre la E/S

A la hora de trabajar con formularios hemos de tener en cuenta una serie de consideraciones:

ⓘ Conviene saber que en PHP es frecuente que una página se llama a sí misma. Este procedimiento se denomina recursividad. Cuando se ejecuta el código por primera

vez, sólo se muestra el formulario vacío y no aparecen los datos. Es lógico, pues las variables aún no tienen contenido. Cuando las variables tengan algún contenido, introducido en la página mediante el formulario, se puede mostrar esta información en la página llamada de nuevo al pulsarse el botón Aceptar y ejecutarse la orden

```
<FORM ACTION=index.php METHOD=POST>
```

Esta forma de proceder es muy útil para verificar si los datos introducidos son correctos. Si ponemos las condiciones de corrección dentro del código PHP, la comprobación se realiza en el servidor.

Ya que podemos cambiar el nombre del fichero PHP que se llama a sí mimo, en lugar de citar dentro del código expresamente su nombre, como hemos hecho, podemos usar la variable de entorno $_SERVER['PHP_SELF'] que guarda el nombre del fichero al ejecutarse. Escribiríamos en este caso:

```
<FORM ACTION=<?php echo $_SERVER['PHP_SELF'] ?> METHOD=POST>
```

ⓘ Para pasar información de una página a otra a través de la URL debe realizarse poniendo en la URL el nombre del fichero donde está el código de la página a la que hay que pasar los valores, seguido del signo **?**. Detrás de éste, se pone el nombre de la variable cuyo contenido se debe pasar sin usar el signo $, un **=** y el valor que se pasa a esa variable. Si hay más de una variable, se ponen todas separadas por el signo **&**, como en el ejemplo siguiente.

```
<A HREF="resultado.php?nom=Pedro&apel=García&des=Alto">
   Pulse aquí para enlazar con la otra página </A>
```

ⓘ HTML no entiende de espacios y, por tanto, si el contenido de las tres variables anteriores hubiera tenido más de una palabra con espacios en medio, la página no se habría mostrado, sino que se produciría un error. Para que HTML respete los posibles espacios entre palabras, es preciso utilizar la función **urlencode()** que se ha explicado en el apartado Funciones de Tratamiento de Cadenas del capítulo de Funciones.

```
<?php
  $variables_pasadas="nom=".urlencode("Pedro José.").
                     "&apel=".urlencode("García de Dios.").
                      "&des=".urlencode("Alto y moreno.");
?>

<A HREF=resultado.php?<?php echo $variables_pasadas; ?> >
   Pulse aquí para enlazar con la otra página </A>
```

ⓘ Cuando se pasan bastantes variables mediante un enlace, lo mejor es crear otra variable que contenga una cadena de caracteres donde se concatenen los nombres de todas las variables pasadas, de forma que, detrás del nombre del fichero que se cita en la orden HREF, se ponga sólo el nombre de la variable que contiene todos los nombres como en el ejemplo anterior.

9. Ficheros y Directorios

- *Abrir y Cerrar Ficheros*
- *Recorrer ficheros y Leer su Contenido*
- *Modificar el contenido de los ficheros*
- *Copiar, borrar y renombrar ficheros*
- *Operaciones con directorios*

9.1. Abrir y Cerrar Ficheros

En este capítulo vamos a abordar el tratamiento de los ficheros en PHP. PHP permite guardar y recuperar información a través de los ficheros, para lo cual dispone de funciones y procedimientos para acceder a los mismos y a su información, así como para crearlos, destruirlos, modificar su contenido, etc. PHP también puede tratar las carpetas o directorios.

En esta capítulo, pues, vamos a estudiar de forma teórico-práctica las diferentes operaciones que se pueden llevar a cabo con los directorios y con los ficheros. Concretamente, en este primer apartado abordamos cómo abrir y cómo cerrar ficheros.

En el **Capítulo XLI. Funciones del sistema de archivos** del **Manual Oficial de PHP** aparecen las funciones de ficheros. En este apartado nos vamos a limitar a explicar brevemente las más frecuentes y necesarias.

9.1.1. Abrir un fichero

Siempre que trabajemos con ficheros para escribir, modificar o leer su contenido, es imprescindible realizar un proceso que está integrado, al menos, por tres operaciones:

1. Abrir el fichero.
2. Realizar las operaciones de lectura o escritura previstas.
3. Cerrar el fichero.

La función **`fopen()`** permite abrir un fichero para trabajar con él. Al ejecutarse, comprueba si el fichero pasado como primer argumento existe o no. En el primer caso esta función devuelve *`True`*; en el segundo, *`False`*. Por ello, conviene incluir también esta función dentro de una estructura condicional que nos informe de la apertura o no del fichero especificado.

Ésta es su sintaxis:

```
fopen("nombre del fichero","modo de apertura",
       entero para mirar también en el camino fijado
       en la instrucción include);
```

donde

El primer parámetro es el nombre exacto del fichero que se desea abrir. Si dentro de este argumento se indica el directorio por no haber usado previamente la función

`chdir()`, es imprescindible poner las dos barras \\, para que la barra de directorio sea interpretada correctamente, es decir, como signo textual de directorio y no como un símbolo del lenguaje HTML.

```
$gestor = fopen("c:\\datos\\info.txt", "r");
```

El segundo parámetro es el modo de apertura, que debe ponerse también entre comillas. Un fichero puede abrirse de varios modos, cada uno de los cuales se indica con el signo correspondiente:

- '**r**' - Abre el fichero para sólo lectura; sitúa el puntero del fichero al comienzo del mismo.
- '**r+**' - Abre el fichero para lectura y escritura; sitúa el puntero del fichero al comienzo del fichero.
- '**w**' - Abre el fichero para sólo escritura; sitúa el puntero del fichero al comienzo del fichero y trunca el fichero con longitud cero. Si el fichero no existe, trata de crearlo.
- '**w+**' - Abre el fichero para lectura y escritura; sitúa el puntero del fichero al comienzo del mismo y trunca el fichero con longitud cero. Si el fichero no existe, trata de crearlo.
- '**a**' - Abre el fichero sólo para escribir (añadir); sitúa el puntero del fichero al final del mismo. Si el fichero no existe, trata de crearlo.
- '**a+**' - Abre el fichero para lectura y escritura (añadiendo); sitúa el puntero del fichero al final del mismo. Si el fichero no existe, trata de crearlo.
- '**x**' – Crea y abre el fichero sólo para escribir; sitúa el puntero del fichero al comienzo del mismo. Si el fichero ya existe, devolvera *False*.
- '**x+**' - Crea y abre el fichero para lectura y escritura; sitúa el puntero del fichero al comienzo del mismo. Si el fichero ya existe, devolvera *False*.

Además de uno de los anteriores, este argumento, denominado mode, puede contener la letra '**b**'. Esto es útil para sistemas que diferencian entre ficheros binarios y de texto. Si no se necesita, es ignorado.

El tercer parámetro, opcional, permite indicar que la búsqueda del fichero se haga también en el camino donde se hallan los ficheros especificados en la orden **include**. En este caso, hay que poner el valor 1.

Veamos un ejemplo:

```
if (fopen("Ficheros_1.php","r"))
   echo "<B>El fichero \"Ficheros_1.php\" existe
         y ha quedado abierto.</B>";
else
   echo "<B>El fichero \"Ficheros_1.php\" no existe.</B>";
```

En las instrucciones anteriores se intenta abrir el fichero *Ficheros_1.php* en modo de sólo lectura especificando que, si se puede abrir, el puntero quede al principio del mismo.

Conviene observar que, al igual que ocurre con las matrices, en los ficheros también hay un puntero o apuntador que señala una de las posiciones del fichero.

Si no se hubiera usado antes la función **`chdir()`**, sería preciso haber escrito:

```
if (fopen("C:\\WAMP\\WWW\Capitulo_09\\fichero1.php","r"))
```

En lugar de poner una estructura de control condicional, también es posible utilizar una sintaxis más elegante usando la función **`die()`**, que permite devolver un mensaje al navegador del cliente y da por finalizado el script que se está interpretando. Así quedaría el código en este caso:

```
@fopen("Fichero_1.php","r") or
die("<B>El fichero \"Fichero_1.php\" no se ha podido abrir.</B><P>");
```

De esta forma, más elegante y eficaz, se consigue evitar el mensaje de error que se envía al cliente (por eso hemos usado el operador **@**) y se dejan de ejecutar las siguientes líneas del código (gracias a la función **`die()`**). Si el fichero existe y puede abrirse, no se muestra el mensaje y se prosigue con la interpretación de las siguientes líneas de código.

La forma más frecuente de citar el nombre de un fichero y su identificador es usando dos variables, una para que contenga el nombre del fichero y otra para que contenga el identificador del mismo. Mira el siguiente código:

```
$fichero1="Ficheros_1.php";
$id_fichero1=@fopen($fichero1,"r")
            or die("<B>El fichero \"Fichero_1.php\" no se
                    ha podido abrir.</B><P>");
echo "<B>El fichero \"Ficheros_1.php\" existe y ha quedado
              abierto en modo lectura.</B><P>";
```

En la variable *`$fichero1`* hemos guardado el nombre del fichero; la variable *`$id_fichero1`* contiene el identificador de fichero. Para realizar cualquier operación con este fichero abierto, debemos utilizar el identificador creado.

Si se pretende abrir un fichero en modo escritura y no existe, se intenta crear el fichero especificado. Si existe, es preciso tener mucho cuidado con el parámetro mode que se utilice, ya que **"w"** y **"w+"** truncan el fichero dejando su tamaño a cero y colocando el puntero al principio del fichero. Esto quiere decir que se perderá su contenido anterior. En cambio, **"a"** y **"a+"** no truncan el fichero y colocan el puntero al final del fichero. Por lo tanto, el contenido nuevo se añadirá a lo que ya había.

```
$fichero2="Ficheros_2.txt";
$id_fichero2=@fopen($fichero2,"w");
```

Con estas dos instrucciones, si el fichero *`Ficheros_2.txt`* no existía, se crea con tamaño *0*; si ya existía y tenía contenido, se deja vacío.

```
$fichero3="Ficheros_3.txt";
$id_fichero3=@fopen($fichero3,"a");
```

Con estas dos instrucciones, si el fichero *`Ficheros_3.txt`* existía, se abre y se deja el puntero al final del mismo, disponible para añadir más contenidos; si no existía, se crea con tamaño *0* también disponible para añadir contenidos.

Se puede acceder a ficheros que no se encuentran en el sistema local haciendo uso de protocolos tales como HTTP o FTP:

```
$gestor = fopen("http://www.example.com/", "r");
$gestor = fopen("ftp://usuario:contrasenya@example.com/un_archivo.txt", "w");
```

9.1.2. Cerrar un fichero

Todo fichero que ha sido abierto debe ser cerrado una vez que se ha acabado de realizar con él las operaciones previstas.

La función **fclose()** permite cerrar los ficheros abiertos. Su sintaxis es sencilla:

```
fclose(nombre de identificador);
```

Se cierra sólo el fichero del identificador que se especifique. Al ejecutarse esta función, se comprueba si el fichero pasado como argumento está abierto y lo ha podido cerrar o no. En el primer caso esta función devuelve `True`; en el segundo (si el fichero no existe o no está abierto), `False`.

9.2. Recorrer ficheros y Leer su Contenido

En este apartado vamos a explicar cómo se recorre un fichero en PHP. Muchas veces es necesario mover el puntero de ficheros a una posición determinada desde otra previa para leer su contenido o para modificarlo o insertar algún texto. PHP tiene también funciones que permiten mover el puntero por el contenido de los ficheros.

9.2.1. Recorrer un fichero

La función **rewind()** sirve para colocar el puntero de acceso a los ficheros en la primera posición. Ya hemos visto que, cuando se abre un fichero, algunos parámetros de su argumento *mode* colocan el puntero en la primera posición, pero, a lo largo de diversas operaciones con el mismo, puede suceder que no sepamos dónde está el puntero. Para colocarlo en la primera posición, debemos usar **rewind()**. Para poder realizar esta operación, lógicamente, el fichero debe estar abierto.

Su sintaxis es la siguiente:

```
rewind("nombre completo del fichero" o identificador);
```

Por ejemplo, la instrucción `rewind("Ficheros_1.php");` coloca el puntero de ese fichero en la primera posición.

La función **fseek()** se utiliza para mover el puntero tantas posiciones hacia delante (si el segundo argumento es un número positivo) o hacia atrás (si es un número negativo) como se indique en su segundo argumento. Tiene una sintaxis también sencilla:

```
fseek("nombre completo del fichero" o identificador,
       número de posiciones que hay que saltar);
```

Por ejemplo, la instrucción `fseek("Ficheros_1.php",75);` coloca el puntero del fichero en el carácter (byte) que ocupa la posición 75.

La función **ftell()** se utiliza para conocer en qué posición del fichero está el puntero. Tiene esta sintaxis:

```
ftell("nombre completo del fichero" o identificador);
```

Devuelve un número entero y se usa cuando se desconoce la posición del puntero. Suele combinarse con la función **`fseek()`** para desplazar el puntero de forma relativa.

Por ejemplo, la instrucción *`ftell("Ficheros_1.php");`* escrita después de *`fseek("Ficheros_1.php",75);`* devuelve el valor *75*.

Si queremos mover el puntero de forma relativa diez posiciones adelante, escribiremos la instrucción

```
fseek("Ficheros_1.php",ftell("Ficheros_1.php")+10);
```

La función **feof()** se utiliza para detectar si se ha sobrepasado la última posición del fichero, es decir, si se ha alcanzado la marca de final de fichero. Tiene esta sintaxis:

```
feof("nombre completo del fichero" o identificador);
```

Devuelve el valor lógico *`True`* si se ha alcanzado la marca de final de fichero o *`False`* si no se ha hecho.

Por ejemplo, la instrucción *`feof("Ficheros_1.php");`* escrita después de *`fseek("Ficheros_1.php",5000);`* devuelve el valor *True*, ya que el fichero tiene menos de *5.000* caracteres (bytes) y, por tanto, se encuentra la marca de final de fichero.

Esta función es muy útil para leer uno a uno o línea a línea todos los caracteres de un fichero usando un bucle *`while (!feof())`*, como estudiaremos más adelante.

9.2.2. Leer los contenidos de un fichero

Hasta ahora sólo hemos accedido a un fichero y nos hemos movido por sus caracteres, posiciones o bytes, pero en ningún momento hemos visto o modificado su contenido. Veamos ahora cómo podemos leer un fichero.

La función **fread()** se utiliza para leer una cadena de un fichero abierto. Tiene esta sintaxis:

```
fread("nombre completo del fichero" o identificador,
       número de caracteres que se deben leer);
```

Si se alcanza la marca de final de fichero (**`feof()`**) antes de leer todos los caracteres indicados, se lee hasta el final del mismo, si bien no se produce ningún error al llegar al final de fichero.

Por ejemplo, la instrucción *`fread("Ficheros_3.txt",25)`* devuelve los *25* primeros caracteres de este fichero, pues el puntero después de abrir el fichero con el

parámetro **"r"** estaba al comienzo del mismo. Además, el puntero se desplaza a la posición *26*, por cual una segunda lectura parte desde esta posición.

La función **fgets()** hace exactamente lo mismo que la función **fread()** y lleva los mismo parámetros como argumentos. Sólo se diferencian en que la función **fgets()** sólo lee una cadena que, como máximo, abarca hasta que encuentre la marca de final de línea (retorno de carro). Por ello, si el número del segundo parámetro es superior a los caracteres de una línea, la cadena que se lee comprenderá sólo el texto de la línea.

Con esta función incluida dentro de un bucle podemos leer línea a línea un fichero de esta forma:

```
rewind("Ficheros_3.txt");
while (!feof("Ficheros_3.txt"))
{
   $linea=fgets("Ficheros_3.txt",256);
   echo "<B>$linea </B><P>";
}
```

Conviene advertir que, cuando se ejecutan las funciones anteriores, la cadena contiene una posición menos que la indicada en el argumento, es decir, si ponemos 10 como longitud de la cadena, se muestran 9 caracteres. Esto se debe a los códigos de salto de línea, que se leen también, aunque no se muestran.

La función **fgetss()** hace exactamente lo mismo que la función **fgets()**, pero en la lectura prescinde de las etiquetas propias del lenguaje HTML. Su sintaxis es la siguiente:

```
fgetss("nombre completo del fichero" o identificador,
        número de caracteres que se deben leer,
       "etiquetas que pueden leerse");
```

La función **fgetc()** se utiliza para leer un carácter de un fichero abierto a partir de la posición del puntero. Tiene esta sintaxis:

```
fgetc("nombre completo del fichero" o identificador);
```

Por ejemplo, la instrucción *`fgetc("Ficheros_3.txt")`* devuelve el carácter de este fichero sobre el que esté el puntero. Éste también se desplaza una posición.

Usando esta función dentro de un bucle que recorra desde la primera posición hasta la marca de final de fichero puede mostrarse el contenido completo de éste.

La función **file()** también se usa para leer un fichero y asignar el texto de cada línea a una matriz. La primera línea se coloca como elemento *0* y así sucesivamente. En este caso el identificador que se pasa como argumento es el propio nombre del fichero y la función **`count()`** permite saber cuántos elementos tiene la matriz para poder recorrerla posteriormente. Veamos un ejemplo:

```
$matriz=file("Ficheros_3.txt");
for ($i=0;$i<count($matriz);$i++)
{
   print ("<B> Elemento $i:</B> $matriz[$i]<P>");
}
```

La función **readfile()** permite también leer un fichero y enviar su contenido a la página del cliente. Tiene la siguiente sintaxis:

```
readfile("nombre completo del fichero", camino de la orden include);
```

El segundo argumento es opcional e indica, si se pone, que el fichero se busque también en los directorios citados en la cláusula *include*.

Prácticamente hace lo mismo la función **fpassthru().** Esta función devuelve `True` si se ha podido realizar la operación y `False` si no se ha hecho.

9.3. Modificar el contenido de los ficheros

En este apartado se va a explicar cómo se modifica el contenido de los ficheros en PHP. Este lenguaje dispone también de funciones que permiten modificar el contenido de los ficheros.

La función **fputs()** sirve para escribir en el fichero especificado una cadena de caracteres del tamaño que se indique. Para poder realizar esta operación, lógicamente, el fichero debe estar abierto. Esta función devuelve el valor lógico `True` si se ha podido realizar la operación sin problema o `False` si no se ha podido.

Su sintaxis es la siguiente:

```
fputs("nombre completo del fichero" o identificador,
       cadena que se escribe,tamaño de la cadena);
```

El tercer argumento es opcional. Si no se pone, se escribe la cadena completa. Si se indica, sólo se escribe el número de caracteres señalado.

Para que esta operación funcione correctamente es preciso utilizar bien el modo de apertura del fichero. Si ha de añadirse el texto por el principio del fichero, debemos usar el parámetro de apertura **"r+"**; si necesitamos añadir el texto por el final, usaremos los parámetros de apertura **"a" o "a+"**; si queremos sustituir el contenido completo del fichero, hay que poner los parámetros de apertura **"w"** o **"w+"**.

Por ejemplo, la instrucción `fputs("Ficheros_2.php","Nuevo texto");` suponiendo que el fichero se ha abierto con el parámetro **"r+"**, añade el texto al principio del fichero indicado sustituyendo los caracteres originales que éste ocupe.

La función **fwrite()** es idéntica a **`fputs()`** y su sintaxis lleva los mismos argumentos, que se comportan igual que en la función explicada. Así pues, puede usarse también en lugar de la anterior.

9.4. Copiar, borrar y renombrar ficheros

En este apartado se va a explicar otras operaciones que también pueden llevarse a cabo en PHP con los ficheros: copiarlos, renombrarlos, borrarlos y conocer sus atributos.

9.4.1. Copiar un fichero

La función **copy()** sirve para hacer una copia física de un fichero en otro. Para poder realizar esta operación, el fichero original **no** debe estar abierto. Esta función devuelve el valor lógico *True* si se ha podido realizar la operación sin problemas o *False* si no se ha podido.

Su sintaxis es la siguiente:

```
copy("nombre del fichero original", "nombre del fichero destino");
```

Por ejemplo, *copy("Ficheros_2.txt","Ficheros2_2.txt");* hace una copia exacta del primero en el segundo.

Si el fichero destino existe, es sustituido automáticamente. Por eso, conviene combinar la función **copy()** con la función **file_exists(),** que detecta si un fichero existe, para no sustituir involuntariamente el contenido de un fichero con el de otro. Esta función devuelve el valor lógico *True* si el fichero existe o *False* si no existe.

```
file_exists(“nombre del fichero”)
```

La función **unlink()** sirve para borrar físicamente un fichero. Para poder realizar esta operación, el fichero original no debe estar abierto. Esta función devuelve el valor lógico *True* si se ha podido realizar la operación sin problema o *False* si no se ha podido.

Su sintaxis es la siguiente:

```
unlink("nombre del fichero que debe borrarse");
```

Por ejemplo, la instrucción *unlink("Ficheros_2.txt");* elimina el fichero especificado. Si no se ha fijado con **chdir()** el directorio actual, hay que indicar delante del nombre del fichero el camino completo donde se halla.

Para evitar errores, conviene combinar la función **unlink()** con la función **file_exists()**, como hemos hecho con la función **copy()**.

La función **rename()** sirve para cambiar el nombre de un fichero. Para poder realizar esta operación el fichero original no debe estar abierto. Esta función devuelve el valor lógico *True* si se ha podido realizar la operación sin problemas o *False* si no se ha podido.

Su sintaxis es la siguiente:

```
rename("nombre del fichero original","nuevo nombre del fichero");
```

Por ejemplo, *rename("Nuevo.txt","Ficheros_2.txt");* cambia el nombre del fichero original por el nuevo nombre.

Si ya hay un fichero con el mismo nombre, la operación no se lleva a cabo. Por eso, conviene combinar la función **rename()** con la función **file_exists()**, tanto para

comprobar que existe el fichero original como para comprobar que no existe otro con el nombre que se le quiere poner.

9.4.2. Conocer los atributos, el tipo y el tamaño de un fichero

Las tres operaciones que hemos explicado antes sólo pueden ejecutarse si el fichero, caso de que exista, tiene el atributo de sólo lectura en estado de `False`. Además, es preciso saber otras cosas para poder tratar un fichero, como si existe o no, si el nombre indicado es un de un fichero o de un directorio y el tipo de fichero que es. Todo esto lo vamos a ver en este apartado.

La función **filesize()** devuelve el tamaño de un fichero expresado en número de bytes.

```
filesize (“nombre del fichero”)
```

La función **filetype()** devuelve el tipo del fichero pasado como argumento. Según el nombre pasado como argumento, los tipos que puede devolver esta función son:

- file: es un fichero normal.
- dir: es un nombre de directorio.
- link: es un enlace simbólico (sólo en sistema UNIX).
- fifo: es una pila de tipo FIFO.
- char: es en dispositivo de tipo carácter, por ejemplo “C:\”.
- block: es un dispositivo de bloque.
- unknown: tipo desconocido.

```
filetype (“nombre del fichero”)
```

La función **is_dir()** devuelve `True` si es un directorio y `False` si no lo es o no existe.

```
is_dir(“nombre del fichero”)
```

La función **is_executable()** devuelve `True` si es un fichero ejecutable por el cliente que accede al mismo y `False` si no lo es o no existe.

```
is_executable(“nombre del fichero”)
```

La función **is_file()** devuelve `True` si es un fichero normal y `False` si no lo es o no existe.

```
is_file(“nombre del fichero”)
```

La función **is_link()** devuelve `True` si es un enlace simbólico y `False` si no lo es o no existe.

```
is_link(“nombre del fichero”)
```

La función **is_writeable()** o **is_writable()** devuelve `True` si es un fichero en el que se puede escribir y `False` si no lo es o no existe.

```
is_writeable(“nombre del fichero”)
```

La función **is_readable()** devuelve *True* si es un fichero que se puede leer y *False* si no lo es o no existe.

```
is_readable(“nombre del fichero”)
```

La función **stat()** devuelve una matriz con 13 elementos que contienen información sobre el fichero, si existe. Concretamente, se informa de los siguientes aspectos: dispositivo, i-nodo, permisos, número de enlace, propietario, grupo, tipo de dispositivo, tamaño, instante del último acceso, instante de la última modificación, instante del último cambio, tamaño del bloque y número de bloques asignados. Como se ve, es una información más bien adecuada para el administrador de un servidor. Además, algunas informaciones sólo son propias del sistema Linux, por lo que Windows devuelve el valor –1.

```
stat(“nombre del fichero”)
```

9.5. Operaciones con directorios

PHP permite tratar también directorios: crearlos, eliminarlos, fijar alguno por defecto, así como subir ficheros desde el cliente al servidor, usarlos y validarlos, etcétera. En este apartado vamos a estudiar estas funciones.

9.5.1. Establecer el directorio por defecto

La función **chdir()** establece el directorio actual por defecto donde están archivados o donde se van a archivar los ficheros que se utilicen para realizar diferentes operaciones. Al ejecutarse, comprueba si el directorio pasado como argumento existe o no. En el primer caso esta función devuelve *True*; en el segundo, *False*. Una vez que se ha establecido el directorio por defecto, si el camino especificado existe, ya no es necesario indicar delante del nombre de fichero el directorio donde debe buscarse o crearse los ficheros que se usen en el mismo script.

Su sintaxis es la siguiente:

```
chdir(“nombre completo de directorio”);
```

La mejor forma de usar esta función es incluyéndola dentro de una estructura condicional de la forma siguiente:

```
if (chdir("C:\\WAMP\\WWW"))
         echo "<B>El directorio\"C:\\WAMP\\WWW\"
               existe y ha quedado fijado como actual.</B>";
       else
         echo "<B>El directorio\"C:\\WAMP\\WWW\"
               no existe y no se ha podido fijar como actual.</B>";
```

9.5.2. Abrir un directorio

La función **opendir()** permite entrar en un directorio del servidor y tener acceso a sus ficheros y subdirectorios. Es imprescindible abrir un directorio para, después,

poder cerrarlo con **closedir()**, leer sus ficheros con **readdir()** o mover el puntero de lectura de un directorio con **rewinddir()**.

Su sintaxis es la siguiente:

```
opendir("camino completo de directorio");
```

Esta función devuelve un identificador de directorio que nos servirá después para hacer referencia a ese directorio, como ocurre con los ficheros.

Por ejemplo, *opendir("Capitulo_09");* abre este directorio. Para que nos devuelva un identificador que necesitaremos después, lo mejor es escribir:

```
$id_dircurso=opendir("Capitulo_09");
```

Además, como hemos hecho en los ficheros, recomendamos la siguiente estructura, ya usada y explicada en los ficheros:

```
$id_dircurso = @opendir("Capitulo_09")
               or die("<B>El directorio \" Capitulo_09\"
                        no se ha podido abrir.</B><P>");
```

Es importante tener en cuenta que, al escribir el camino, la barra \ debe escribirse doble, si bien recomendamos usar mejor la barra única /.

PHP dispone también de una clase que nos permite acceder y tratar los directorios. Es la clase **dir().** Para abrir un directorio funciona igual que **`opendir()`**. Además, tiene dos propiedades, que son handle (puntero) y path (camino), y tres métodos, que son **`read()`** (leer el nombre de un fichero o subdirectorio), **`rewind()`** (llevar el puntero al principio de directorio) y **`close()`** (cerrar el directorio). Dentro de esta clase pueden usarse, además, las funciones estándar de manejo de directorios.

```
<?php
$d = dir("Capitulo_09");
echo "Gestor: " . $d->handle . "\n";
echo "Ruta: " . $d->path . "\n";
while (false !== ($entrada = $d->read())) {
   echo $entrada."\n";
}
$d->close();
?>
```

9.5.3. Cerrar un directorio

La función **closedir()** permite cerrar un directorio del servidor. El directorio debe haber sido abierto previamente.

Su sintaxis es la siguiente:

```
closedir("nombre del identificador de directorio");
```

9.5.4. Leer un directorio

La función **readdir()** permite leer los ficheros o directorios del directorio abierto.

Su sintaxis es la siguiente:

```
readdir("nombre del identificador de directorio");
```

Esta función lee el nombre del fichero o directorio sobre el que esté el puntero y avanza éste al siguiente. Lo normal es usar esta función dentro de un bucle que vaya leyendo uno a uno los elementos (ficheros o subdirectorios) del directorio.

Por ejemplo, podemos recorrer un directorio completo de la forma siguiente:

```
$id_dircurso= @opendir("Capitulo_09")
              or die("<B>Capitulo_09" no se ha podido abrir.</B><P>");
while ($fichero=readdir($id_dircurso))
{
   if (is_dir($fichero))
       print ("$fichero <B>es un directorio.</B><P>");
   else
       print ("$fichero <B>es un fichero.</B><P>");
}
```

La clase **`dir()`** también permite leer un directorio, como hemos explicado antes.

9.5.5. Mover el puntero de lectura de un directorio

La función **rewinddir()** lleva el puntero de lectura de un directorio al principio de éste, para poder leerlo completo si el puntero se había desplazado anteriormente.

Su sintaxis también es sencilla:

```
rewinddir("nombre del identificador de directorio");
```

9.5.6. Crear un directorio

La función **mkdir()** permite crear un directorio dentro del directorio actual. La sintaxis es sencilla:

```
mkdir("nombre del nuevo directorio",entero octal de modo);
```

En UNIX hay que poner un segundo parámetro (número entero en base octal), que indica el modo o permisos de creación. En Windows debemos poner 0 en este parámetro o no ponerlo (es opcional).

Esta función devuelve el valor lógico *`True`* si se ha podido realizar la operación sin problema o *`False`* si no se ha podido. Si no hay un directorio establecido como actual, es preciso poner el camino completo del directorio que se quiere crear.

Por ejemplo, las instrucciones

```
$nuevo_dir="Directorio_Prueba";
mkdir($nuevo_dir,0);
```

crean el directorio “Directorio_Prueba” dentro del directorio actual.

Si el directorio ya existe, se produce un error. Por eso, al igual que ocurre con los ficheros, conviene comprobar con la función **`file_exists()`** si ya existe, para no intentar crearlo en este caso.

9.5.7. Borrar un directorio

La función **rmdir()** permite borrar un directorio. Su sintaxis es también sencilla:

```
rmdir("nombre del directorio que se quiere borrar");
```

Esta función devuelve el valor lógico *True* si se ha podido realizar la operación sin problema o *False* si no se ha podido. Si no hay un directorio establecido como actual, es preciso poner el camino completo del directorio que se quiere eliminar. Para poder borrarse, el directorio debe existir y estar vacío.

Por ejemplo, las instrucciones

```
$directorio="Directorio_Prueba";
rmdir($directorio);
```

eliminan el directorio "Directorio_Prueba" dentro del directorio actual.

Si el directorio no existe o no está vacío, se produce un error. Por eso, al igual que ocurre con los ficheros, en esta operación conviene comprobar con la función **file_exists()** si existe, con el signo **@** para evitar el mensaje si no existe y con **die()** para mostrar un mensaje propio si existe, pero no está vacío. Ésta sería una forma elegante de hacerlo:

```
$directorio="Pruebas";
if (file_exists($directorio)) @rmdir($directorio)
     or die("<B>El directorio \"Pruebas\" no se ha podido borrar
              al no estar vacío.</B><P>");
```

Si el problema consiste en que no está vacío, hay que eliminar todos los ficheros con la función **unlink()** antes de intentar borrar el directorio.

9.5.8. Subir ficheros al servidor, usarlos y validarlos

Con PHP el cliente de una página web también puede subir ficheros al servidor desde su ordenador, es decir, se puede convertir en emisor de información.

La forma más frecuente de hacerlo es rellenando formularios con la información que el web le pida, sea ésta personal, profesional o económica, a fin de conseguir algo, como obtener datos sobre algo que le interesa, comprar algún producto, apuntarse en un servicio, etcétera. Los datos que el usuario envía al servidor suelen ser de tipo texto. Cuando se reciben en su destino, se procede con ellos como convenga según la finalidad de la aplicación.

Pero también es posible en PHP enviar ficheros al servidor, en formato texto o binario. Por ejemplo, se puede remitir un fichero con el currículum vitae o una fotografía, etcétera.

Vamos a estudiar los pasos que se dan cuando se sube un fichero al servidor:

1. Mediante un formulario se pregunta el nombre del fichero. Explicamos brevemente el código, que está compuesto exclusivamente por etiquetas HTML.

```
<FORM ENCTYPE="multipart/form-data" ACTION="recibir.php" METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="51200">Fichero: <INPUT
NAME="fichero" TYPE="file">
```

```
<INPUT TYPE="submit" VALUE="Enviar">
</FORM>
```

Las etiquetas HTML nuevas son:

- `ENCTYPE=multipart/form-data` para poder examinar directorios al seleccionar el fichero que se quiere subir.
- `TYPE=File` para indicar que el tipo de dato que se va a pasar al script `recibir.php` es un fichero.
- `TYPE=\"hidden\"name=\"MAX_FILE_SIZE\" VALUE=\"51200\` para definir una entrada oculta (hidden) al usuario de la página en la que se fija que el tamaño máximo (`MAX_FILE_SIZE`) del fichero debe ser de 50 Kb. Debe estar antes del imput de tipo File.

Cuando se selecciona un fichero y se pulsa sobre el botón `"Enviar"`, de forma automática se pasa el fichero al servidor.

2. Recepción de fichero en el servidor. Al ejecutarse el script *`recibir.php`*, la matriz superglobal `$_FILES['fichero']` contiene los datos del fichero que se ha subido al servidor. El servidor recibe este fichero en un directorio temporal, especificado en la variable de entorno `TMPDIR`. Normalmente, este directorio es `c:\Apache\Php`. Al fichero se le pone un nombre que se inicia por php, seguido de unos números y de la extensión tmp. Por ejemplo, `Php1001.tmp`. Además, de forma automática, se crean una serie elementos en la matriz, que nos informan de los siguientes datos:

- `$_FILES['fichero']['name']`: nombre original del fichero que se pasa.
- `$_FILES['fichero']['size']`: tamaño del fichero que se pasa.
- `$_FILES['fichero']['type']`: tipo del fichero que se pasa.
- `$_FILES['fichero']['tmp_name']`: nombre del fichero temporal.

Pero el fichero temporal permanece sólo unos instantes en el directorio temporal del servidor, por lo cual hay que copiarlo al directorio donde queremos que se mantenga.

3. Copia del fichero desde el directorio temporal hasta el directorio destino. Al llevar a cabo esta operación es el momento de hacer los controles que creamos oportunos para que no se copie un fichero de cualquier tipo, ni se sustituya otro que tuviera el mismo nombre, ni se sobrepase una determinado tamaño, etcétera.

Para copiar el fichero, podemos usar las instrucciones

```
if (is_uploaded_file($_FILES['fichero']['tmp_name'])) {
    copy($_FILES['fichero']['tmp_name'], "/destino/fichero");
} else {
    echo "Posible ataque. Fichero: " . $_FILES['fichero']['name'];
}
/* ...o... */
move_uploaded_file($_FILES['fichero']['tmp_name'], "/destino/fichero");
```

En cuanto a los controles anteriores a la copia, podemos hacerlos así:

Para evitar que se copie un fichero de tamaño mayor de 50 Kb o que no ha sido elegido en el formulario al enviar la variable `$fichero` vacía, podemos escribir:

```
if ($_FILES['fichero']['size']>51200 or $_FILES['fichero']['size']<1)
{
     die ("<B>El fichero ocupa más de 51.200 bytes o no ha indicado
           su nombre en el formulario. No puede copiarse.</B>");
}
```

Para copiar ficheros de cualquier tipo, podemos escribir:

```
if ($subir_cualquiera=="SI")
{
   $lugar=$directorio. $_FILES['fichero']['name'];
   move_uploaded_file($fichero, $lugar);
}
```

4. Destrucción del fichero temporal. Aunque el servidor borra automáticamente en pocos segundos el fichero temporal, es conveniente eliminarlo también desde el propio programa para evitar que se vayan quedando en ese directorio temporal ficheros que ya no se necesitan. Por ejemplo, el servidor puede funcionar mal, apagarse, etcétera, sin haber llegado a eliminar el fichero temporal. Se hace con la instrucción *`unlink($fichero);`* .

9.5.9. Permisos y propietarios

Cuando se trabaja en una plataforma UNIX, cosa habitual en Internet, PHP dispone de un conjunto de funciones que permiten gestionar los permisos y propietarios de los ficheros y de los directorios:

- **`chgrp()`**: permite cambiar el grupo al que pertenece un fichero.
- **`chmod()`**: permite cambiar los permisos de un fichero.
- **`chwon()`**: permite cambiar el propietario de un fichero.
- **`filegroup()`**: devuelve el identificador de un grupo.
- **`fileowner()`**: devuelve el identificador del usuario de un fichero.
- **`fileperms()`**: devuelve los permisos de un fichero.

10. Apéndice HTML

10.1. Introducción

El HTML (Hyper Text Markup Language) es un lenguaje que sirve para escribir hipertexto, es decir, documentos de texto presentado de forma estructurada, con enlaces (links) que conducen a otros documentos o a otras fuentes de información (por ejemplo bases de datos) que pueden estar en la propia máquina o en máquinas remotas de la red. Todo ello se puede presentar acompañado de cuantos gráficos estáticos o animados y sonidos seamos capaces de imaginar.

Por supuesto, la estética de los documentos escritos en HTML no se limita a texto plano; consigue todos los efectos que habitualmente se pueden producir con un moderno procesador de textos: negrita, cursiva, distintos tamaños y fuentes, tablas, párrafos tabulados, sangrías, incluso texto y fondo de página de colores, y muchos más.

Básicamente, la cosa es simple: la pieza clave es el "browser", "navegador", "visualizador" o "cliente" HTML. Todas las codificaciones de efectos en el texto que forman el lenguaje HTML no son más que instrucciones para el visualizador. Partiendo de esto, se entiende el porqué no se ve lo mismo con todos los visualizadores. Depende de cómo estén diseñados y para qué versión de lenguaje estén diseñados.

Actualmente existen multitud de ellos, aunque los más conocidos son el Internet Explorer de Microsoft y el Netscape Navigator de Netscape.

Todo lo que se dice en esta guía funciona correctamente con Netscape, casi todo con el IE, y algunas cosas no se verán con otros visualizadores o se verán mal, sobre todo si son versiones antiguas.

Así pues, en esencia, una página escrita en HTML no es más que texto normal, escrito con cualquier editor, y al que, cuando se le quiere dar algun aspecto especial, como por ejemplo el tamaño de la letra, habrá que acompañar de ciertos códigos para indicar el efecto deseado. A estos códigos se les llama **elementos** del lenguaje.

A las instrucciones que forman el lenguaje HTML se les llama **elementos** . La notación de los elementos consiste en los símbolos < y > que encierran dentro una instrucción.

Los elementos pueden ser llenos o vacíos.

10.1.1. Elementos llenos

Se forman escribiendo la instrucción correspondiente seguida del texto al que se quiere aplicar la instrucción y se termina repitiendo la instrucción pero con una barra inclinada inmediatamente antes de la misma.

Por ejemplo, el elemento **H1** que sirve para dar el máximo tamaño al texto, se escribirá:

```
      <H1> Texto de prueba </H1>
```

y este sería el resultado:

Texto de prueba

Si olvidas poner </H1> todo el resto de la página tendrá el mismo tamaño grande.

10.1.2. Elementos vacíos

Los elementos vacíos se escriben como los llenos, pero no es necesario poner la instrucción repetida al final con una barra. Esto se debe a que estos elementos no producen un efecto sobre el texto. Generalmente se utilizan para separar bloques de texto, y por tanto no es necesario indicar su fin. Empiezan y terminan en el mismo punto.

Por ejemplo, el elemento **<HR>** que sirve para dibujar una linea horizontal en la pantalla, se escribirá:

```
<HR>
```

Y este sería el resultado:

10.1.3. Elementos con argumento

Algunos elementos se escriben con argumento. Es como pasarle parámetros a la instrucción, y se llaman **atributos** del elemento.

Por ejemplo, el elemento **<A>** que sirve para hacer un link (enlace) con otro documento, se escribirá:

```
<A HREF="http://www.miservidor.es/mifichero.htm"> Link de prueba  </A>
```

Este es un elemento lleno donde al atributo HREF se le asigna el valor que aparece entre comillas: "http://www.miservidor.es/mifichero.htm". El texto al que afecta este elemento es **Link de prueba** y realiza un link con el fichero **mifichero.htm** que está en el servidor **www.miservidor.es**

Los elementos pueden escribirse tanto en mayúsculas como en minúsculas. Puede ser preferible la primera opción ya que aporta claridad al documento fuente, y eso se agradece a la hora de hacer correcciones, pero CUIDADO, el valor de algunos atributos hay que escribirlos EXACTAMENTE como deban ser. En el ejemplo anterior no funcionaría el link si escribiéramos *www.miservidor.es* en mayúsculas, eso es un nombre de máquina y sería interpretado como otra máquina.

Los elementos pueden anidarse unos con otros, teniendo cuidado de poner los cierres en el lugar adecuado. Por ejemplo, el elemento **<H1>** combinado con **<I>** que sirve para generar texto en itálica, se escribirá:

```
<H1><I>Texto de prueba </I></H1>
```

y este sería el resultado:

Texto de prueba

10.2. Estructura de un documento HTML

Los documentos escritos en HTML estan estructurados en dos partes diferenciadas: la **HEAD** (cabecera) y el **BODY** (cuerpo).

Si escribes:

```
<HTML>
<HEAD>
<TITLE> Documento de prueba </TITLE>
</HEAD>

<!-- Esto es un comentario-->

<BODY>

<H1> Esto es una "demo" de documento HTML </H1>

Esto es el más sencillo de los documentos HTML.

</BODY>
</HTML>
```

El elemento <HTML> </HTML> no es obligatorio. Solo sirve como identificación del tipo de contenido del fichero para ciertos programas que hacen cambios masivos en muchas páginas a la vez.

Para escribir comentarios en la página (que sólo se ven en el texto fuente, pero no en el visualizador) se utilizará el elemento <!-- -->

Los documentos HTML los puedes escribir con lo que quieras, siempre que los salves en modo **Solo Texto**. Es decir, que si utilizas, por ejemplo, Word de Windows o de MAC, por defecto estos programas salvan en formato Word. Y aunque luego los renombres como documentos **.htm** o **.html**, el formato interno sigue siendo Word.

Tampoco vale dejarlos con la extensión **.txt** que les dejan la mayoría de editores al salvarlos en modo Solo Texto. No serían interpretados correctamente. Por lo tanto si tienen extensión **.txt** (y por supuesto **son** .txt), entonces puedes renombrarlos a .htm, o bien en el momento de guardarlos en formato **Solo Texto** no dejar la extensión por defecto .txt y darle ya directamente la extensión **.htm**.

Algunos procesadores de texto, ya incluyen entre sus tipos el **.htm**. Si es así lo que hacen es eliminar la codificación propia, y convertirla a elementos de HTML. Por ejemplo, si tienes un texto en itálica, automáticamente es convertido al elemento <I>, etc..

10.3. Cabecera (HEAD) de un documento HTML

La **HEAD** es la primera de las dos partes en que se estructura un documento HTML.

En la HEAD reside información acerca del documento, y generalmente no se ve cuando se navega por él. En la HEAD se pone el elemento lleno <**TITLE**> que es una breve descripción que identifica el documento. Es lo que el visualizador se guarda en el "**Bookmarks**", con lo que crea la lista que aparece en la orden "**Go**" de la barra de órdenes y lo que aparece en la esquina superior izquierda cuando se imprime el documento.

No hay que confundir el elemento <TITLE> con el nombre del fichero. Por ejemplo, una página que está contenida en un fichero llamado **head.htm** y el texto de su <TITLE> es: **Head de un documento**. Se escibirá asi:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
</HEAD>
```

Dentro de la HEAD puede utilizarse otro elemento: **META**. Por ejemplo, si se escribe:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="10">
</HEAD>
```

Esto hace que el visualizador vuelva a cargar la página activa al cabo de 10 segundos. También puede hacerse a un servidor. Así:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="10;
URL=http://miservidor/mipagina.htm">
</HEAD>
```

Utiliza esto con precaución. Si sabes que el contenido de la página no va a cambiar, es inútil hacer esto, y si lo haces contra un servidor, puedes sobrecargarlo. Este elemento, sólo tendrá utilidad en casos muy especiales.

Otra opción es forzar la expiración inmediata en la caché del navegador de la página recibida, lo que provoca que la página sea solicitada de nuevo al servidor cada vez, en lugar de cargar la copia que ya existe en la máquina del cliente. Se escribe así:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
<META HTTP-EQUIV="Expires" CONTENT="Tue, 20 Aug 1996 14:25:27 GMT">
</HEAD>
```

Si se pone una fecha ya pasada, como la que hay en el ejemplo, el navegador elimina inmediatamente de la caché la página recibida, y si no es pasada, lo hará en el momento indicado por la misma.

Si tienes interés en que tus páginas aparezcan en los grandes buscadores de Internet, y puedan ser encontradas con facilidad, puedes poner las palabras clave que contiene la página separadas por comas. Por ejemplo:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
<META NAME="keywords" CONTENT="HTML, internet  ">
</HEAD>
```

Este otro sirve para que los buscadores puedan ofrecer un breve resumen de los contenidos de tu página:

```
<HEAD>
<TITLE>Head de un documento </TITLE>
<META NAME="description" CONTENT="Manual para escribir HTML.">
</HEAD>
```

Puedes poner todos los elementos <**META**> que creas necesarios, pero sin repetirlos.

Habrás notado que se ha utilizado la palabra "caché", y tal vez no sepas a qué se refiere. Todos los navegadores, por defecto, siempre que reciben una página de un servidor se hacen una copia de la misma en el disco de tu máquina. Con esto se pretende que si vuelves a solicitar la misma página, en lugar de pedirla de nuevo al servidor, te mostrará la que tiene guardada en el disco. A esta área del disco donde el navegador va poniendo las páginas visitadas, se le denomina **caché**. El tamaño de la caché lo puedes modificar, desde la opciones de configuración del navegador, e incluso puedes darle tamaño cero, con lo que siempre que veas una página, ésta habrá sido solicitada al servidor.

Hay otros elementos que pueden aparecer en la HEAD, como ISINDEX, NEXTID, LINK o BASE, pero son de uso muy especializado y poco corriente, algunos ya en desuso, y ninguno obligatorio (<TITLE> sí lo es).

10.4. Cuerpo (BODY) de un documento

El cuerpo (**BODY**) es la segunda y última de las dos partes en que se estructura un documento HTML. Por supuesto es obligatoria, ya que es aquí donde reside el verdadero contenido de la página, y por tanto, al contrario de la HEAD sí se ve cuando navegamos por ella.

Se escibirá asi:

```
<BODY>

texto texto texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texto texto texto

</BODY>
```

A continuación se analizan los elementos más habituales del lenguaje que deben escribirse en esta parte del documento.

10.4.1. Tamaños y tipos de letra en HTML

Para definir distintos tamaños de letra, en HTML se utiliza el elemento lleno **<Hx>** </Hx> donde x es un número que puede variar entre 1 y 6, siendo 1 el tamaño mayor.

Se escribirán así:

```
<H1> Texto de prueba (H1)</H1>.
<H2> Texto de prueba (H2)</H2>
<H3> Texto de prueba (H3)</H3>
<H4> Texto de prueba (H4)</H4>
<H5> Texto de prueba (H5)</H5>
<H6> Texto de prueba (H6)</H6>
```

y este sería el resultado:

Texto de prueba (H1)
Texto de prueba (H2)
Texto de prueba (H3)
Texto de prueba (H4)
Texto de prueba (H5)
Texto de prueba (H6)

No hay que olvidar poner el cierre </Hx> a cada elemento utilizado. Este tipo de elemento se suele utilizar para escribir encabezamientos, ya que después del cierre automáticamente el visualizador inserta un salto de párrafo.

Por ejemplo: si escribes

```
<H1> Texto en H1 </H1> <H3> Texto en H3 </H3>
```

se verá:

Texto en H1
Texto en H3

Y no una cosa al lado de la otra, como cabría esperar.

Otra forma de cambiar los tamaños de letra es utilizar el elemento **<FONT** > con el atributo **VALOR**, que es un número entre 1 y 7. El valor por defecto del texto es 3, por lo que valor puede ser positivo (+) o negativo (-) respecto a 3. Una gran ventaja de esta notación respecto a la anterior es que no se produce un salto de párrafo después de cada cambio, por lo que pueden hacerse cosas como esta:

```
<FONT SIZE=3>A</font><FONT SIZE=4>A</font><FONT SIZE=5>A</font>

<FONT SIZE=6>A</font><FONT SIZE=7>A</font><FONT SIZE=6>A</font>

<FONT SIZE=5>A</font><FONT SIZE=4>A</font><FONT SIZE=3>A</FONT>
```

Dará como resultado:

AAAAAAAAA

Se puede cambiar el tamaño por defecto (3) de toda la página con el elemento **<BASEFONT SIZE**=valor>. El texto tomará el tamaño indicado por **valor** y lo mantendrá hasta que aparezca otro elemento <BASEFONT SIZE=valor> y lo restaure o lo cambie por otro diferente.

Con la versión 3.0 de Netscape se ha implementado un nuevo atributo del elemento <FONT> que permite elegir tipos de letra entre los varios de que dispone por defecto Windows. Se trata del atributo **FACE**. Este atributo permite forzar el tipo de letra que el diseñador de la página quiere que vea el cliente, sin importar el que por defecto tenga establecido el visualizador.

Si escribes

```
<FONT FACE="arial">Texto de prueba 12345 con tipo ARIAL</FONT>
<FONT FACE="times new roman">Texto de prueba 12345 con tipo TIMES NEW
ROMAN</FONT>
<FONT FACE="courier new">Texto de prueba 12345 con tipo COURIER
NEW</FONT>
<FONT FACE="courier">Texto de prueba 12345 con tipo COURIER</FONT>
<FONT FACE="roman">Texto de prueba 12345 con tipo ROMAN</FONT>
<FONT FACE="small fonts">Texto de prueba 12345 con tipo SMALL
FONTS</FONT>
```

Se verá:

Texto de prueba 12345 con tipo ARIAL
Texto de prueba 12345 con tipo TIMES NEW ROMAN
Texto de prueba 12345 con tipo COURIER NEW
Texto de prueba 12345 con tipo COURIER
Texto de prueba 12345 con tipo ROMAN
Texto de prueba 12345 con tipo SMALL FONTS

Por supuesto, este atributo es compatible con todos los demás ya conocidos, como color y tamaño. Por ejemplo, si escribes

```
<FONT FACE="impact" SIZE=6 COLOR="red">
 Texto de prueba 12345 con tipo IMPACT</FONT>
```

Se verá:

Texto de prueba 12345 con tipo IMPACT

Se pueden hacer todas la combinaciones que se quieran, pero hay que tener presente que si en la máquina cliente no está instalada una determinada fuente, ésta no se verá y en su lugar aparecerá la fuente por defecto del visualizador. No es interesante por tanto, definir tipos raros, que probablemente, no llegarán a verse nunca.

10.4.2. Texto en color

Se puede controlar el color del texto utilizando el elemento <FONT> con el atributo **COLOR=xxx**, donde **xxx** es el nombre en inglés del color que se desea. Hay que tener presente que algunos no se verán o se verán mal si la máquina no soporta 256 colores. Por supuesto, este efecto es anidable con el de tamaño y todos los demás.

Si escribes:

```
<B><FONT COLOR="red">Texto ROJO </FONT>
<br>
<FONT COLOR="blue">Texto AZUL </FONT>
<br>
<FONT COLOR="navy">Texto AZUL MARINO </FONT>
<br>
<FONT COLOR="green">Texto VERDE </FONT>
<br>
<FONT COLOR="olive">Texto OLIVA </FONT>
<br>
<FONT COLOR="yellow">Texto AMARILLO </FONT>
<br>
<FONT COLOR="lime">Texto LIMA </FONT>
<br>
<FONT COLOR="magenta">Texto MAGENTA </FONT>
<br>
<FONT COLOR="purple">Texto PURPURA </FONT>
<br>
<FONT COLOR="cyan">Texto CYAN </FONT>
<br>
<FONT COLOR="brown">Texto MARRON </FONT>
<br>
<FONT COLOR="black">Texto NEGRO </FONT>
<br>
<FONT COLOR="gray">Texto GRIS </FONT>
<br>
<FONT COLOR="teal">Texto TEAL </FONT>
<br>
<FONT COLOR="white">Texto BLANCO </FONT>
<br>
</B>
```

Se verá:

Texto ROJO
Texto AZUL
Texto AZUL MARINO
Texto VERDE
Texto OLIVA
Texto AMARILLO
Texto LIMA
Texto MAGENTA
Texto PURPURA
Texto CYAN
Texto MARRON
Texto NEGRO
Texto GRIS
Texto TEAL

He aquí una combinación de colores y tamaños:
Si escribes:

```
<FONT SIZE=6 COLOR="red">E</FONT><FONT SIZE=4>sto es una </FONT>
<FONT SIZE=6 COLOR="red">P</FONT><FONT SIZE=4>rueba </FONT>
```

Resulta:

Esto es una Prueba

Los colores también se pueden definir en hexadecimal (por ejemplo el rojo es <FONT COLOR=#FF0000>). Cuando son colores básicos, como los del ejemplo de arriba, es más cómodo escribir el nombre aunque sea en inglés, pero cuando se trata de definir un color que "no tiene nombre" no hay más remedio que usar la codificación hexadecimal.

10.4.3. Cambios de párrafo y de línea. Divisiones de texto.

Para definir los saltos de párrafo se utiliza el elemento lleno **<P> </P>** (por **P**aragraph). No suele utilizarse el cierre </P>, ya que el texto continuará normalmente hasta que encuentre otro salto <P>

Se escribirá así:

```
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
Texto 1 Texto 1 <P>
Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
```

y este sería el resultado:

Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1

Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2

Como puede verse, hay un línea en blanco entre los dos bloques. El efecto del elemento **
** (por **BR**eak) es el mismo, pero sin esa línea vacía.

Se escribirá así:

```
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
Texto 1 Texto 1 <BR>
Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
```

y este sería el resultado:

Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2

El elemento <P> admite cuatro atributos: **ALIGN=LEFT** (por defecto), **ALIGN=RIGHT** , **ALIGN=CENTER** y **ALIGN=JUSTIFY**

Se escribirán así:

```
<P ALIGN=LEFT>
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
<P ALIGN=RIGHT>
Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
<P ALIGN=CENTER>
```

```
Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3
Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3
<P ALIGN=JUSTIFY>
Texto 4 tex Texto 4 Texto 4  tex Texto 4 Texto 4  tex Texto 4 Texto 4
Texto 4 Texto 4 Texto 4 Texto 4 Texto 4  tex Texto 4 Texto 4 Texto 4
Texto 4 tex Texto 4 Texto 4  tex Texto 4 Texto 4  tex Texto tex 4 Texto 4
```

y este sería el resultado:

```
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1 Texto 1
    Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
    Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2 Texto 2
  Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3
  Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3 Texto 3
Texto 4 tex Texto 4 Texto 4 tex Texto 4 Texto 4 tex Texto 4 Texto
4 Texto 4 Texto 4 Texto 4 Texto 4 Texto 4 tex Texto 4 Texto 4
Texto 4 Texto 4 tex Texto 4 Texto 4 tex Texto 4 Texto 4 tex Texto
tex 4 Texto 4
```

En el caso de utilizar alguno de estos atributos, es recomendable usar el cierre **</P>**

Un elemento que se comporta de forma parecida a <BR> es **<DIV>** pero que admite los mismos atributos que <P>: **ALIGN=LEFT ALIGN=RIGHT** y **ALIGN=CENTER**

Se escribe así:

```
<DIV ALIGN=LEFT>
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
</DIV>
<DIV ALIGN=RIGHT>
texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
< /DIV>
<DIV ALIGN=CENTER>
texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
</DIV>
```

Fíjate en que aquí sí se utiliza el cierre **</DIV>**. Este sería el resultado:

```
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
texto1 texto1 texto1 texto1 texto1 texto1 texto1 texto1
     texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
     texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
     texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
     texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
            texto2 texto2 texto2 texto2 texto2 texto2 texto2 texto2
  texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
  texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
  texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
```

```
texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
   texto3 texto3 texto3 texto3 texto3 texto3 texto3 texto3
```

Otro interesante efecto es el elemento lleno **<BLOCKQUOTE>** que sirve para presentar párrafos adentrados (como si estuviesen tabulados).

Se escribirá así:

```
<BLOCKQUOTE>
texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texto
<BLOCKQUOTE>
texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texto
</BLOCKQUOTE>
</BLOCKQUOTE>
```

Y este sería el resultado:

```
texto texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto
      texto texto texto texto texto texto texto texto texto texto
      texto texto texto texto texto texto texto texto texto texto
```

Fíjate que en este ejemplo hay un anidamiento, y por tanto, debe haber dos cierres **</BLOCKQUOTE>** al final

Otro separador de bloques de texto es el elemento vacío **<HR>** (por **H**orizontal **R**ule). Este sería el resultado:

Se puede cambiar su apariencia añadiéndole el atributo **<NOSHADE>**. Así:

```
<HR NOSHADE>
```

El resultado es:

El elemento <HR> admite dos parámetros: **WIDTH** y **SIZE**. El primero define la longitud de la línea y el segundo su anchura. No es obligado usar los dos a la vez Por ejemplo, si escribes

```
<HR WIDTH=400 SIZE=5>
```

El resultado será:

El valor del atributo WIDTH se puede expresar, como en el ejemplo anterior, en número de puntos (píxels), o en tantos por ciento referidos al ancho total de la ventana. Así:

```
<HR WIDTH=80% SIZE=5>
```

El resultado será:

Además se puede indicar su posición respecto a los márgenes de la ventana con los atributos **ALIGN=CENTER** (por defecto) **ALIGN=LEFT** y **ALIGN=RIGHT**. Por ejemplo:

```
<HR WIDTH=80% SIZE=5 ALIGN=LEFT>
```

El resultado será:

O bien:

```
<HR WIDTH=80% SIZE=5 ALIGN=RIGHT>
```

El resultado será:

Hay otro elemento, aparecido en la versión 6 de Netscape, que permite rodear un texto con un marco, y opcionalmente ponerle una etiqueta. Se trata del elemento: **<FIELDSET>**

Si se escribe:

```
<FIELDSET>
   Esto es una prueba de enmarcado
</FIELDSET>
```

Se obtiene:

Esto es una prueba de enmarcado

Este elemento tiene un parámetro que permite etiquetar el recuadro: **<LEGEND>** Si se escribe:

```
<FIELDSET>
         <LEGEND>Esto es una etiqueta</LEGEND>
   Esto es una prueba de enmarcado
</FIELDSET>
```

10.4.4. Otros efectos en el texto

Una o varias líneas de texto, pueden estar centradas respecto a los márgenes de la ventana. Esto afecta sólo al texto que hay entre **<CENTER>** y **</CENTER>**, pero no a todo el párrafo. Por ejemplo:

```
texto texto texto texto texto texto texto texto texto texto texto texto
```

```
<CENTER>texto centrado texto centrado</CENTER>
texto texto texto texto texto texto texto texto texto texto texto texto
```

Se verá:

```
texto texto texto texto texto texto texto texto texto txto texto texto
texto texto texto texto texto
                    texto centrado texto centrado
texto texto texto texto texto texto texto texto texto txto texto texto
texto texto texto texto texto
```

Esto, además de a texto sólo, se puede aplicar a una tabla, una imagen o cualquier otra cosa.

Quedan algunos otros elementos que modifican el aspecto del texto. Algunos, aparentemente, hacen la misma cosa, y otros no funcionan con el visualizador de Netscape, por lo que se omiten aquí.

Texto en negrita:

```
<B>Texto en negrita</B>
```

Texto realzado:

```
<STRONG>Texto realzado</STRONG>
```

Texto en itálica:

```
<I>Texto en itálica</I>
```

Texto con énfasis:

```
<EM>Texto con énfasis</EM>
```

`Texto ejemplo de código:`

```
<CODE>Texto ejemplo de código</CODE>
```

Texto teletipo:

```
<TT>Texto teletipo</TT>
```

Texto subrayado:

```
<U>Texto subrayado</U>
```

~~Texto tachado:~~

```
<STRIKE>Texto tachado</STRIKE>
```

Texto de dirección:

```
<ADDRESS>Texto de dirección</ADDRESS>
```

Texto intermitente

```
<BLINK>Texto intermitente</BLINK>
```

Texto superíndice Texto normal

```
<SUP>Texto Superíndice</SUP>
```

Texto subíndice Texto nomal

```
<SUB>Texto Subíndice</SUB>
```

Texto grande

```
<BIG>Texto grande</BIG>
```

Texto pequeño

```
<SMALL>Texto pequeño</SMALL>
```

10.4.5. Listas y menús

Hay elementos que permiten crear texto con varios formatos de listado: Estos pueden ser **ordenados** <OL> (se refiere a numerados, no ordenados por algun criterio), **desordenados** <UL> (no numerados), **directorios** <DIR>, **menu** <MENU> y **listados de definición** <DL>. Veamos comó es la sintaxis básica y apariencia de estos elementos, a partir de los cuales, pueden hacerse combinaciones muy complejas

mediante anidamientos de unos con otros, hasta conseguir prácticamente cualquier presentación deseada:

Esto es una **lista ordenada** (numerada):

1. Primera linea
2. Segunda linea

Se escribe:

```
<OL>
<LI>Primera linea
<LI>Segunda linea
</OL>
```

Fíjate en que los elementos <LI> no tienen cierre. Terminan cuando aparece otro igual o se cierra la lista.

Esto es una **lista desordenada** (no numerada):

- Primera linea
- segunda linea

Se escribe:

```
<UL>
<LI>Primera linea
<LI>Segunda linea
</UL>
```

En este caso los números han sido sustituidos por unos puntos gruesos. Esa es la apariencia por defecto; se puede cambiar usando el atributo **TYPE** con tres valores (el valor por defecto es **DISC**). Con el valor **CIRCLE** se verá:

- Primera linea
- segunda linea

Se escribe:

```
<UL TYPE=CIRCLE>
<LI>Primera linea
<LI>Segunda linea
</UL>
```

También puede usarse el valor **SQUARE**. Así:

- Primera linea
- segunda linea

Se escribe:

```
<UL TYPE=SQUARE>
<LI>Primera linea
<LI>Segunda linea
</UL>
```

Las listas ordenadas no sólo se pueden ordenar con números. También se pueden utilizar letras y numeración romana tanto en mayúsculas como minúsculas. Para esto se utiliza el atributo TYPE del elemento <OL> con los siguientes valores: **TYPE=1** (por defecto) para números, **TYPE=A** para letras mayúsculas, **TYPE=a** para letras minúsculas, **TYPE=I** para numeración romana en mayúsculas y **TYPE=i** para numeración romana en minúsculas.

Esto es una **lista ordenada con letras mayúsculas**:

A. Primera linea
B. Segunda linea
C. Tercera linea
D. Cuarta linea

Se escribe:

```
<OL TYPE=A>
<LI>Primera linea
<LI>Segunda linea
<LI>Tercera linea
<LI>Cuarta linea
</OL>
```

Esto es una **lista ordenada con letras minúsculas**:

a. Primera linea
b. Segunda linea
c. Tercera linea
d. Cuarta linea

Se escribe:

```
<OL TYPE=a>
<LI>Primera linea
<LI>Segunda linea
<LI>Tercera linea
<LI>Cuarta linea
</OL>
```

Esto es una **lista ordenada con numeración romana en mayúsculas**:

I. Primera linea
II. Segunda linea
III. Tercera linea
IV. Cuarta linea

Se escribe:

```
<OL TYPE=I>
<LI>Primera linea
<LI>Segunda linea
<LI>Tercera linea
<LI>Cuarta linea
</OL>
```

Esto es una **lista ordenada con numeración romana en minúsculas**:

i. Primera linea

ii. Segunda linea

iii. Tercera linea

iv. Cuarta linea

Se escribe:

```
<OL TYPE=i>
<LI>Primera linea
<LI>Segunda linea
<LI>Tercera linea
<LI>Cuarta linea
</OL>
```

En algunos casos puede interesarnos que la lista no comience por el número 1 (por ejemplo si es una lista que continúa en otra página). Se puede conseguir con el atributo **START** combinado con **TYPE**.

Esto es una **lista ordenada con letras mayúsculas y que comienza por la E**:

E. Primera linea

F. Segunda linea

G. Tercera linea

H. Cuarta linea

Se escribe:

```
<OL TYPE=A START=5>
<LI>Primera linea
<LI>Segunda linea
<LI>Tercera linea
<LI>Cuarta linea
</OL>
```

El número que pones en el atributo START indica en que número o letra comenzará la lista. La E es la quinta letra.

Esto es un **menú**:

- Primera linea
- Segunda linea

Se escribe:

```
<MENU>
<LI>Primera linea
<LI>Segunda linea
</MENU>
```

La diferencia entre un menú y una lista desordenada, es que las líneas en la lista desordenada comienzan en el margen izquierdo y las del menú unas posiciones más a la derecha (aunque eso depende del visualizador, con Netscape se ven igual).

Esto es un **directorio**:

- Primera linea
- Segunda linea

Se escribe:

```
<DIR>
<LI>Primera linea
<LI>Segunda linea
</DIR>
```

Ocurre lo mismo que con el menú, con Netscape no se aprecia diferencia.

Esto es una **lista de definicion**:

Primera linea
    Segunda linea

Se escribe:

```
<DL>
<DT>Primera linea
<DD>Segunda linea
</DL>
```

Fíjate que ahora en lugar del elemento <LI> se utiliza <**DT**> y <**DD**>, y como tipo **<DL>**. Estos tres nuevos elementos también se pueden usar con cualquiera de los anteriores tipos de lista, alterando por completo su apariencia. Para hacerse una idea de la variedad de aspectos que se pueden conseguir ver los ejemplos de listados.

```
<DL>
    <DT>Linea de texto 1
    <DT>Linea de texto 2
    <DT>Linea de texto 3
        <DL>
            <DT>Linea de texto 3.1
            <DT>Linea de texto 3.2
                <DL>
                    <DT>Linea de texto 3.2.1
                    <DT>Linea de texto 3.2.2
                       <DL>
                           <DT>Linea de texto 3.2.2.1
                           <DT>Linea de texto 3.2.2.2
                       </DL>
                </DL>
         </DL>
     <DT>Linea 4 de texto
</DL>
```

```
Linea de texto 1
Linea de texto 2
Linea de texto 3
          Linea de texto 3.1
          Linea de texto 3.2
                      Linea de texto 3.2.1
```

```
                        Linea de texto 3.2.2
                                    Linea de texto 3.2.2.1
                                    Linea de texto 3.2.2.2
Linea 4 de texto
```

10.4.6. Tablas

Las tablas son sin duda uno de los elementos más potentes del HTML. Con pocos elementos se pueden conseguir efectos espectaculares. En el interior de las celdas de una tabla, que pueden ser con borde visible o invisible, se puede poner cualquier cosa: texto de cualquier tamaño y color, imágenes, links... Por supuesto, además de permitir cualquier contenido, tienen sus propios atributos de alineación tanto horizontal como vertical, y atributos de dimensionado. Por defecto se autodimensionan, es decir, se adaptan en tamaño a su contenido, pero también es posible definirlo de forma fija.

El elemento básico de definición de tabla es <**TABLE**> </**TABLE**> y en su interior se disponen los sub elementos <**TR**> para definir una fila (**R**ow), <**TH**> para definir una cabecera (**H**eader), <**TD**> para definir una celda de datos (**D**ata). Estos sub elementos también han de llevar sus correspondientes cierres: </**TR**> </**TH**> <**TD**>.

Una cabecera <**TH**> es los mismo que una celda de datos <**TD**> pero de forma automática el texto de su contenido recibe los atributos de negrita y centrado. Sólo es posible definirlas al principio de las filas, de las columnas o de ambas a la vez.

He aquí un resumen de los elementos utilizados y los atributos que admite cada uno:

	TABLE	TR	TD	TH	CAPTION
BORDER	X	-	-	-	-
ROWSPAN	-	-	X	X	-
COLSPAN	-	-	X	X	-
ALIGN	-	X	X	X	X
VALIGN	-	-	X	-	-
WIDTH	X	-	X	-	-
HEIGTH	X	-	X	-	-
CELLPADDING	X	-	-	-	-
CELLSPACING	X	-	-	-	-

Resumen de tablas

10.4.7. Códigos hexadecimales de color

Estos códigos de color se pueden utilizar como valor de atributo en los elementos <FONT COLOR> y <BODY BGCOLOR FGCOLOR TEXT LINK VLINK ALINK>. Algunos no funcionarán en ciertos visualizadores, así como si Windows no está configurado con 256 colores al menos.

La codificación de estos colores está basada en mezclas cromáticas RGB (RED GREEN BLUE), es decir, se mezclan ciertas cantidades de los tres colores básicos (rojo verde y azul), para conseguir el color deseado. La cantidad de cada color está definida por el código que utilizamos, que tiene seis dígitos en hexadecimal. Si los cortamos en 3

grupos de dos de izquierda a derecha, obtendremos la cantidad (en hexadecimal) de cada color.

La sintaxis para cambiar el fondo de la página (background) a color rojo es: <BODY BGCOLOR=#FF0000> y para dar color al texto, ver la página Texto de colores. Algunos de los mas usados son

White	rgb=#FFFFFF
Red	rgb=#FF0000
Green	rgb=#00FF00
Blue	rgb=#0000FF
Magenta	rgb=#FF00FF
Cyan	rgb=#00FFFF
Yellow	rgb=#FFFF00
Black	rgb=#000000
Aquamarine	rgb=#70DB93
Baker's Chocolate	rgb=#5C3317

10.4.8. Creación de enlaces (links)

El siguiente es sin duda el elemento más importante del HTML, ya que permite realmente "navegar" por uno o varios documentos, que pueden residir en cualquier parte, pareciéndonos que siempre estamos en el mismo; a esto se le llama **hipertexto** ya que con esta posibilidad, en realidad, nuestro documento puede ser infinito... En efecto, para la persona que está leyendo sobre un determinado tema, no hay diferencias ostensibles que le hagan notar dónde está el documento que lee. Para el lector, todo parece un mismo documento, cuando en realidad, probablemente el conjunto de lo que lee está repartido por medio mundo, o en un plano más modesto, el documento leído puede estar compuesto en realidad por varios cientos de páginas que "saltan" de unas a otras sin notarlo.

Todo esto lo consigue el elemento <**A**> (por **A**nchor, en inglés = ancla o punto de anclaje). En lo sucesivo le llamaremos enlace o sinplemente **link** (en inglés link=eslabón o enlace).

Pero seguramente te preguntarás qué es eso de un "link". Pues un link es un área de la pantalla, que puede contener una o varias palabras o una imagen, que es "sensible" al puntero del ratón, y al ponerlo sobre ella y pulsar el botón izquierdo el visualizador llamará a la página que tiene asignada el link. Habitualmente por defecto los visualizadores señalan un área linkada subrayándola, si es texto, o poniéndole un borde si es una imagen, ambas cosas en color azul. Por supuesto, eso se puede cambiar en las configuraciones de visualizador. Si no se desea que aparezca el subrayado para mejorar la estetica en algunos casos, se puede parametrizar con una instruccion de estilo.

Se escribirá:

```
<A HREF="http://www.miservidor.es/mifichero.htm"> Link de prueba  </A>
```

Y se vería así: Link de prueba

Si lo que se desea es que aparezca sin subrayado o de otro color, o ambas cosas, se puede recurrir a una instruccion de estilo. Así:

```
<A HREF="http://www.miservidor.es/mifichero.htm"
   style="color:red; text-decoration:none";> Link de prueba  </A>
```

Y se vería así: Link de prueba (Esto puede que no funcione en algunos navegadores antiguos)

El elemento **<A>** tiene dos atributos: **HREF** y **NAME**.

En el ejemplo de arriba al atributo HREF se le asigna el valor que aparece entre comillas: **"http://www.miservidor.es/mifichero.htm"**. El texto sobre el que se monta el enlace es **Link de prueba** y realiza un link con el fichero **mifichero.htm** que está en el servidor **www.miservidor.es**.

Esta forma de escribir la ruta del link de forma completa, se utiliza cuando hay que saltar de una máquina a otra, si el enlace es para otra página de nuestro propio servidor es suficiente escribir la ruta virtual corta: **/mifichero.htm** o bien: **/otro_directorio/mifichero.htm** según proceda.

Como has visto, el punto de enlace se realiza en el texto escrito antes del cierre del elemento **</A>**, pero también puede hacerse con una imagen cualquiera, que iría en lugar del texto, incluso puedes poner el texto junto con una imagen. Por ejemplo para hacer un link que nos lleve al índice desde una bola roja, se escribe:

```
Ir al índice <A HREF="indice.htm"> <img src="bolaroja.gif"> </A>
```

Y este sería el resultado: Ir al índice 

Al igual que el texto aparece subrayado en azul cuando forma parte de un enlace, los gráficos reciben un borde azul alrededor de toda la imagen, lo que a veces puede resultar poco estético. Esto también tiene solución, pero aquí se hace en la instrucción de la imagen:

```
Ir al índice <A HREF="indice.htm"> <img src="bolaroja.gif" border=0>
</A>
```

Y este sería el resultado: Ir al índice 

El valor de los atributos hay que escribirlos EXACTAMENTE como se deba. En el ejemplo anterior no funcionaría el link si escribiéramos **www.miservidor.es** en mayúsculas o con alguna otra diferencia, eso es un nombre de máquina y sería interpretado como OTRA máquina. Esto es aplicable a todos los parámetros: **servicio** (http://) **servidor** (www.miservidor.es) y **fichero** (mifichero.htm).

Estas precauciones son ineludibles en el caso de servidores montados en máquinas con sistema operativo UNIX. En el caso de que el servidor resida en una máquina con Windows NT o Windows 95/98, es indiferente. Si no se sabe el sistema que tiene la máquina, es mejor atenerse a la forma UNIX.

Dentro de los parámetros pasados a HREF, podrás ver que al principio pone **http://**, esto es, el tipo de servicio al que el visualizador va a llamar, y hay varios :

Servicio	Descripción	Ejemplo	Efecto
http://	Servicios WWW	<A HREF="http://www.uv.es/"> WWW</A>	WWW
ftp://	Servicios FTP	<A HREF="ftp://ftp.uv.es/">FTP</A>	FTP
news://	Servicios NEWS	<A HREF="news://news.uv.es/">NEWS</A>	NEWS
mailto://	Servicios E-mail	<A HREF="mailto:jac@uv.es/">E-mail</A>	E-mail
file:///C|	Fichero local	<A HREF="indice.htm/">Fichero</A>	Fichero

Además de HREF, el elemento <A> puede tener otro atributo: **TARGET**. Se utiliza para ordenar la apertura de otra ventana del visualizador con la página que se desee. Se escribirá:

```
<A HREF="indice.htm" TARGET="Ventana-2"> </A>
```

Esta instrucción mostrará la página **indice.htm** pero con otra ventana del visualizador, es decir tendremos lanzado el visualizador **2 veces**. Esto sólo funciona si el sistema operativo de tu máquina es multi tarea (p.ej.: Windows-95).

Del mismo modo, el atributo **TARGET** puede utilizarse para salir de una pantalla compuesta de frames. En efecto, si estamos en una pantalla con, por ejemplo, dos frames, cualquier link invocado desde cualquiera de ellos, hace que la página llamada aparezca en ese mismo frame. Para volver a una pantalla "normal" sin frames, se puede escribir un link con la sintaxis siguiente:

```
<A HREF="indice.htm" TARGET="_parent"> </A>
```

Como se acaba de ver, el atributo **HREF** sirve para enlazar con otro documento, que puede estar en un servidor o ser un fichero local, pero que siempre se nos presentará desde la primera línea del mismo.

El atributo **NAME** se utiliza para definir algo así como "un punto de aterrizaje" en cualquier línea del documento de destino, de forma que nos aparecerá en pantalla desde la línea deseada y no desde el principio. Esto es muy útil cuando se trata de documentos largos divididos en secciones.

Se escribirá así:

En el documento activo:

```
<A HREF="http://www.miservidor.es/mifichero.htm#punto1"> Ir al punto
1</A>
```

En el documento destino:

```
<A NAME="punto1"></A>
```

NAME también puede utilizarse para saltar de una línea a otra dentro de un mismo documento. Se escribirá así:

Dentro del documento activo **En la línea de partida:**

```
<A HREF="#punto1">Ir al punto 1</A>
```

En la línea de destino

```
<A NAME="punto1"></A>
```

Como habrás visto en el cuadro de arriba, es posible enviar un e-mail desde un link con la instrucción: <A HREF="mailto:jac@uv.es">Enviar e-mail</A>. Con esta sintaxis se invoca al cliente de correo predeterminado para que abra una ventana con la dirección deseada ya escrita. Si además quieres que dicha ventana se abra con el asunto (subject) y el texto (body) ya escritos se puede conseguir así:

```
<A HREF="mailto:carlos.vazquez@edu.xunta.es?&subject=Asunto de
prueba&body=Texto de prueba">Enviar e-mail</A>
```

Debes saber que...

- El texto que pongas después del símbolo # puede ser cualquiera, siempre que no tenga espacios en blanco, caracteres especiales ni caracteres codificados, y por supuesto, que no esté repetido en el mismo documento de destino.
- Si en el documento de destino no existe el punto definido en el documento de origen, el visualizador nos presentará en pantalla la última línea del mismo.
- Cuando hagas un link, es preferible utilizar direcciones relativas. Si utilizas direcciones absolutas y después tienes que mover los ficheros por cualquier razón, tendrás que modificar todas las direcciones.
- Si al hacer un link, después del nombre de la máquina no pones el nombre de un fichero, por defecto el servidor entiende que se le pide la "home page" (página inicial).
- Si un servidor no tiene definida página inicial, simplemente nos mostrará una lista de todos los ficheros y directorios que tenga en el directorio definido como raíz si el servidor tiene activada la opción de listar directorios, en caso contrario, dará un error.
- Si después de la dirección de la máquina a la que le haces el link en lugar de un nombre de fichero pones el de un directorio, generalmente el servidor nos mostrará un simple listado de los ficheros y directorios que éste contenga si el servidor tiene activada la opción de listar directorios, en caso contrario, dará un error.
- No sólo puedes montar el link sobre un texto, también puedes hacerlo sobre una imagen cualquiera. Es decir, después del link y antes de </A> puedes poner lo que quieras.
- El arte de escribir buen HTML reside en dar una buena estructura a la información. Siempre que puedas huye de crear páginas muy largas. Crea cuantas necesites, con buena estructura y enlázalas con cuantos links sean precisos.

- Recuerda: si tu trabajo HTML no va a residir en un servidor en red, y has utilizado direcciones absolutas, al hacer un link, no empieces con aquello de: "**http://....**". No funcionará.

- Si continuas deseando poner direcciones absolutas con ficheros locales, tendrá que ser así **file:///C|/MIDIRECTORIO/mifichero.htm**

- Después de **C** (que puede ser A, B, D o la unidad de disco que quieras) fíjate que va una barra vertical (pipe), y que las barras que separan los directorios y ficheros son barras **a la derecha**.

10.4.9. Insertar imágenes

Insertar imágenes en un documento permite crear páginas mucho más atractivas. Según el tipo de gráficos utilizado se pueden conseguir efectos realmente sorprendentes.

Para insertar una imagen en un documento HTML se utiliza el elemento **<IMG>**. Este elemento puede ir acompañado de los atributos **SRC ALT ISMAP ALIGN WIDTH HEIGTH BORDER VSPACE HSPACE**. Con las nuevas implementaciones que Netscape hace del elemento, se consiguen efectos de *imagen flotante* y por tanto se ha hecho necesario dotar al elemento <**BR**> del atributo **CLEAR**.

Las imágenes utilizadas pueden estar en formato GIF o JPG. Estos formatos son interpretados directamente por el visualizador. Se puede usar cualquier otro formato como el MPG (vídeo), pero entonces el visualizador llamará a un programa auxiliar, que previamente le habrás definido, para que sea éste el que visualice el fichero. La desventaja de las imágenes en formatos no tratados por el propio visualizador es que no verás el texto junto con la imagen, y por tanto no podrás imprimir la página compuesta.

Algunos visualizadores no son capaces de tratar imágenes, o aunque el visulizador sí pueda, tal vez tu máquina tenga muy poca memoria o una tarjeta de vídeo floja. Entonces podemos recurrir al atributo **ALT** (por alternativo) para definir un texto que aparecerá en lugar de la imagen. Esto es importante cuando una imagen, además de ser un elemento decorativo, soporta un link.

Veamos ejemplos del uso de <IMG>:

Imagen alineada a la izquierda (por defecto)

```
<IMG SRC="sugeren.gif">texto texto texto texto
texto texto texto texto
```

texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto

Imagen con un texto alternativo

Para verlo tienes de deshabilitar la opción de cargar imágenes de tu visualizador.

```
<IMG SRC="sugeren.gif" ALT="AQUI VA UNA IMAGEN">
```

Imagen alineada a la izquierda.Texto alineado arriba

```
<IMG SRC="sugeren.gif" ALIGN=TOP> texto texto texto texto texto
texto exto texto texto texto texto texto
```

Imagen alineada a la izquierda.Texto alineado abajo

```
<IMG SRC="sugeren.gif" ALIGN=BOTTOM> texto texto texto texto texto texto
exto texto texto texto texto texto
```

Imagen alineada a la izquierda.Texto alineado al centro

```
<IMG SRC="sugeren.gif" ALIGN=MIDDLE> texto texto texto texto texto texto
exto texto texto texto texto texto
```

Imagen alineada a la izquierda.Texto alineado a la derecha de la imagen

```
<IMG SRC="sugeren.gif" ALIGN=LEFT> texto texto texto texto texto
texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texzto texto texto
texto texto texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto
```

Imagen alineada a la izquierda.Texto envolviendo la imagen

```
<IMG SRC="sugeren.gif" ALIGN=LEFT> texto texto texto texto texto <BR
CLEAR>
texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texzto texto texto
texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto texto texto
```

Imagen alineada a la derecha.Texto alineado a la izquierda de la imagen

```
<IMG SRC="sugeren.gif" ALIGN=RIGHT> texto texto texto texto texto
texto texto texto texto texto texto texto
texto texto texto texto texto texto texto texto texto texzto texto texto
texto texto texto texto texto texto texto texto texto texto texto texto
texto texto texto texto texto
```

Si tu visualizador no soporta esto, hay un truco para conseguir algo parecido:

```
<TABLE> <TR><TD>texto texto texto texto texto texto texto texto texto
texto texto texto</TD><TD align=center><IMG SRC="sugeren.gif"></TD></TR>
</TABLE>
```

Imagen redimensionada a más.Texto alineado a la derecha de la imagen

```
<IMG SRC="sugeren.gif" ALIGN=LEFT WIDTH=272 HEIGTH=92 > texto texto texto
texto texto texto texto texto texto texto texto texto texto
```

Aquí la imagen es la misma de los ejemplos anteriores, pero ha sido redimensionada por el visualizador. Su tamaño original es de 136x46 puntos. Este es su aspecto al doble: 272x92.

Imagen redimensionada a menos.Texto alineado a la derecha de la imagen

```
<IMG SRC="sugeren.gif" ALIGN=LEFT WIDTH=75 HEIGTH=20 > texto texto texto
texto texto texto texto texto texto texto texto texto texto
```

Aquí la imagen es la misma de los ejemplos anteriores, pero ha sido redimensionada por el visualizador. Su tamaño original es de 136x46 puntos. Este es su aspecto a la mitad: 75x23

Imagen alineada a la izquierda con marco.Texto alineado a la derecha de la imagen

```
<IMG SRC="sugeren.gif" ALIGN=LEFT BORDER=5> texto texto texto texto texto
```

```
texto texto texto texto texto texto texto
```

Imagen alineada a la izquierda.Texto alineado a la derecha de la imagen. Espacios verticales y horizontales vacíos forzados.

```
texto texto texto texto texto texto texzto texto texto texto texto texto
<IMG SRC="sugeren.gif"> ALIGN=LEFT HSPACE=100 VSPACE=30> texto texto
texto
texto texto texto texto texto texto texto texto texto texto texto texto
texto texto texto
```

Usar una imagen como punto de montaje de un link

```
texto texto <A HREF="indice.htm"><IMG SRC="sugeren.gif" > </A>texto texto
```

La imagen aparece rodeada de un marco de color como es habitual en los puntos de link (en el texto es el subrayado). Si quieres eliminar el marco, escribe:

```
....  IMG SRC="sugeren.gif" BORDER=0 >
```

Ahora que ya sabes cómo manejar las imágenes, veamos algunos efectos especiales algo más complicados.

Mapeado de una imagen para usarla como direccionador

Netscape ha propuesto una solución mucho más simple: que sea el propio visualizador quien haga las veces de programa conversor. En efecto, un recurso tan bueno como los mapas, no debe depender de tener tu máquina en red y de que haya un servidor http que te atienda. Además, eso resta portabilidad a los documentos locales y a los trabajos personales en disquete. Para ello ha implementado el elemento **MAP** que acompañará a IMG.

Por ejemplo, en una imagen, la mitad izquierda nos envía a indice.htm y la mitad derecha a intro.htm. Al mover el puntero del ratón horizontalmente sobre la imagen y observar la ventanita que hay en la parte baja de la pantalla, se vera como cambia el nombre del link.

Se escribe así:

```
<MAP NAME="nombre1">
   <AREA SHAPE="rect" COORDS="1,1,75,46" HREF="indice.htm">
   <AREA SHAPE="rect" COORDS="76,1,136,46" HREF="intro.htm">
</MAP>
<IMG SRC="sugeren.gif" USEMAP="#nombre1">
```

Aunque se comprende a simple vista, analicémoslo:

En primer lugar tenemos el elemento **MAP**, que lo que está haciendo es definir un mapa de coordenadas. Va acompañado del atributo **NAME** que da nombre al mapa. Es necesario nombrarlo porque podría haber más de uno en la misma página, y evidentemente, sus nombres no deberán repetirse.

A continuación tenemos el atributo **AREA** que define las áreas de la imagen. El parámetro **SHAPE="rect"** indica la figura geométrica que estamos utilizando para ello. Al igual que con los otros mapas, puede ser **rect circle** y **poly**.

El parámetro **COORDS** fácilmente se adivina que indica las coordenadas del área, en este caso vértice superior izquierdo e inferior derecho, respectivamente.

HREF ya sabemos lo que hace: indica un link con una página, pero esta vez no va acompañando al elemento <A>, digamos que es un atributo prestado. Hay un área por cada link definido. Si un área no queremos que tenga link se definirá con **NOHREF**.

IMG SRC también son conocidos: hacen que se visualice la imagen.

Y por fin, **USEMAP** nos dice qué mapa de coordenadas hay que aplicar a la imagen. A una misma imagen se le pueden aplicar distintos mapas si se desea. Fíjate en que **nombre1**, (el nombre del mapa) va **precedido** del símbolo **#**. Esto se debe a que, en este caso, el mapa al que se hace referencia está en la misma página que la instrucción IMG. Podría estar en otra, y en ese caso el símbolo tendría que estar **después** del nombre de la página. Por ejemplo: **otrapagina.htm#nombre1**

Usar una imagen como elemento decorativo de una lista

Se pueden usar imágenes como elementos decorativos en una lista, como se ve en el ejemplo siguiente:

```
<DL>
    <DT><IMG SRC="bolaroja.gif">Linea de texto
    <DT><IMG SRC="bolaroja.gif">Linea de texto
    <DT><IMG SRC="bolaroja.gif">Linea de texto
</DL>
```

Linea de texto
Linea de texto
Linea de texto
Linea de texto

Usar una imagen como fondo de ventana

Se pueden usar imágenes como imagen de fondo en una página, como se ve en el ejemplo siguiente. El fondo (background) de este documento es una imagen en formato GIF. Es un atributo del elemento <BODY>

Se escribe así:

```
<BODY BACKGROUND="yellow_r.gif">
```

Y es compatible con todos los demás atributos de este elemento.

10.4.10. Introducción a los formularios

Esta es la parte más técnica y compleja del HTML. Los formularios o forms en inglés, son unas páginas especiales que se utilizan para realizar transacciones. En una página que contenga un formulario pueden además existir todos los elementos hasta ahora conocidos, incluso el propio formulario puede estar integrado dentro de otros elementos, como por ejemplo una tabla.

El formulario no es más que una página escrita en HTML como cualquier otra. Lo que la hace diferente es que permite capturar datos de varias maneras: directamente desde el teclado, escogiendo un valor de una lista desplegable o seleccionando una opción desde botones fijos o de todas ellas combinadas.

Una vez terminada la captura de datos, estos forman un registro que es enviado a un servidor http (web) que lo procesará y devolverá una respuesta. Pueden utilizarse también para enviar su contenido a una dirección de correo electrónico, o que simplemente abran otra página.

Si el formulario se diseña para abrir una página o para ser enviado por e-mail, todo se reduce a escribir HTML, pero la cosa se complica cuando se trata de transacciones de datos que hay que guardar en un servidor (comercio electrónico). En este caso, en el servidor tiene que haber un programa especial, creado específicamente para ese formulario, que reciba los datos y sepa qué hacer con ellos, procesarlos y confeccionar la respuesta que remitirá al navegador. A esa respuesta se le denomina *documento virtual* ya que esa página no está escrita en ninguna parte; es generada, enviada y destruida.

A los programas que hacen de puente entre el formulario y el servidor, se les llama programas **CGI** (**C**ommon **G**ateway **I**nterface) y no hay que confundirlos con las páginas dinámicas escritas en ASP o PHP, aunque se parecen mucho.

Los programas CGI pueden estar escritos en cualquier lenguaje de programación que sea soportado por el sistema operativo del servidor, y pueden estar diseñados para cualquier función: desde una simple captura de datos que serán guardados en un fichero, hasta la más sofisticada consulta a una base de datos.

Todo esto y algunas cosas más (que habrá que ver en la documentación de referencia) han de tenerse presentes a la hora de realizar programas CGI. Para terminarlo de complicar, no todos los visualizadores utilizan la misma forma de enviar los datos recibidos, por lo que se plantea a menudo la necesidad de decidir para qué visualizador queremos programar nuestro WEB.

10.4.11. ¿Cómo se escriben los formularios ?

La base del formulario es el elemento lleno **<FORM>**, es el que define una zona de la página como formulario. En una página puede haber varias zonas definidas como formulario. Dentro de este elemento a su vez se utilizan otros elementos, que son los que realmente dibujan en pantalla los componentes del formulario, son:

- **<INPUT>**
    - Campos de entrada por teclado.
    - Botones de selección.
    - Casillas de marca.
    - Botones de proceso.
    - Botones de inicialización (reset).

- Imágenes sensibles al ratón.

- **<SELECT>**
  - Listas desplegables de valores.
  - Listas fijas de valores.
- **<TEXTAREA>**
  - Ventanas de escritura libre.

Este sería el más elemental de los formularios, con sólo un campo y un botón de envío:

Analicemos cómo se escribe:

```
<FORM NAME="MI_FORMULARIO" METHOD="POST"
ACTION="http://servidor.com/pagina.php">
 <INPUT NAME="campo1">
 <INPUT TYPE="submit" VALUE="Procesar"></FORM>
```

En la primera línea vemos el elemento de definición de formulario: **<FORM>.** Va acompañado de tres atributos: **NAME, METHOD** y **ACTION**.

NAME, evidentemente, se refiere al nombre que se le asigne al formulario. No es obligatorio, pero si el formulario va a ser utilizado en páginas ASP, PHP, o simplemente en VBScript o JavaScript, se necesitará un nombre, ya que el formulario será considerado como un **objeto**.

METHOD se refiere al método que emplearemos para enviar los datos al servidor, y pueden ser dos: **POST** y **GET**. El optar por uno u otro obedece a complejas cuestiones de programación basadas en la máxima longitud de registro que puede enviarse; cada uno usa un canal de salida distinto. El que soporta más longitud es POST, y será el que utilicemos en casi todos los ejemplos.

ACTION se refiere a la acción que queremos que ejecute el formulario en un servidor http o en local. Con el valor del ejemplo **ACTION="http://servidor.com/pagina.php"** se le está indicando que ejecute una pagina llamada **pagina.php** que está en el servidor http **servidor.com** .

Además de enviar datos a un servidor, **ACTION** también puede realizar una acción en local, por ejemplo traer una página:

Se escribe:

```
<FORM METHOD="POST" ACTION="indice.htm">
  ......
</FORM>
```

En este caso el formulario funcionaría igual que un link, y no es necesario el atributo **NAME**.

Otra cosa que puede hacer **ACTION**, y que además tiene la ventaja de que no sería preciso diseñar un CGI, es enviar un e-mail (correo electrónico) a un usuario o a una lista de usuarios. Los datos enviados por el formulario, tendrán la forma que ya conocemos, pero una vez recibidos por esta vía, pueden ser tratados con programas de edición de texto, y convertidos al formato que se quiera. Se escribe:

```
<FORM METHOD="POST" ACTION="mailto:mi-usuario@miservidor-e-mail">
  ......
</FORM>
```

Si no quieres complicarte la vida con descodificaciones (depende de lo que se vaya a hacer con los datos recibidos), el navegador puede remitirlos con un formato más sencillo y ya descodificado. Para ello sólo hay que añadir el parámetro **ENCTYPE** con el valor **TEXT/PLAIN**. Se escribe así:

```
<FORM METHOD="POST" ACTION="mailto:mi-usuario@miservidor-e-mail"
 ENCTYPE="TEXT/PLAIN">
  ......
</FORM>
```

Resumiendo: sin poner ningun parámetro al elemento **ACTION** los datos te llegarán así:

```
CAMPO1=Esto+es+una&CAMPO2=demostracion+de+formularios
```

y poniendo **ENCTYPE="TEXT/PLAIN"** así:

```
CAMPO1=Esto es una
CAMPO2=demostracion de formularios
```

Al igual que con el elemento <A> es posible hacer que el asunto (subject) del e-mail se reciba con el texto que quieras, pero omitiendo la parte correspondiente al cuerpo (body) del mismo, ya que en este caso el contenido del e-mail son los campos del formulario:

```
<FORM METHOD="POST" ACTION=
"mailto:mi-usuario@miservidor-e-mail?&subject=asunto de prueba"
 ENCTYPE="TEXT/PLAIN">
  ......
</FORM>
```

El elemento INPUT

INPUT sin ningún atributo define por defecto una ventana de escritura de 20 caracteres de longitud por una línea de ancho:

Se escribe:

```
<FORM> <INPUT> </FORM>
```

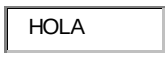
<INPUT> admite varios atributos:

- **SIZE** define la longitud de la ventana de texto.
- **MAXLENGTH** define la máxima longitud de la cadena que se puede escribir dentro de la ventana.
- **NAME** define el nombre de la ventana (que en realidad es un campo dentro del registro, que en conjunto, es el formulario). Si escribes:

```
<FORM> <INPUT SIZE=10 MAXLENGTH=10 NAME="campo1"></FORM>
```

Como puedes comprobar, sólo se pueden escribir 10 caracteres dentro de la ventana, que también es de longitud 10. No es obligatorio que concidan ambos valores, puedes definir la ventana al valor que quieras y la longitud de la cadena puede ser mayor o menor.

VALUE sirve para que la ventana aparezca con un valor predeterminado, y no vacía como hace por defecto:



Se escribe:

```
<FORM> <INPUT VALUE="HOLA"> </FORM>
```

TYPE=TEXT Es el valor por defecto de TYPE. Puede omitirse.

TYPE=PASSWORD para que el valor predeterminado de la ventana esté en formato oculto.Se escribe:

```
<FORM> <INPUT TYPE="PASSWORD" VALUE="HOLA"> </FORM>
```

TYPE=HIDDEN define que el valor predeterminado de la ventana y la ventana misma estén en formato oculto.Se escribe:

```
<FORM> <INPUT TYPE="HIDDEN" VALUE="HOLA"> </FORM>
```

Esta opción se utiliza cuando es necesario enviar un valor fijo al servidor, pero no se desea que el usuario intervenga o que lo vea. CUIDADO, esto no es un formato encriptado, viendo el documento fuente se puede saber el valor de la ventana.

TYPE=SUBMIT para generar un botón que al ser pulsado cierra la captura de datos del formulario y procede a ejecutar lo definido en el atributo ACTION que ya conocemos.

Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT NAME="campo1">
 <INPUT TYPE="SUBMIT" VALUE="Procesar"></FORM>
```

VALUE puede acompañar a SUBMIT , y en este caso sirve para definir el texto que queremos que tenga el botón en su interior. Si se omite, por defecto el visualizador le da el valor "Submit Query".

Existe otro tipo llamado **TYPE=BUTTON** que genera un botón igual al generado por el tipo submit, pero que no realiza ninguna acción; es simplemente un botón "muerto" en cuanto a acciones directas. Suele utilizarse para asociarle instrucciones en JavaScript mediante el evento **onClick**. Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT NAME="campo1">
 <INPUT TYPE="BUTTON" VALUE="No hace nada"></FORM>
```

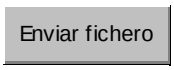
TYPE=RESET para generar un botón que al ser pulsado inicializa todos los componentes del formulario.

Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT NAME="campo1">
 <INPUT TYPE="RESET" VALUE="Inicializar"></FORM>
```

VALUE puede acompañar a RESET , y en este caso sirve para definir el texto que queremos que tenga el botón en su interior. Si se omite, por defecto el visualizador le da el valor "Reset".

TYPE=FILE Este es un nuevo tipo que sólo funciona con Netscape 3.0 o superior, y sirve para enviar un fichero a una máquina remota (hacer FTP), pero no funciona solo, necesita de ciertos acompañantes que hacen su uso un poco más complicado. Aquí hay que echar mano de un atributo parametrizado de <FORM>: **ENCTYPE="multipart/form-data"** para generar un botón que permitirá buscar un fichero en nuestra máquina, ponerlo en una ventanita de entrada como las ya conocidas (por tanto tanbién se puede escribir directamente el nombre del fichero en lugar de buscarlo) y junto con un botón tipo submit enviar el fichero.

Enviar el fichero: 

Se escribe:

```
<FORM enctype="multipart/form-data"
  ACTION="ftp://miservidor/" METHOD="POST">
Enviar el fichero:
<INPUT NAME="mifichero" TYPE="FILE">
<INPUT TYPE="SUBMIT" VALUE="Enviar fichero"></FORM>
```

el botón generado aparece con el texto "Browse.." y no sirve de nada añadirle el parámetro VALUE para darle otro nombre, como se puede hacer con el botón de tipo submit.

En este ejemplo se haría una conexión al servicio FTP general de la máquina remota, pero se puede hacer a un directorio en particular de un usuario concreto. Sólo habría que cambiar la línea ACTION="ftp://**miservidor**/" METHOD="POST"> y escribir:

```
    ACTION="ftp://miusuario@miservidor/" METHOD="POST">
```

Otra forma más directa de hacer un FTP en las dos direcciones, es desde la ventana "Location" del visualizador. La sintaxis de los parámetros de ACTION son válidos en esa ventana.

Por supuesto, todo esto si no tienes cuenta y permiso de escritura en la máquina remota, no sirve para nada...

Como ya se ha dicho antes, el atributo **ACTION** es capaz de enviar el contenido del formulario por correo electrónico, pero no sólo es capaz de enviar el formulario: tambien puede adjuntar al e-mail un fichero cualquiera. Lo Único que hay que hacer es añadirle un instruccion como la que acabamos de ver para el FTP. Así:

```
  <FORM enctype="multipart/form-data"
  ACTION="mailto:mi-usuario@miservidor-e-mail/" METHOD="POST">
Adjuntar el fichero:
<INPUT NAME="mifichero" TYPE="FILE">
<INPUT TYPE="SUBMIT" VALUE="Adjuntar fichero"></FORM>
```

TYPE=IMAGE hace que el visualizador presente una imagen que es sensible al ratón. Lo que el formulario envía al servidor es un registro cuyos campos son el nombre definido para la imagen seguido de los parámetros **.x=n .y=n** donde **n** son los números de las coordenas **x y** del punto en el que estaba el ratón en el momento del envío. Para presentar la imagen se utiliza el atributo **SRC** del elemento IMG. Este sería un ejemplo de respuesta:

Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
<INPUT TYPE=IMAGE NAME="imagen" SRC="sugeren.gif">
</FORM>
```

Los formularios que utilicen este tipo de recursos pueden prescindir del tipo SUBMIT, ya que el formulario se cierra con la imagen.

TYPE=RADIO genera botones circulares que permiten seleccionar un valor predeterminado o activar una opción

Procesar | Inicializar Clase A ○ Clase B ○

Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">
```

```
Clase A
 <INPUT TYPE="RADIO" NAME="clase" VALUE="A">
Clase B
 <INPUT TYPE="RADIO" NAME="clase" VALUE="B"></FORM>
```

En este ejemplo, los botones permiten seleccionar la clase A, la clase B o ninguna de las dos. Si se desea que por defecto un valor esté seleccionado, por ejemplo clase B, hay que añadir el parámetro **CHECKED** después del valor:

```
          ... VALUE="B" CHECKED></form>
```

TYPE=CHECKBOX genera casillas de marca que permiten seleccionar un valor predeterminado o activar una opción

Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">
Clase A
 <INPUT TYPE="CHECKBOX" NAME="clase" VALUE="A">
Clase B
 <INPUT TYPE="CHECKBOX" NAME="clase" VALUE="B"></FORM>
```

En este ejemplo, las casillas de marca permiten seleccionar la clase A, la clase B, ninguna, o las dos. Si se desea que por defecto un valor esté seleccionado, por ejemplo clase B, hay que añadir el parámetro **CHECKED** después del valor:

```
          ... VALUE="B" CHECKED></form>
```

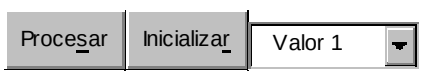
El elemento SELECT

SELECT sin ningún atributo define por defecto una lista desplegable de toda la pantalla de alto y una columna de ancho:

Se escribe:

```
<FORM> <SELECT> </SELECT>  </FORM>
```

Como evidentemente esto no es muy práctico, hay que parametrizar el elemento. En primer lugar se le da nombre, ya que éste es un campo más del formulario, la diferencia es que no hay que escribir en él sino escoger un valor de los que nos muestre al desplegarlo, se usará el atributo **NAME**, y para entrar valores en la lista se usa el atributo **OPTION**.



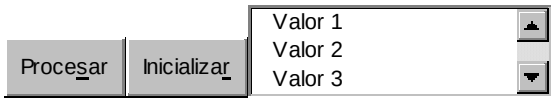
Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">

 <SELECT NAME="lista1">
  <OPTION>Valor 1
  <OPTION>Valor 2
  <OPTION>Valor 3
</SELECT>
</FORM>
```

Se deben tener en cuenta varias cosas respecto a SELECT:

- La longitud de la ventana de selección se autoajusta al valor más largo de la lista.
- La ventana de selección, por defecto, muestra una línea, si se quieren mostrar más se utilizará el atributo **SIZE**.
- Se debe procurar que los contenidos de los campos sean lo más cortos posibles; una excesiva longitud implica mayor tráfico por la red, pero se deben construir las listas de forma clara, y que resulten de fácil lectura al usuario, lo que puede aumentar la longitud de los valores.
- Para enviar un valor diferente del que aparece en la lista, se utiliza el parámetro **VALUE** asociado al atributo **OPTION**. Prueba con el valor 5 de la siguiente lista:



Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">

 <SELECT NAME="lista1" SIZE=3>
  <OPTION>Valor 1
  <OPTION>Valor 2
  <OPTION>Valor 3
  <OPTION>Valor 4
  <OPTION VALUE="Valor 5">Este es el Valor 5
  <OPTION>Valor 6
  <OPTION>Valor 7
</SELECT>
</FORM>
```

Esto siginifica que cuando selecciones "Este es el valor 5" en realidad lo que se envía al servidor es "Valor 5".

Con las listas desplegables es obligatorio seleccionar siempre un valor, y si no hay ninguno seleccionado, por defecto el visualizador enviará el primer valor de la lista.

Se puede hacer que haya un valor preseleccionado añadiendo el parámetro **SELECTED** al atributo OPTION, y si se quiere permitir más de una selección, se pondrá el atributo **MULTIPLE** al elemento SELECT. Para seleccionar y deseleccionar hay que atenerse a las normas de Windows.

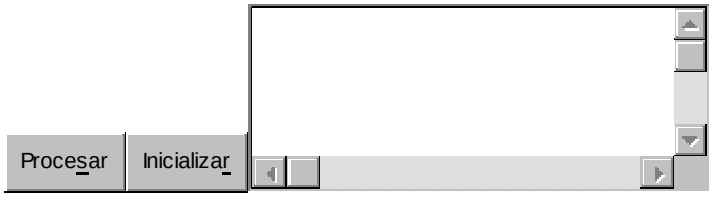
Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">

 <SELECT NAME="lista1" MULTIPLE SIZE=3>
  <OPTION>Valor 1
  <OPTION SELECTED>Valor 2
  <OPTION>Valor 3
  <OPTION>Valor 4
  <OPTION>Valor 5
  <OPTION>Valor 6
  <OPTION>Valor 7
</SELECT>
</FORM>
```

El elemento TEXTAREA

Este elemento, como su nombre indica, permite definir un área de texto en la pantalla en la que podemos escribir cualquier cosa. Se debe escribir ya parametrizada en cuanto a dimensiones, con los atributos **ROWS** (líneas) y **COLS** (COLUMNAS). No tiene otros atributos o posibles variantes. Por supuesto, para que sea operativo, deberá ir acompañado de los componentes necesarios para enviar e inicializar.



Se escribe:

```
<FORM METHOD="POST" ACTION="http://miserver.midominio.mipais/cgi-
bin/test2-cgi">
 <INPUT TYPE="SUBMIT" VALUE="Procesar">
 <INPUT TYPE="RESET" VALUE="Inicializar">
 <TEXTAREA NAME="texto1" ROWS=5 COLS=40 ></TEXTAREA>
</FORM>
```

Aunque no es habitual, dentro de un área de texto puede haber un contenido por defecto (que puede ser borrado o modificado por el usuario). Para ello simplemente se escribirá el texto entre **<TEXTAREA>** y **</TEXTAREA>**. Así:

```
<TEXTAREA NAME="texto1" ROWS=5 COLS=40>
  Contenido del area de texto
</TEXTAREA>
```

El tratamiento de los datos recibidos en el servidor desde los formularios requiere conocer la tabla de conversión hexadecimal que utilizan la mayoría de visualizadores. Recuerda que las palabras en destino aparecen separadas por el signo +, y las letras acentuadas y otros caracteres, por valores hexadecimales precedidos del símbolo %

10.4.12. Qué son los frames

Los frames (en inglés *frame* = cuadro, bastidor o marco) es un elemento implementado por Netscape, que permite dividir la pantalla en varias áreas independientes unas de otras, y por tanto con contenidos distintos, aunque puedan estar relacionados. No hay límites para el contenido de cada una de estas áreas: tienen las mismas propiedades que la pantalla completa normal, tal y como la conocemos. No hay que confundir los frames con las tablas. Su apariencia, a veces, puede ser similar, pero mientras el contenido de la celda de una tabla es fijo, en un área de pantalla creado por el elemento **FRAME** se dispone de todos los recursos del HTML. Es una zona viva.

Las páginas que contienen una definición de FRAME **no pueden** contener el elemento BODY ni ninguno de los elementos que habitualmente aparecen en el BODY **antes** del elemento **FRAMESET** que es el que define la creación del FRAME. Si esto no se cumple, el FRAME será ignorado.

Algunos visualizadores no soportan los frames. Para que nuestra página con frames no resulte opaca a ellos, se utilizará el elemento **NOFRAMES** que permite ofrecer un texto alternativo en entorno normal.

Los frames permiten una flexibilidad de presentación extraordinaria, y para contenidos muy complejos, de difícil estructura por los medios convencionales del HTML, son casi insustituibles. De nuevo se presenta la duda de para qué visualizador es preferible programar nuestro WWW.

Por supuesto, los frames son parametrizables en cuanto a tamaño y número de áreas, si éstas se pueden redimensionar por el usuario o son fijas, si tienen o no barras de scrolling, se pueden anidar, relacionar sus contenidos, etc. Veamos algunos ejemplos prácticos y su sintaxis:

En general, todas las páginas que contengan definiciones de frames, se comportan como si fuesen llamadores o "lanzaderas", y deberán ser más o menos así:

```
<HTML>
<HEAD><TITLE> Mi titulo ></TITLE></HEAD>
<FRAMESET>
     <NOFRAMES>
         <BODY>
             Su visualizador no soporta frames. Pulse
             <A HREF="indice.htm">aqui </A> para volver.
         </BODY>
    </NOFRAMES>
    <FRAME SRC="pagina1.htm" >
    <FRAME SRC="pagina2.htm" >
</FRAMESET>
</HTML>
```

Fíjate en que no aparece el elemento <BODY> en su posición habitual, sino dentro de un elemento especial que se activa sólo cuando el visualizador no soporta los frames: **NOFRAMES**. Si no declaras el área NOFRAMES y el visualizador no soporta este efecto, no se verá nada. Obviamente, los visualizadores que sí soportan frames ignorarán el contenido del área NOFRAMES.

Hasta aquí ya podemos hacernos una idea de cómo funcionan los frames: Lo primero es crear una minipágina con la definición del frame, viene a ser algo así como una "lanzadera" y sólo contiene la definición del frame. Y por último, crear las páginas

que constituirán el contenido de cada una de las áreas definidas en la "lanzadera". En el ejemplo anterior son **pagina1.htm** y **pagina2.htm**, y aquí es donde escribirás tus cosas; es decir que son páginas completamente normales, y que también pueden ser utilizadas de la forma habitual. A su vez, la "lanzadera" puede ser invocada con un link desde cualquier página normal. Como puedes ver, no es obligado crear todo tu documento con frames: puedes utilizarlos solamente allí donde sea necesario, si quieres.

Es posible definir cualquier combinación de áreas verticales y horizontales. La clave está en combinar adecuadamente los anidamientos del elemento **<FRAMESET>** con sus atributos **COLS** o **ROWS** según interese. Como puedes ver, el más importante es el primer <FRAMESET>, ya que es el que define cómo va a ser "troceada" la página inicialmente, si en porciones verticales u horizontales, y sobre esta base se deberán definir todos los demás anidamientos.

Un frame de 3 áreas verticales (COLS)

```
<FRAMESET COLS=30%,20%,50%>
   <FRAME SRC="a.htm">
   <FRAME SRC="b.htm">
   <FRAME SRC="c.htm">
</FRAMESET>
```

Un frame de 3 áreas horizontales (ROWS)

```
<FRAMESET ROWS=25%,25%,50%>
   <FRAME SRC="a.htm">
   <FRAME SRC="b.htm">
   <FRAME SRC="c.htm">
</FRAMESET>
```

Un frame combinado de un área vertical y dos horizontales

```
<FRAMESET COLS=20%,*>
   <FRAME SRC="a.htm">
        <FRAMESET ROWS=40%,*>
           <FRAME SRC="b.htm">
           <FRAME SRC="c.htm">
        </FRAMESET>
</FRAMESET>
```

Habrás podido comprobar que los parámetros que dimensionan los frames actuan al presentar la página inicialmente. Después, si el usuario lo desea, puede redimensionarlos como quiera. Esta es la opción por defecto. Si no se quiere permitir el redimensionado, se aplica el atributo **NORESIZE** al elemento FRAME. Así:

```
    <FRAME NORESIZE SRC.....>
```

Algo parecido ocurre con las barras de scrolling. Están regidas por el atributo **SCROLLING**, que puede valer **YES NO** o **AUTO**. Por defecto es **AUTO**. Con esta opción el visualizador decide, en función del contenido, si son necesarias las barras o no. Con **YES** las pondrá siempre, aunque no sean necesarias, y con **NO** no las pondrá nunca, aunque sean necesarias.

```
    <FRAME SCROLLING=YES ....>
```

Un frame con dos áreas verticales. Una normal, la otra con márgenes forzados para el texto (MARGINWIDTH MARGINHEIGHT)

```
<FRAMESET COLS=50%,50%>
   <FRAME SRC="a.htm">
   <FRAME SRC="a.htm"
       MARGINWIDTH=50
       MARGINHEIGHT=50>
</FRAMESET>
```

Un frame con referencias cruzadas (NAME TARGET)

```
<FRAMESET COLS=50%,50%>
   <FRAME SRC="aa.htm">
   <FRAME SRC="bb.htm" NAME="VENTANA">
</FRAMESET>
```

Para poner en un frame un enlace que se visualice en otro frame se ha de poner lo siguiente:

```
<A HREF="cc.htm" TARGET="VENTANA">
```

Hasta aqui se ha utilizado el nombre de "VENTANA" como destino del atributo TARGET. Este nombre, como ya sabes, es el que hemos definido en el código de este ejemplo. Si en lugar de poner el nombre de destino que se ha definido en el FRAMESET, pones otro cualquiera, el visualizador lo que hace es abrir otra ventana y colocar alli el contenido de la pagina. Evidentemente, cuantas mas ejecuciones del navegador haya en marcha, mas memoria del ordenador se necesita, y abrir una nueva ventana, implica ejecutar otra vez el navegador. Recuerda cuando hagas esto, que algunos no andan muy sobrados de memoria....

La versión 3.0 de Netscape, implementa dos nuevas posibilidades de los frames: la de darle color a la barra de separación de los distintos frames o la de que no se vea dicha barra. Para ello se utilizan los nuevos atributos de FRAMESET y FRAME:

FRAMEBORDER se utiliza como atributo de FRAMESET y establece si serán visibles los bordes del frame o no. Puede tener dos valores: **YES** (por defecto) y **NO**. Si escribes:

```
<FRAMESET FRAMEBORDER=NO COLS=30%,30%,30%>
   <FRAME SRC="a.htm">
   <FRAME SRC="b.htm">
   <FRAME SRC="c.htm">
</FRAMESET>
```

Se obtiene un frame de tres columnas sin barras de separación entre ellas.

BORDERCOLOR se utiliza como atributo de FRAME y establece el color de los bordes visibles. Evidentemente, para que funcione, se tendrá que haber establecido **FRAMEBORDER=yes**
Si escribes:

```
<FRAMESET  COLS=30%,30%,30%>
   <FRAME BORDERCOLOR="red" SRC="a.htm">
   <FRAME BORDERCOLOR="blue"SRC="b.htm">
   <FRAME SRC="c.htm">
```

```
</FRAMESET>
```

Se obtiene un frame de tres columnas con la primera barra de separación roja y la segunda azul.

Salir de una pantalla con frames

Para salir de una pantalla compuesta de frames es necesario definir algun link. En efecto, si estamos en una pantalla con, por ejemplo, dos frames, cualquier link invocado desde cualquiera de ellos, hace que la página llamada aparezca en ese mismo frame. Para volver a una pantalla "normal" sin frames, se puede escribir un link con la sintaxis siguiente:

```
<A HREF="indice.htm" TARGET="_parent">Ver índice sin marcos </A>
```

El destino **"_parent"** indica que presente la página llamada por el link en el navegador que se esta ejecutando, sin marcos y sin abrir nuevas ocurrencias del navegador.

10.5. ¿Por qué hay que usar códigos?

Existen diversos sistemas operativos dentro del mundo de los ordenadores. Estos sistemas no son otra cosa que unos programas especiales que se ejecutan inmediatamente después de encender el ordenador, y son los que se encargan de darle "vida" a nuestra máquina.

Junto con estos programas, los ordenadores cargan en memoria unas tablas especiales que coinciden con los caracteres que tiene tu teclado y algunos más que no están en el mismo, pero que se pueden escribir en pantalla por otros medios. Estas tablas, desafortunadamente, no son siempre las mismas, y varían de un sistema a otro, de una marca de ordenador a otra, y por supuesto, de un idioma a otro. Por ejemplo, no se ven igual las letras con acentos desde un PC que desde un MACintosh, o una terminal UNIX, si las escribimos directamente desde el teclado.

Dado que el HTML pretende ser un lenguaje universal, y que una página debe verse como su creador desea, sin importar si estamos ante un ordenador que "habla" en inglés o en español, o que tiene tal o cual sistema operativo, se ha creado una tabla de caracteres "conflictivos" (en realidad están todos, pero generalmente sólo se usa para caracteres especiales) que se escriben con un código en lugar de pulsar la tecla que lo contiene directamente.

Por ejemplo, nuestra denostada en medio mundo **"Ñ"** se escribirá:

`&Ntilde;` o bien `&#209;`

Esto habrá que hacerlo con todos los caracteres que no sean las letras del alfabeto (mayúsculas y minúsculas), los números y unos pocos signos, como el punto, la coma, el guión y algunos otros. A este conjunto de caracteres, común para todos los sistemas, se le denomina código **ASCII**, y desde luego hay que codificar todas las letras acentuadas, eñes, cedillas, etc., etc.

En el código ASCII tiene una columna con el carácter deseado seguida de su código numérico en base **decimal**, una descripción del carácter y después un nombre corto (una especie de alias) que para los más habituales se llegan a memorizar, y para cosas cortas, ayudan. No todos tienen ese nombre corto, y esos hay que escribirlos con el código numérico. En algunos casos especiales que se verán más adelante, habrá que escribir los códigos numéricos en base **hexadecimal** (la calculadora de Windows tiene un conversor).

Generalmente, se pueden escribir directamente desde teclado todos los que no tienen alias en la tabla, los que sí tienen normalmente darán problemas en sistemas diferentes al que se ha utilizado para escribir el documento. Si lo que escribes en HTML tienes la seguridad de que sólo va a ser utilizado como ficheros locales en máquinas similares a la tuya, no será necesario complicarse la vida; pero si tus páginas van a residir en un servidor WWW, cualquier tipo de máquina podrá acceder a ellas, y su aspecto no será el adecuado en algunas.