Capítulo 1: PHP y HTML

Páginas PHP

Las páginas PHP pueden ser páginas web *normales* a las que se cambia la extensión, poniendo **.php** en vez de **.htm** ó **.html**. En una página cuyo nombre tenga por extensión **.php** se pueden insertar instrucciones –escritas en lenguaje PHP– anteponiendo **<?** a la primera instrucción y escribiendo después de la última **?>**. A cada uno de estos *bloques de instrucciones* le llamaremos un **script**. No existe límite en cuanto al número de *scripts* distintos que pueden insertarse dentro de una página.

Un poco de *sintaxis*

La primera instrucción PHP que conoceremos será esta:

echo "*un texto..***";**

La instrucción **echo** *seguida de un texto entrecomillado* hará que el PHP escriba en la página web resultante lo contenido en esa cadena de texto.

Al final de cada instrucción debemos insertar siempre un punto y coma (;) El (**;**) indicará a PHP que lo que viene a continuación es una nueva instrucción. Para *facilitar* la depuración los *scripts no suelen escribirse* **dos instrucciones** en una misma **línea**.

print "*un texto..***";**

La instrucción **print** tiene una función similar –no es exactamente la misma– a la descrita para **echo**.

print ("*un texto..***");**

Esta es otra posibilidad –la más habitual– de utilizar **print**. Aquí *encerramos entre paréntesis* la cadena que pretendemos que aparezca impresa en la página web. El hecho de que utilicemos paréntesis no nos evita tener que encerrar la *cadena* (texto) a imprimir entre comillas.

Comillas *dentro de* comillas

Existen dos tipos de comillas: dobles « **"** » (SHIFT+2) y sencillas « **'** » (tecla **?** *en minúsculas*).

Cuando es preciso *anidar* comillas *deben utilizarse tipos distintos* para las exteriores y para las interiores. Para que una etiqueta **echo** interprete *unas comillas* como **texto** –y no como el final de la cadena– es necesario **anteponerles** un signo de *barra invertida*(\).

En ningún caso –ni con **echo** ni con **print**– está permitido sustituir las comillas exteriores (las que encierran la cadena) por \". Esta sintaxis solo es válida para indicar a PHP que debe interpretar las comillas como un carácter más.

La primera página en PHP

Observemos este código fuente. Como verás, se trata de una página web muy simple que **no contiene ningún script PHP**.

Hemos guardado esa página con el nombre **ejemplo4.html** y luego la hemos vuelto a guardar –sin modificar nada en sus contenidos– como **ejemplo4.php**.

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
Esta es una página supersimple
</body>
</html>
```

Si visualizamos ambos ejemplos veremos que los resultados son idénticos.

Los primeros *script* PHP

Editemos la página anterior (*ejemplo4.php*) y añadámosle nuestra primera etiqueta PHP guardándola como **ejemplo5.php**. Este sería el *código fuente*:

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
Esta es una página supersimple
<?
echo "¿Aparecerá esta linea?";
?>
</body>
</html>
```

Veamos ahora un ejemplo con las diferentes opciones de uso de las comillas

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
<?
/* Las instrucciones PHP son las que aparecen en rojo.
Las etiquetas en azul intenso son el código HTML.
Todo lo que aparece en este color son líneas de comentario
de las que hablaremos más adelante
Cuando rescribas estos primeros scripts
bastará que incluyas las instrucciones escritas en rojo */
/* ponemos <br> al final del texto para que cuando se
ejecute cada una de las instrucciones echo
se escriba -además del texto- un salto de linea HTML.
De este modo, el resultado de cada ECHO
aparecerá en una línea diferente */
# aquí utilizamos solo unas comillas
echo "Este texto solo lleva las comillas de la instrucción<br>";
# aquí anidaremos comillas de distinto tipo
echo "La palabra 'comillas' aparecerá entrecomillada<br>";
# esta es otra posibilidad invirtiendo el orden de las comillas
echo 'La palabra "comillas" aparecerá entrecomillada<br>';
# una tercera posibilidad en la que utilizamos un mismo
# tipo de comillas. Para diferenciar unas de otras anteponemos
# la barra invertida, pero esta opción no podríamos utilizarla
# al revés.
# No podríamos poner \" en las comillas exteriores.
echo "La palabra \"comillas\" usando la barra invertida<br>";
?>
</body>
</html>
```

¿Por qué usar líneas de comentario?

A primera vista pueden parecer inútiles. ¿Para qué *recargar* las páginas con contenidos que *no se van a ver ni ejecutar*? Las líneas de comentario sirven para poder *recordar* en un futuro *qué es lo que hemos hecho* al escribir un script y *por qué razón lo hemos hecho así*.

A medida que vayamos avanzando verás que en muchos casos tendremos que aplicar estrategias individuales para resolver cada problema concreto. Cuando necesites hacer una corrección o una modificación al cabo de un tiempo verás que *confiar en la memoria* no es una buena opción. Es mucho mejor utilizar una línea de comentario que confiar en la *memoria*. *¡Palabra!*

Comentarios

Para insertar comentarios en los *scripts* de PHP podemos optar entre varios métodos y varias posibilidades:

• Una sola linea

Basta colocar los símbolos **//** al comienzo de la línea o detrás del *punto y coma* que señala el final de una instrucción. También se puede usar el símbolo **#** en cualquiera de las dos posiciones.

• Varias líneas

Si un comentario va a ocupar más de una línea podremos escribir **/*** al comienzo de la primera de ellas y ***/** al final de la última. Las líneas intermedias no requieren de ningún tipo de marca. Los comentarios para los que usemos la forma */* ... **/* **no pueden anidarse**. Si, por error, lo hiciéramos, PHP nos dará un mensaje de error.

Ensayando líneas de comentario

En este ejemplo hemos incluido –marcados en rojo– algunos ejemplos de inserción de líneas de comentario.

```
<HTML>
<HEAD>
<TITLE>Ejemplo 7</TITLE></HEAD>
<BODY>
<?
// Este comentario no se verá en la página
echo "Esto se leerá <BR> "; // Esto no se leerá
/* Este es un comentario de
múltiples líneas y no se acabará
hasta que no cerremos así.... */
echo "Este es el segundo comentario que se leerá<BR>";
# Este es un comentario tipo shell que tampoco se leerá
# Este, tampoco
echo ("Aquí el tercer texto visible"); #comentario invisible
/* Cuidado con anidar
/* comentarios
multilinea con estos*/
al PHP no le gustan */
?>
</body>
</html>
```

Al ejecutarlo **nos dará un error**. Esto es una muestra, de la importancia que tiene el evitar **anidar** los comentarios

¿Qué es una constante?

Una **constante** es un valor –un número o una *cadena*– que **no va a ser modificado** a lo largo del proceso de ejecución de los *scripts* que contiene un documento. Para mayor comodidad, a cada uno de esos valores se le *asigna un nombre*, de modo que cuando vaya a ser *utilizado* **baste** con escribir su nombre.

Cuando *ponemos nombre* a una *constante* se dice que **definimos** esa constante.

¿Cómo definir constantes?

En **PHP** las constantes se definen mediante la siguiente instrucción:

define("*Nombre***","***Valor***")**

Los valores asignados a las constantes se mantienen en todo el documento, *incluso* cuando son invocadas desde *una función*. No es necesario escribir *entre comillas* los valores de las constantes cuando se trata de constantes **numéricas**.

Si se realizan operaciones aritméticas con constantes tipo **cadena**, y su valor **comienza por una letra**, PHP les asigna **valor cero**.

Si una *cadena* **empieza** por **uno o varios caracteres numéricos**, al tratar de operarla aritméticamente PHP considerará **únicamente** el valor de los **dígitos** *anteriores a la primera letra o carácter no numérico*.

El **punto** entre caracteres numéricos es considerado como **separador de parte decimal**. Tal como puedes ver en el *código fuente* del ejemplo, es posible **definir constantes** a las que se **asigne** como **valor** el *resultado de una operación aritmética*.

Ampliando *echo*

Mediante *una sola* instrucción **echo** se pueden *presentar* (en la ventana del navegador del cliente) *de forma simultánea* varias *cadenas de caracteres* y/o *constantes* y *variables*. Basta con **ponerlas** una a continuación de otra utilizando **una coma** como *separador* entre cada una de ellas.

La forma anterior no es la única –ni la más habitual– de enlazar elementos mediante la instrucción **echo**. Si en vez de utilizar la **coma** usáramos un **punto** (el *concatenador* de cadenas) conseguiríamos el mismo resultado. Cuando *enlacemos* elementos distintos –cadenas, constantes y/o números– hemos de tener muy en cuenta lo siguiente:

– **Cada una** de las sucesivas **cadenas** debe ir encerrada entre sus **propias** comillas.
– Los **nombres** de constantes **nunca** van entre comillas.

Ampliando *print*

Las instrucciones **print** también permiten concatenar cadenas en una misma instrucción. En este caso **solo es posible** usar el **punto** como elemento de unión. Si pusiéramos *comas* –como hacíamos con **echo**– PHP nos daría un **error**.

Un ejemplo con constantes

```
<HTML><HEAD><TITLE>Constantes</TITLE></HEAD>
<BODY>
<?
/* Definiremos la constante EurPta y le asignaremos el valor 166.386 */
define("EurPta",166.386);
/* Definiremos la constante PtaEur asignándole el valor 1/166.386
En este caso el valor de la constante es el resultado
de la operación aritmética dividir 1 entre 166.386*/
define("PtaEur",1/166.386);
/* Definimos la constante Cadenas y le asignamos el valor:
12Esta constante es una cadena*/
define("Cadena","12Esta constante es una cadena");
/* Definimos la constante Cadena2 y le asignamos el valor:
12.54Constante con punto decimal*/
define("Cadena2","12.54Constante con punto decimal");
/* Comprobemos los valores.
Observa la nueva forma en la que utilizamos echo
Lo hacemos enlazando varias cadenas separadas con
punto y/o coma, según se trate de echo o de print */
echo "Valor de la constante EurPta: ", EurPta, "<BR>";
echo "Valor de la constante PtaEur: ". PtaEur . "<BR>";
print "Valor de la constante Cadena: " . Cadena . "<BR>";
print "Valor de la constante Cadena x EurPta: " . Cadena*EurPta ."<br>";
print "Valor de la constante Cadena2 x EurPta: " . Cadena2*EurPta ."<br>";
echo "Con echo los números no necesitan ir entre comillas: " ,3,"<br>";
print "En el caso de print si son necesarias: " . "7" . "<br>";
print ("incluso entre paréntesis necesitan las comillas: "."45"."<br>");
```

```
print "Solo hay una excepción en el caso de print. ";
print "Si los números van en un print independiente no necesitan comillas ";
print 23;
# Pondremos la etiqueta de cierre del script y escribiremos
# una línea de código HTML
?>
<br>Ahora veremos los mismos resultados usando la function prueba<br><br>
<?
# Estamos dentro de un nuevo script abierto por el <? anterior
/* Aunque aún no la hemos estudiado, escribiremos una función
a la que (tenemos que ponerle siempre un nombre)
vamos a llamar prueba()
Lo señalado en rojo es la forma de indicar el comienzo
y el final de la funcion
Lo marcado en azul son las instrucciones
que deben ejecutarse cuando la función prueba()
sea invocada */
function prueba(){
echo "Valor de la constante EurPta: ". EurPta . "<BR>";
print "Valor de la constante PtaEur: ". PtaEur. "<BR>";
echo "Valor de la constante Cadena: ", Cadena , "<BR>";
print ("Valor de la constante Cadena x EurPta: " . Cadena*EurPta . "<br>");
print ("Valor de la constante Cadena2 x EurPta: " . Cadena2*EurPta . "<br>");
}
# Las funciones solo se ejecutan cuando son invocadas
/* La función anterior no se ejecutará hasta que escribamos
una línea -como esta de abajo- en la que ponemos
únicamente el nombre de la función: prueba()
*/
?>
<?
prueba();
?>
</body>
</HTML>
```

¿Qué es una variable?

Podríamos decir que es un espacio de la *memoria RAM* del ordenador que se *reserva* –a lo largo del tiempo de ejecución de un script– para almacenar un determinado tipo de datos cuyos valores son susceptibles de ser modificados por medio de las instrucciones contenidas en el propio programa.

Nombres de variables

En PHP todos los nombres de variable tienen que empezar por el símbolo **$**. Las nombres de las variables han de **llevar una letra** inmediatamente después del símbolo **$** –**$pepe1** es un nombre válido, pero **$1pepe** *no es un nombre válido*–. Para PHP las letras *mayúsculas* y las *minúsculas* son **distintas**. La variable **$pepe** es **distinta** de **$Pepe**.

Tipos de variables

En PHP no es necesario definir el **tipo de variable**, por lo tanto, **una misma variable** puede contener una *cadena de caracteres* en un momento del proceso y, posteriormente, un *valor numérico*, susceptible de ser operado matemáticamente.

Definición de variables

PHP no requiere una definición previa de las variables. Se definen en el momento en que son necesarias y para ello basta que se les asigne un valor.

La sintaxis es esta:

$variable=*valor***;**

El *valor* puede ser una *cadena* (texto o texto y números que no requieren ser operados matemáticamente) o sólo un *número*. En el primero de los casos habría que escribir el valor **entre comillas**.

Ámbito de las variables

Los valores de una variable definida en cualquier parte de un script –*siempre que no sea dentro de una función*– pueden ser utilizados desde cualquier otra parte de ese script, **excepto desde dentro de las funciones** que contuviera el propio script o desde las que pudieran estar contenidas en un *fichero externo*.

Si una variable es definida **dentro de una función** sólo podrá ser utilizada dentro esa función. Si en una *función* aludimos a una **variable externa** a ella PHP considerará esa llamada como si la variable tuviera valor **cero** (en caso de ser tratada como número) o una **cadena vacía** (**""** es una cadena vacía).

Igual ocurriría si **desde fuera** de una función hiciéramos alusión a una variable definida en ella. Si definimos dos variables con el mismo nombre, una dentro de una función y otra fuera, PHP las considerará distintas. La función utilizará –cuando sea *ejecutada*– sus propios valores sin que sus resultados modifiquen la variable *externa*.

Variables globales

Lo comentado anteriormente, admite algunas **excepciones**. Las funciones pueden utilizar valores de *variables externas a ellas* pero ello requiere **incluir** *dentro de la propia función* la siguiente instrucción:

global *nombre de la variable***;**

Por ejemplo: **global $a1;**

En una instrucción –**global**– pueden definirse como tales, de forma simultánea, varias variables. Basta con escribir los nombres de cada una de ellas separados por comas.

P. ej.: **global $a1, $a2, $a3;**

Variables superglobales

A partir de la versión **4.1.0** de **PHP** se ha creado un nuevo tipo de variables capaces de *comportarse como globales* sin necesidad de que se definan como tales.

Estas variables que *no pueden ser creadas por usuario*, recogen de forma automática *información muy específica* y tienen nombres preasignados que no pueden modificarse.

Las estudiaremos un poco más adelante. Por ahora, sólo citar los nombres de algunas de ellas:

$_SERVER, **$_POST**, **$_GET** o **$_ENV** son los de las más importantes.

Practicando con variables y sus *ámbitos*

Podemos comparar la **memoria** de un ordenador con el *salón* de un restaurante y la **ejecución de un programa** con los

servicios que van a darse en la *celebración del final de año*. La forma habitual de hacer una **reserva** de mesa –**espacio de memoria**– para ese evento sería facilitar un nombre –**nombre de la variable**– y especificar además cuantos comensales –**tipo de variable**– prevemos que van a asistir.

Cuando acudamos a la cena de *San Silvestre* podremos sentarnos en esa mesa un número determinado de comensales –**daremos un valor a la variable**– y a lo largo de ella podremos levantarnos o incorporar *un nuevo invitado* –**modificación del valor de la variable**– siempre que sea *alguien de nuestro* **ámbito** quien realice la invitación.

Probablemente no permitiríamos que el *cocinero* decidiera quien debe sentarse o levantarse, pero si lo permitiríamos a cualquiera de nuestros invitados. La diferencia estaría –**ámbito de la variable**– en que el *cocinero* no pertenece a nuestro ámbito mientras que los invitados a nuestra mesa sí.

Quizá si celebráramos el evento otro día cualquiera no necesitaríamos hacer *una reserva previa* y bastaría con acudir a la hora deseada y *hacer la reserva* justo en el momento de sentarse.

El *restaurante* de **PHP** no necesita que hagamos ninguna reserva previa. Otros muchos lenguajes de programación, por el contrario, si la necesitan.

Siguiendo con lo que nos ocupa, aquí tienes un ejemplo del uso de las variables y la forma de utilizarlas en los diferentes ámbitos.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<?
# Definimos la variable $pepe como vacía
$pepe="";
# Definimos las variables $Pepe y $Pepa (ojo con mayúsculas y minúsculas)
$Pepe="Me llamo Pepe y soy serio y formal";
$Pepa="Me llamo Pepa y también soy seria y formal";
?>
<!-- esto es HTML, hemos cerrado el script -->
<center><b>Vamos a ver el contenido de las variables</b></center>
<!-- un nuevo script PHP -->
<?
echo "<br> El valor de la variable pepe es: ",$pepe;
echo "<br> No ha puesto nada porque $pepe esta vacía";
echo "<br> El valor de la variable Pepe es: ",$Pepe;
?>
<center><b><br>Invocando la variable desde una función</b></center>
<?
/* Escribiremos una function llamada vervariable
Observa la sintaxis. La palabra function delante
y el () al final seguidos de la llave.
Hasta que no cerremos la llave todas las líneas
serán consideradas parte de la función */
function vervariable(){
echo "<br> Si invoco la variable Pepe desde una función";
echo "<br>me aparecerá en blanco";
echo "<br>El valor de la variable Pepe es: ",$Pepe;
}
/* esta llave de arriba señala el final de la función.
Los contenidos que hay en adelante ya no pertenecen a ella */
/* Haremos una llamada a la funcion vervariable.
Las funciones no se ejecutan hasta que no se les ordena
y se hace de esta forma que ves aquí debajo:
nombre de la funcion seguido de los famosos paréntesis */
vervariable();
?>
<!-- mas HTML puro -->
<center><b><br>Ver la variable desde la función
poniendo <i>global</i></b></center>
<?
# una nueva funcion
function ahorasi(){
# aqui definiremos a $Pepe como global
# la función leerá su valor externo
global $Pepe;
echo "<br><br> Hemos asignado ámbito global a la variable";
echo "<br>ahora Pepe aparecerá";
echo "<br>El valor de la variable Pepe es: ", $Pepe;
}
# hemos cerrado ya la funcion con la llave.
# Tendremos que invocarla para que se ejecute ahora
```

```
ahorasi();
?>
<center><b><br>Un solo nombre y dos <i>variables distintas</i></b><br>
Dentro de la función el valor de la variable es <br></center>
<?
function cambiaPepa(){
$Pepa="Ahora voy a llamarme Luisa por un ratito";
echo "<br>",$Pepa;
}
cambiaPepa();
?>
<center>... pero después de salir de la función
vuelvo al valor original...</center>
<?
echo "<br>",$Pepa;
?>
</BODY>
</HTML>
```

Constantes predefinidas

PHP dispone de algunas *constantes predefinidas* que no requieren la instrucción:

define("Nombre","Valor")

Algunas de ellas son estas:

__FILE__

Recoge el nombre del fichero que se está ejecutando y la ruta completa de su ubicación en el servidor.

__LINE__

Recoge el número de línea (incluidas líneas en blanco) del fichero **PHP** cuyos scripts está interpretando. Puede resultar muy útil para *depurar* programas escritos en PHP.

PHP_OS

Recoge información sobre el Sistema Operativo que utiliza el servidor en el que se está interpretando el fichero.

PHP_VERSION

Recoge la versión de **PHP** que está siendo utilizada por el servidor.

¡Cuidado!

Por si existieran dudas –por problemas de visualización– tanto **FILE** como **LINE** tienen que llevar **dos guiones *bajos*** delante y otras dos detrás.

Un ejemplo con constantes predefinidas

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<?
# La constante del sistema __FILE__ nos devolverá
echo "La ruta completa de este fichero es: ";
echo __FILE__;
# La constante del sistema __LINE__ nos devolverá
# el número de línea que se está interpretando
# también cuenta las líneas en blanco
# cuenta las líneas y verás que devuelve ... 16
echo "<br>Esta es la línea: ",__LINE__, "del fichero";
echo "<br>Estamos utilizando la versión: ",PHP_VERSION, " de PHP";
echo "<br>El PHP se está ejecutando desde el sistema operativo: ",PHP_OS;
?>
</BODY>
</HTML>
```

Capítulo 6: Otras variables

Valores de las variables

Cuando hablábamos de las variables, nos referíamos a su **ámbito** y comentábamos que las variables definidas dentro de una función pierden sus valores en el momento en el que abandonemos el ámbito de esa función, es decir, cuando finaliza su ejecución.

Decíamos también que si el **ámbito** en el que hubiera sido definida fuera *externo a una función* los valores sólo *se perderían* –**temporalmente**– mientras durara la eventual *ejecución* de las instrucciones de aquella y que, una vez acabado ese proceso, volvían a recuperar sus valores.

Bajo estas condiciones, si invocáramos repetidamente la misma función obtendríamos cada vez el mismo resultado. Las posibles modificaciones que pudieran haberse efectuado (a través de las instrucciones contenidas en la función) en el valor inicial de las variables, se perderían cada vez que abandonáramos la función con lo cual, si hiciéramos *llamadas* sucesivas, se repetirían tanto el valor inicial como el resultado.

Variables estáticas

Para poder conservar el último valor de una variable definida *dentro de una función* basta con definirla como **estática**.

La instrucción que permite establecer una variable como **estática** es la siguiente:

static *nombre = valor;*

P. ej: si la variable fuera **$a** y el **valor inicial** asignado fuera **3** escribiríamos:

static $a=3;

La variable conservará el último de los valores que pudo habérsele asignado durante la ejecución de la **función** que la contiene. No retomará el **valor inicial** hasta que **se actualice** la página.

Variables *de variables*

Además del método habitual de asignación de nombres a las variables -poner el signo **$** delante de una palabra-, existe la posibilidad de **que *tomen como nombre* el *valor* de** *otra variable* previamente definida.

La forma de hacerlo sería esta:

$$*nombre_variable_previa;*

Veamos un ejemplo. Supongamos que tenemos una variable como esta:

$color=*"verde";*

Si ahora queremos definir una nueva variable que utilice como nombre el valor (*verde*) que está contenido en la variable previa (*$color*), habríamos de poner algo como esto:

$$color=*"es horrible";*

¿Cómo podríamos *visualizar* el valor de esta *nueva variable*?

Habría tres formas de escribir la instrucción:

print $$color;

o

print ${$color};

o también

print $verde;

Cualquiera de las instrucciones anteriores nos produciría la misma salida: *es horrible*.

Podemos preguntarnos ¿cómo se justifica que existan dos sintaxis tan similares como *$$color* y *${$color}*? ¿Qué *pintan* las llaves?.

La utilización de las llaves es una forma de evitar situaciones de interpretación confusa. Supongamos que las variables tienen un nombre un poco *más raro*. Por ejemplo que *$color* no se llama así sino *$color[3]* (podría ser que $color fuera un *array* –una lista de colores– y que esta variable contuviera el tercero de ellos).

En este supuesto, al escribir: *print $$color[3]* cabría la duda de si el número 3 pertenece (es un índice) a la variable *$color* o si ese número corresponde a *$$color*.

Con *print ${$color[3]}* no habría lugar para esas dudas. Estaríamos aludiendo de forma inequívoca a **3** como índice de la variable *$color*.

¿Qué ocurre cuando la variable previa cambia de valor?

Cuando la **variable** utilizada para definir una *variable de variable cambia de valor* **no se modifica** *ni el nombre de esta última ni tampoco su valor*.

Puedes ver este concepto, con un **poco más de detalle, en el *código* *fuente*** del ejemplo.

Variables estáticas

```
<?
# Observa que hemos prescindido de los encabezados HTML.
# No son imprescindibles para la ejecución de los scripts
/* Escribamos una función y llamémosla sinEstaticas
Definamos en ella dos variables sin ninguna otra especificación
e insertemos las instrucciones para que al ejecutarse
se escriban los valores de esas variables */
function sinEstaticas(){
# Pongamos aquí sus valores iniciales
$a=0;
$b=0;
# Imprimamos estos valores iniciales
echo "Valor inicial de $a: ",$a,"<br>";
echo "Valor inicial de $b: ",$b,"<br>";
/* Modifiquemos esos valores sumando 5 al valor de $a
y restando 7 al valor de $b.
$a +=5 y $b -=7 serán quienes haga esas
nuevas asignaciones de valor
ya lo iremos viendo, no te preocupes */
$a +=5;
$b -=7;
# Visualicemos los nuevos valores de las variables
echo "Nuevo valor de $a: ",$a,"<br>";
echo "Nuevo valor de $b: ",$b,"<br>";
}
# Escribamos ahora la misma función con una modificación que será
# asignar la condición de estática a la variable $b
# Llamemos a esa función: conEstaticas
function conEstaticas(){
# Definimos $b como estática
$a=0;
static $b=0;
echo "Valor inicial de $a: ",$a,"<br>";
echo "Valor inicial de $b: ",$b,"<br>";
$a +=5;
$b -=7;
echo "Nuevo valor de $a: ",$a,"<br>";
echo "Nuevo valor de $b: ",$b,"<br>";
}
# Insertemos un texto que nos ayude en el momento de la ejecución
print ("Esta es la primera llamada a sinEstaticas()<br>");
# Invoquemos la función sinEstaticas;
sinEstaticas();
# Añadamos un nuevo comentario a la salida
print ("Esta es la segunda llamada sinEstaticas()<br>");
print ("Debe dar el mismo resultado que la llamada anterior<br>");
# Invoquemos por segunda vez sinEstaticas;
sinEstaticas();
# Hagamos ahora lo mismo con la función conEstaticas
print ("Esta es la primera llamada a conEstaticas()<br>");
conEstaticas();
print ("Esta es la segunda llamada a conEstaticas()<br>");
print ("El resultado es distinto a la llamada anterior<br>");
conEstaticas();
?>
```

Variables de *variables*

```
<?
# Definamos una variable y asignémosle un valor
$color="rojo";
# Definamos ahora una nueva variable de nombre variable
# usando para ello la variable anterior
$$color=" es mi color preferido";
# Veamos impresos los contenidos de esas variables
print ( "El color ".$color. $$color ."<br>");
```

```
#o también
print ( "El color ".$color. ${$color}."<br>");
# o también
print ( "El color ".$color. $rojo."<br>");
# advirtamos lo que va a ocurrir al visualizar la página
print ("Las tres líneas anteriores deben decir lo mismo<br>");
print ("Hemos invocado la misma variable de tres formas diferentes<BR>");
# cambiemos ahora el nombre del color
$color="magenta";
/* La variable $rojo seguirá existiendo.
El hecho de cambiar el valor a $color
no significa que vayan a modificarse
las variables creadas con su color anterior
ni que se creen automáticamente variables
que tengan por nombre el nuevo valor de $color */
# Pongamos un mensaje de advertencia para que sea visualizado en la salida
print ("Ahora la variable $color ha cambiado a magenta<br>");
print ("pero como no hemos creado ninguna variable con ese color<br>");
print ("en las lineas siguientes no aparecerá nada <br>");
print ("detrás de la palabra magenta <br>");
# Escribimos los print advertidos
print (" El color ".$color.$$color."<br>");
print (" El color ".$color.${$color}."<br>");
# Comprobemos que la variable $rojo creada como variable de variable
# cuando $color="rojo" aún existe y mantiene aquel valor
print ("Pese a que $color vale ahora ".$color."<br>");
print ("la vieja variable $rojo sigue existiendo <br>");
print ("y conserva su valor. Es este: ".$rojo);
?>
```

Capítulo 7: Tipos de variables

Tipos de variables

En PHP **no es necesaria** una definición previa del tipo de variables. Según los valores que se les vayan asignando, las variables podrán cambiar de tipo –de modo automático– y se irán adaptando a los valores que contengan en cada momento.

Las variables en PHP pueden ser de tres tipos:

– **Enteras** (tipo ***Integer***)
– **De coma flotante** (tipo ***Double***)
– **Cadenas** (tipo ***String***)

Cualquier número entero cuyo valor esté comprendido entre ± 2^{31} será interpretado por PHP como de tipo **Integer**.

Si el valor de una variable es un *número decimal* o –siendo entero– *desborda* el intervalo anterior, bien por asignación directa o como resultado de una operación aritmética, PHP la convierte a tipo **Double**.

Cualquier variable a la que se le asigne como valor el contenido de una cadena de caracteres (letras y/o números delimitados por comillas) es interpretada por PHP como tipo **String**.

Determinación de tipos de variables

PHP dispone de la función:

gettype(*variable***)**

que *devuelve una cadena de caracteres* indicando el tipo de la variable que contiene. La cadena devuelta por esta función puede ser: **Integer**, **double** o **string**.

Forzado de tipos

PHP permite *forzar* los tipos de las variables. Eso quiere decir que se puede **obligar** a PHP a asignar *un tipo determinado* a *una variable determinada*, siempre que los valores que contenga **estén dentro del rango** del nuevo tipo de variable. ***Los tipos se pueden forzar tanto en*** el momento de definir la variable como después de haber sido definida.

Forzado y asignación simultánea de valores

Al asignar un valor a una variable, se puede *forzar* su tipo de la siguiente forma. Si deseamos que la variable pase a ser tipo de **double** basta con **anteponer** a su valor –entre paréntesis– tal como se indica una de las expresiones:

(double), **(real)** o **(float)**.

Por ejemplo:

$a=((double)45); o
$a=((float)45); o
$a=((real)45);

cualquiera de ellas produciría el mismo efecto: convertir la variable **$a** a tipo **Double**.

Para *forzar* una variable a tipo **Integer** podemos **anteponer** a su valor una de estas expresiones:

(integer), o **(int)**.

Por ejemplo:

$b=((integer)4.5); o
$b=((int)45);

producirían el mismo efecto: convertir la variable **$b** a tipo **Integer**.

Para *forzar* una variable a tipo **String** basta con **anteponer** a su valor (entre paréntesis): **(string)**.

Por ejemplo:

$c=((string)4.5);

convertiría la variable **$c** a tipo **String**.

Forzado de tipos en variables ya definidas

La forma más aconsejable de *forzado* de tipos en variables que ya estuvieran definidas previamente, es el uso de la siguiente instrucción:

settype(*var,tipo***)**

donde *var* es el nombre de la variable cuyo tipo pretendemos modificar y *tipo* una expresión que puede contener (**entre comillas**) uno de estos valores: *'double'*, *'integer'*, o *'string'* según se trate de forzar a: *coma flotante*, *entero*, o *cadena*.

Un ejemplo podría ser este:

settype($a,'integer')

que convertiría a *tipo entero* la variable *$a*. ***La ejecución de la instrucción* settype** *devuelve* (da como resultado) un valor que puede ser: **true** o **false** (**1** ó **0**) según la *conversión* se haya realizado con *éxito* o no haya podido realizarse.

Operaciones con distintos tipos de variables

PHP permite la realización de operaciones aritméticas con cualquiera de los tres tipos de variables y adecúa el resultado al *tipo* más apropiado.

En la última tabla puedes ver algunos ejemplos, pero, en resumen, ocurre lo siguiente:

– Al operar con dos enteros, si el resultado está dentro del rango de los enteros, devuelve un entero.
– Si al operar con dos enteros el resultado desborda el rango *entero*, convierte su valor, de forma automática, al tipo *coma flotante*
– Al operar un *entero* con una variable tipo *coma flotante* el resultado es de *coma flotante*.
– Al operar con una *cadena* lo hace como si se tratara de un *entero*. Si hay caracteres numéricos al comienzo, los extrae (hasta que aparezca un **punto** o **un carácter no numérico**) y los opera como un número **entero**.
– Si una cadena *no comienza* por un carácter numérico PHP la operará tomando su valor numérico como CERO.

Tipos de variables

En el cuadro siguiente podemos ver los tres tipos de variables que utiliza PHP.

Las variables en PHP				
Tipo	Ejemplo	Valor máximo	Valor mínimo	Observaciones
Integer	$a=1234	2147483647	-2147483647	Cualquier valor numérico entero (dentro de este intervalo) que se asigne a una variable será convertido a este tipo
Double	$a=1.23	Cualquier valor numérico decimal, o entero fuera del intervalo anterior, que se asigne a una variable la convertirá a este tipo		
String	$a="123"	Cualquier valor *entrecomillado* (sean números o letras) que se asigne a una variable la convertirá a este tipo		

Determinación del tipo de variable utilizada

Dado que PHP gestiona las variables de forma automática y modifica los tipos de acuerdo con los valores que va tomando durante la ejecución del *script*, se pude recurrir a la función *gettype(nombre de la variable)* para determinar el **tipo de la variable actual**.

En la tabla siguiente tienes algunos ejemplos de aplicación de esa función. Podemos observar –en la columna *Sintaxis*– que para *visualizar* el resultado anteponemos *echo* a *gettype*. Es decir, le indicamos a PHP que *muestre* el resultado obtenido al determinar el tipo de variable.

Ejemplos de determinación del tipo de una variable		
Variable	Sintaxis	Devuelve
$a1=347	**echo gettype($a1)**	**Integer**
$a2=2147483647	**echo gettype($a2)**	**Integer**
$a3=-2147483647	**echo gettype($a3)**	**integer**
$a4=23.7678	**echo gettype($a4)**	**double**
$a5=3.1416	**echo gettype($a5)**	**double**
$a6="347"	**echo gettype($a6)**	**string**
$a7="3.1416"	**echo gettype($a7)**	**string**
$a8="Solo literal"	**echo gettype($a8)**	**string**
$a9="12.3 Literal con número"	**echo gettype($a9)**	**string**
$a10=""	**echo gettype($a10)**	**string**

Forzado de tipos

Aquí tienes algunos ejemplos de *forzado de tipos*. Te sugerimos que *eches un vistazo* a las advertencias que hemos puesto después de esta tabla.

Forzado de tipos		
Variable	Sintaxis	Devuelve
$a1=347	**echo gettype((real)$a1)**	**Double**
$a2=2147483647	**echo gettype((double)$a2)**	**double**
$a3=-2147483647	**echo gettype((float)$a3)**	**double**
$a4=23.7678	**echo gettype((int)$a4)**	**integer**
$a5=3.1416	**echo gettype((integer)$a5)**	**integer**
$a6="347"	**echo gettype((double)$a6)**	**double**
$a7="3.1416"	**echo gettype((int)$a7)**	**integer**
$a7="3.1416"	**echo gettype((string)$a7)**	**string**
$a8="Solo literal"	**echo gettype((double)$a8)**	**double**
$a9="12.3 Literal con número"	**echo gettype((int)$a9)**	**integer**

¡Cuidado!

Al modificar los tipos de variables pueden modificarse sus valores. Si forzamos a entera una variable que contenga un número decimal se perdería la parte decimal y la variable modificada solo contendría el valor de la parte entera.

Si tratamos de convertir a numérica una variable alfanumérica el nuevo valor sería cero. Aquí tienes algunos ejemplos relacionados con la advertencia anterior

Nuevos valores de la variable		
Valor inicial	Sintaxis	Nuevo valor
$a1=347	**echo ((real)$a1)**	**347**
$a2=2147483647	**echo ((double)$a2)**	**2147483647**
$a3=-2147483647	**echo ((float)$a3)**	**-2147483647**
$a4=23.7678	**echo ((integer)$a5)**	**23**
$a5="3.1416"	**echo ((double)$a6)**	**3.1416**
$a6="347"	**echo ((int)$a7)**	**347**
$a7="3.1416"	**echo ((string)$a7)**	**3.1416**
$a8="Solo literal"	**echo ((int)$a8)**	**0**
$a9="12.3 Literal con número"	**echo ((double)$a9)**	**12.3**
$a10=""	**echo ((int)$a9)**	**0**

Forzado de tipos usando *settype()*

Aquí tienes algunos ejemplos del uso de esa función. La tabla está organizada en bloques de *tres filas* que corresponden a la ejecución de tres instrucciones y a la visualización del resultado de cada una de ellas.

El resultado de **settype** –primera fila– solo podrá ser **1** ó **0** según la instrucción se haya ejecutado con éxito o no haya podido realizarse.

En la *segunda fila* comprobamos el nuevo tipo de variable obtenida mediante la ejecución de la instrucción anterior y en la *tercera* visualizamos los nuevos valores de la variable, que pueden haber cambiado como consecuencia del cambio de tipo.

Forzado de tipos con settype()		
Variable	Sintaxis	Devuelve
$a1=347	**echo (settype($a1,'double')**	**1**

	echo gettype($a1)	**Double**
	echo $a1	**347**
$a2=2147483647	**echo (settype($a2,'double')**	**1**
	echo gettype($a2)	**double**
	echo $a2	**2147483647**
$a3=-2147483647	**echo settype($a3,'double')**	**1**
	echo gettype($a3)	**double**
	echo $a3	**-2147483647**
$a4=23.7678	**echo settype($a4,'integer')**	**1**
	echo gettype($a4)	**integer**
	echo $a4	**23**
$a5=3.1416	**echo settype($a5,'integer')**	**1**
	echo gettype($a5)	**integer**
	echo $a5	**3**
$a6="347"	**echo settype($a6,'double')**	**1**
	echo gettype($a6)	**double**
a7="3.1416"	**echo settype($a7,'integer')**	**1**
	echo gettype($a7)	**integer**
	echo $a1	**3**
$a8="Solo literal"	**echo settype($a8,'double')**	**1**
	echo gettype($a8)	**double**
	echo $a8	**0**
$a9="12.3 Literal con número"	**echo settype($a9,'integer')**	**1**
	echo gettype($a9)	**integer**
	echo $a9	**12**

Tipos de variable de los operadores y de los resultados

La tabla siguiente contiene –en cada fila– los valores asignados a dos variables (A y B) y el resultado de la suma de ambas. A continuación se recogen los tipos de variable de cada una de ellas y el del resultado. El tipo de este último –generado por PHP– estará condicionado por el valor del resultado de cada una de las operaciones.

Resultados de operaciones y tipos de variables resultantes					
Valores			Tipos de variables		
A	B	A+B	A	B	A+B
12	16	28	integer	integer	integer
12	2147483647	2147483659	integer	integer	Double
-12	-2147483640	-2147483652	integer	integer	Double
12	1.2456	13.2456	integer	double	Double
1.2456	12	13.2456	double	integer	Double
1.2456	123.4567	124.7023	double	double	Double
12	abc	12	integer	string	Integer
1.2456	abc	1.2456	double	string	Double
12	12abc	24	integer	string	Integer
12	12.34567abc	24.34567	integer	string	Double
1.2456	12.34567abc	13.59127	double	string	Double

1.2456	12.3e2abc	1231.2456	double	string	Double
abc	12abc	12	string	string	Integer
abc	12.34567abc	12.34567	string	string	Double
12abc	12.34567abc	24.34567	string	string	double

Capítulo 8: Utilizando formularios

PHP dinámico

Lo que hemos visto hasta el momento, solo nos ha servido para escribir una serie de scripts PHP y ver los resultados de su ejecución, pero aún no hemos visto de qué forma se puede lograr que **interactúen** el *cliente* y el *servidor*. Veamos cómo hacerlo.

Envío a través del navegador

La forma más simple de que un **cliente** pueda *enviar* valores a un servidor es incluir esos valores en la propia petición, insertándolos directamente en la barra de direcciones del navegador.

Forma de envío

Deberá insertarse –en la barra de direcciones del navegador– lo siguiente:

*pagina.php***?***n1***=***v1***&***n2***=***v2*

donde

pagina.php será la dirección de la página que contiene el script que ha de procesar los valores transferidos. **?** es un carácter obligatorio que indica que detrás de él van a ser insertados nombres de variables y sus valores. *n1, n2, etcétera* representan los *nombres* de las variables. **=** es el separador de los nombres de las variables y sus valores respectivos. *v1, v2,...* simbolizan el valor asignado a cada una de las variables. **&** es el símbolo que separa los distintos bloques *variable* = *valor*.

Los **nombres** de las variables **nunca llevan** el signo **$**. Los **valores** de las variables –sean números o cadenas– **nunca se escriben entre comillas**. Algunos caracteres especiales (& por ejemplo) no pueden escribirse directamente dado que se prestan a confusión (no sabría si es un valor de una variable o si se trata de el símbolo de unión.

En esos casos es necesario sustituir en carácter por su codificación URL que representa cada carácter anteponiendo el signo % al valor de su código ASCII expresando en formato hexadecimal.

Se pueden incluir tantos *nombre* = *valor* como se desee. La única restricción es la longitud máxima permitida por el método **GET** (el utilizado en este caso) que se sitúa en torno a los 2.000 caracteres.

Recepción de datos

Cuando es recibida por el *servidor* la **petición** de un documento con extensión **.php** en la que tras el signo **?** se incluyen una o varias parejas *nombre* = *valor*, los nombres de las variables y sus valores respectivos se incluyen, de forma automática, en variables predefinidas del tipo:

$HTTP_GET_VARS['*n1***']**
$HTTP_GET_VARS['*n2***']**

en las que *n1*, *n2*, ... coinciden **exactamente** con nombres asignados a cada una de las variables en esa transferencia. Cada una de esas variables contendrá como valor aquel que hubiera recibido a través de la petición.

Si la *versión* de **PHP** es *superior* a la **4.1.0**, esos mismos valores se incluirán en variables superglobales del tipo **$_GET** de modo que –en el supuesto de que la versión lo soporte– los valores de la petición **también** (esta opción no excluye la anterior) estarían disponibles en:

$_GET['*n1***']**
$_GET['*n2***']**

Según el modo en que esté configurado el **php.ini** podría caber una *tercera* posibilidad de registro de esos valores. Si la directiva *register_globals* –en el fichero **php.ini**– está configurada con la opción **ON**, los valores transferidos desde el navegador –además de ser recogidos en las variables anteriores– son asignados a otras variables del tipo: **$n1**, **$n2**, ... cuyos nombres son el resultado de anteponer el símbolo **$** a los nombres de las variables contenidas en la petición.

La elección a la hora de escribir los scripts de uno u otro tipo de variable debe hacerse teniendo en cuenta que:

– Esta última –sin duda la más ***cómoda– tiene el problema de que*** *sólo es válida* cuando *register_globals=on* y, además, es la más *insegura* de todas.
– La superglobal **$_GET** tiene una sintaxis más corta que su alternativa y, además, añade como ventaja su condición de **superglobal**, que permite utilizarla en cualquier ámbito sin necesidad de declararla expresamente como **global**. Es la *opción del futuro*. Su único inconveniente es que puede no estar disponible en hostings que aún mantienen versiones antiguas de PHP.

Envío a través de formularios

La interacción *cliente–servidor* que acabamos de ver, resulta incómoda en su uso y no demasiado estética. Hay una segunda opción –la de uso más frecuente– que es la utilización de formularios.

Los formularios no son elementos propios de PHP –actúan del lado del **cliente**– y su estudio es más propio del ámbito de HTML que de PHP.

Interpretación de los datos recibidos a través de formularios

Igual que ocurría en el caso anterior, los datos enviados a través de un formulario son recogidos en diferentes tipos de variables predefinidas, pero ahora se añade una nueva particularidad. Existe la posibilidad de dos métodos (**method**) de envío: '**GET**' y '**POST**'.

En el caso anterior decíamos que se utilizaba el método **GET**, pero en el caso de los formularios son posibles ambos métodos. Conviene tenerlo en cuenta.

Método GET

No se diferencia en nada del descrito para el supuesto anterior. Utiliza las mismas variables predefinidas, las utiliza con idéntica sintaxis y se comporta de igual forma en lo relativo a las opciones de **register_globals**. Los nombres de las variables son en este caso, los incluidos como **name** en cada una de las etiquetas del formulario. Respecto a los valores de cada variable, éstos serían los recogidos del formulario. En los casos de ***campos tipo: text, password y*** *textarea* serían los valores introducidos por el usuario en cada uno de esos campos.

En el caso de los campos tipo *radio* –en el que varias opciones pueden tener el mismo nombre– recogería el valor indicado en la casilla marcada; mientras que si se trata de campos tipo *checkbox* se transferirían únicamente las variables –y los valores– que corresponden a las casillas marcadas.

Si se tratara de un campo tipo *hidden* se transferiría el valor contenido en su etiqueta y, por último, en el caso del *select* sería transferido como valor de la variable la parte del formulario contenida entre las etiquetas <option></option> de la opción seleccionada.

Método POST

En el caso de que el método de envío sea POST hay una diferencia a tener en cuenta en cuanto a las variables que recogen la información. Ahora será:

$HTTP_POST_VARS['*n1*']

quien haga la función atribuida en el método anterior a: $HTTP_GET_VARS['*n1*'] y ocurrirá algo similar con las superglobales, que pasarían a ser del tipo:

$_POST['*n1*']

en sustitución del $_GET['*n1*'] usado en el caso del método GET Si **register_globals** está en **On** el comportamiento de las *variables directas* es idéntico con ambos métodos.

Identificación del método de envío

PHP recoge en una variable el método utilizado para enviar los datos desde un formulario. Se trata de la variable **REQUEST_METHOD**. Puede ser invocada como una *variable directa* (en caso de que **register globals** esté en **on**) o a través de una de las **variables de servidor**.

En el primer caso la variable se llamaría: **$REQUEST_METHOD** y en el segundo: **$HTTP_SERVER_VARS[REQUEST_METHOD]**

Cuando PHP permita el uso de variables **superglobales** se puede ***utilizar:*** **$_SERVER[REQUEST_METHOD]**

Una **advertencia** importante: Observa que en este caso **no se incluyen comillas** dentro del corchete como ocurría con todos los nombres de variable anteriores.

Diferencias ente los métodos GET y POST

Las diferencias entre uno y otro método son las siguientes:

Método GET

Las particularidades de este método son las siguientes:

– Al ser enviado el formulario se *carga* en el navegador la dirección especificada como **action**, se le añade un **?** y a continuación se incluyen los datos del formulario. Todos los datos de la petición **van a ser visibles** desde la *barra de direcciones del navegador*.
– Únicamente son aceptados los caracteres ASCII.
– Tiene una limitación en el tamaño máximo de la cadena que contiene los datos a transferir. En IE esa limitación es de 2.083 caracteres.

Método POST

No tiene las limitaciones indicadas para el caso de **GET** en lo relativo a **visibilidad** ni en cuanto a aceptación de caracteres no ASCII. Este método de transferencia de datos es el más habitual cuando se utilizan formularios.

Tipos de contenidos de los formularios

Respecto a los formularios, vamos a contemplar un último aspecto: la *forma de encriptar* de los datos en el momento de la transmisión (**ENCTYPE**). Puede especificarse dentro de la etiqueta**<form>** utilizando la sintaxis: **enctype**='*valor*'.

En el caso de que no se especifique, tomará el valor **application / x-wwwformurlencoded**, sea **GET** o **POST** el método que se utilice. El método **POST** admite **multipart/form-data** como una opción alternativa a la anterior. Suele utilizarse cuando se trata de enviar grandes cantidades de datos, formularios en los que se adjuntan ficheros, datos *no ASCII* o contenidos *binarios*. ***Las diferencias***

básicas entre ambos modos de encriptación son las siguientes:

En el tipo **application/x-www-formurlencoded** los nombres de control y los valores se transforman en *secuencias de escape*, es decir, convirtiendo cada byte en una cadena **%HH**, donde **HH** es la *notación hexadecimal del valor del byte*. Además, los *espacios* son convertidos en **signos +**, los *saltos de línea* se representan como **%0D%0A**, el *nombre y el valor se separan con el* signo **=** y los *diferentes bloques nombre/valor*, se separan con el carácter **&**.

En cuanto a la encriptación tipo **multipart/form-data**, sigue las reglas de las transferencias MIME, que comentaremos más adelante cuando tratemos el tema del correo electrónico.

¡Cuidado!

Cuando se incluye una *cadena vacía* ("") como valor de **action** en un formulario se recargará el mismo documento como respuesta al envío del formulario.

La seguridad en los envíos de datos

El tema de la seguridad es una preocupación constante entre los usuarios de Internet. Cuando utilizamos las técnicas que venimos comentando en esta página –nos referimos siempre al caso de servidores remotos– corremos dos tipos de riesgo de seguridad que no estaría de más tener en cuenta. El riesgo de que la información sea interceptada durante el proceso de transmisión desde el cliente hasta el servidor lo compartimos con todos los demás usuarios de la Red, pero hay otro –el *riesgo de daños en los contenidos de nuestro espacio de servidor*– que es **exclusivamente** nuestro.

La *transparencia* del método GET es tal, que incluso muestra –en el momento del envio– todos los datos en la barra de direcciones del navegador. Eso permite que cualquier usuario pueda conocer a simple vista la ruta completa hasta el script, así como los nombres y valores de las variables.

Cuando se usa el método POST los formularios son un poco más ***discretos, pero igual de*** transparentes. El *código* de cualquier formulario estará accesible sólo con ir a la opción *Ver código fuente* y allí estarán de nuevo todos los datos: nombre del script, nombres de las variables, etcétera, con lo que, cualquier usuario y desde cualquier sitio, puede acceder a ese script. No haría falta ni usar *nuestro* formulario. Bastaría guardar una copia del mismo en el ordenador del *visitante* y después –haciendo ligerísimos retoques– se podría acceder a nuestro script sin necesidad de utilizar el formulario alojado en nuestro servidor.

Si pensamos que uno de nuestros scripts puede estar diseñado con el fin de modificar algunos de los contenidos de nuestro espacio –**borrar** datos, por ejemplo– seguramente sería cuestión de empezar a preocuparnos, y mucho más si en nuestro servidor tenemos datos *importantes*. Existen formas de evitar, o al menos reducir, este tipo de riesgos. Restringir a usuarios autorizados el uso de algunos subdirectorios es una de ellas, almacenar datos importantes fuera del directorio root del servidor es otra y el uso de algunas de las variables predefinidas como elementos de protección puede ser una tercera.

Hacemos este comentario a *título meramente informativo*. Por el momento nos basta con manejar los formularios, pero queremos que tengas presente la existencia de ese tipo de riesgos. Más adelante, veremos la manera de *tratar de evitarlos*.

¿Que valor tiene register_globals en tu php.ini?

Cuando hablábamos de la configuración de PHP, no hicimos ninguna alusión a la directiva **register_globals** que tenía dos opciones de configuración: ON y OFF. Según el valor que tenga esa configuración cambiará el modo de afrontar diversos *asuntos*.

En las últimas versiones de PHP viene configurado por defecto **register_globals=OFF**, pero no está de más el comprobarlo.

Tabla de multiplicar

```
<?
/* Escribamos una instrucción, que imprima en pantalla
el valor de una variable ($a), una "x" que hará funciones de aspa
en la presentación, otra variable ($b), el signo igual
y $a*$b que es el producto de ambas variables
El simbolo de multiplicar es (*) */
print ($a." x ".$b." = ".$a*$b);
# Añadamos una bobadilla, para animar... ;-)
print ("<br> ¡¡Ya eres mayorcit@.. Deberías saberte la tabla");
?>
```

Ejecuta este ejemplo y observa lo que aparece. ¿Sólo x =0? ¿y la *otra línea*?

El resultado es lógico. No hemos asignado valores a las variables **$a** y **$b** y por eso no escribió su valor. Sin embargo, a la hora de multiplicar -recuerda que una variable vacía es interpretada como cero a la hora de hacer operaciones– la interpretó como cero y –con muy buen criterio– nos respondió que cero por cero es cero.

Escribamos ahora en la barra de direcciones del navegador la siguiente dirección: http://localhost/ejemplo13.php?a=21&b=456

```
<?
/* Modifiquemos la instrucción anterior
```

```
y utilicemos las variables predefinidas
$HTTP_GET_VARS['a'], $HTTP_GET_VARS['b']
en vez de $a y $b que utilizabamos en el ejemplo anterior */
# Pongamos un comentario de advertencia. Recuerda que <br>
# sirve para insertar un salto de línea en la salida
print ("Este resultado es el que utiliza $HTTP_GET_VAR<br>");
print ($HTTP_GET_VARS['a']." x ".$HTTP_GET_VARS['b']." = ".
$HTTP_GET_VARS['a']*$HTTP_GET_VARS['b']);
/* Ahora trataremos de comprobar que también podemos
utilizar la superglobal $_GET como $_GET['a'] y $_GET['b']
con iguales resultados que las anteriores */
# Un comentario para identificar el origen del resultado
print("<br>El resultado siguiente ha sido generado usando $_GET <br>");
print ($_GET['a']." x ".$_GET['b']." = ".$_GET['a']*$_GET['b']);
?>
```

Un formulario

```
<html>
<head>
</head>
<body>
<!-- Un formulario debe empezar siempre con una etiqueta de este tipo
<form ...> en la que será obligatorio indicar con esta sintaxis
action='nombre.extension'
nombre.extension debe contener el nombre (o la ruta completa
en el caso de que estuviera en un directorio o hosting distinto
del que alberga el documento que contiene el formulario desde
el que se realiza la petición)
Es opcional incluir method que puede tener dos valores
method='GET' ó method='POST', por defecto (cuando no se indica)
el envio se realizará usando method='GET'
También es opcional -a los efectos de PHP- incluir name.
Ese valor es útil cuando se incluyen scripts del lado del cliente
del tipo JavaScript //-->
<form name='mi_formulario' action='formu1.php' method='post'>
<!-- Pueden incluirse textos dentro del formulario -->
Escribe tu nombre:
<!-- Uno de los tipos de campos posibles es el tipo texto
su sintaxis (hablamos de HTML) requiere la etiqueta
<input type='text'> que indica el contenido del texto
esa etiqueta debe incluir obligatoriamente un name='nombre'
el nombre a usar serán caracteres alfabéticos, sin tildes
ni eñes y sin espacios. Salvo excepciones que comentaremos
no puede usarse el mismo nombre para dos campos distintos
el value='' puede no contener nada entre las comillas
tal como ocurre aquí o contener el texto que por defecto queremos
que aparezca en ese campo al cargar el formulario.
el size=xx es opcional. Su utilidad es la de ajustar el
tamaño de la ventana al número de caracteres que se indiquen //-->
<input type='text' name='nombre' value='' size=15><br>
Escribe tu clave:
<!-- <input type='password'> solo se diferencia del anterior
en que en el momento de rellenarlo se sustituyen los carecteres
visualizados (no el contenido) por asteriscos //-->
<input type='password' name='clave' value=''><br>
Elige tu color de coche favorito:<br>
<!-- Los <input type='radio'> permite optar entre varios
valores posibles. Habrá que repetirlos tantas veces como
opciones queramos habilitar.
Todos los input -correspondientes a la misma opción-
deben tener el mismo nombre (name)
value='loquesea' deberá tener un valor
distinto en cada uno de ellos. Ese valor (loquesea)
será transferido a través del formulario
Si queremos que una opción aparezca marcada (por defecto)
al cargar el formulario, deberemos incluir en su etiqueta
la palabra checked
los contenidos de value no se visualizan en el navegador
por lo que conviene incluir una descripción de los
valores después de cerrar la etiqueta de cada input
Al enviar el formulario solo se transmite el value
correspondiente a la opción seleccionada //-->
<input type='radio' name='color' value='Rojo'>Rojo</br>
```

```
<input type='radio' checked name='color' value='Verde'>Verde</br>
<input type='radio' name='color' value='Azul'>Azul</br>
Elige los extras:<br>
<!-- Cada uno de los <input type='checkbox'>
requiere un nombre distinto (name) y un valor (value)
permite optar entre varios
Esos valor (loquesea)
serám transferido a través del formulario
cuando la casilla de verificación esté marcada
Si queremos que una casilla aparezca marcada (por defecto)
al cargar el formulario, deberemos incluir en su etiqueta
la palabra checked
los contenidos de value tampoco aquí
se visualizan en el navegador
por lo que conviene incluir una descripción de los
valores después de cerrar la etiqueta de cada input
Al enviar el formulario solo se transmite los value
correspondiente a la opcion marcadas //-->
<input type='checkbox' name="acondicionado" value="Aire">
Aire acondicionado<br>
<input type='checkbox' checked name="tapiceria" value="Tapicieria">
Tapiceria en piel<br>
<input type='checkbox' name="llantas" value="aluminio">
Llantas de aluminio<br>
¿Cual es el precio máximo<br>
que estarías dispuesto a pagar?
<!-- La etiqueta <input type='select'>
requiere un nombre
y requiere también una etiqueta de cierre
</select>
Entre ambas -apertura y cierredeben
incluirse las diferentes opciones
entre las de etiquetas
<option>valor<option>
Al enviar el formulario se transmite lo
contenido después de opción
en la opción seleccionada
si dentro de una etiqueta option
escribimos selected será esa
la que aparezca por defecto al cargarse el formulario//-->
<select name="precio">
<Option>Menos de 6.000 euros</option>
<Option>6.001 - 8.000 euros</option>
<Option selected >8.001 - 10.000 euros</option>
<Option>10.001 - 12.000 euros</option>
<Option>12.001 - 14.000 euros</option>
<Option>Más de 14.000 euros</option>
</select>
<!-- Las áreas de texto deben tener una etiqueta de apertura
<textarea name='checkbox'>
seguida de una etiqueta de cierre </textarea>
Dentro de la etiqueta de apertura puede incluirse
rows=xx (indicará el número de filas)
cols=yy (indicará el ancho expresado en número de caracteres)
y opcionalmente un value='lo que sea...'
que puede contener el texto que -por defectopretendemos
que aparezca en ese espacio
en el momento de cargar rl formulario //-->
<br> Escribe aquí cualquier otro comentario:<br>
<textarea rows=5 cols=50 name='texto'></textarea><br>
<!-- El <input type='hidden'>
permite insertar en un formulario una valor oculto
que no requiere ser cumplimentado por el usuario
y que no aparece visible en el documento
requiere un name y un value //-->
<input type="hidden" name='oculto' value='Esto iría oculto'><br>
<!-- El <input type='submit'>
es el encargado de ejecutar la action
incluida en la etiqueta de apertura del formulario
que en este caso sería la llamada
a la página que se indica en la action
El texto que incluyamos en value='enviar...'
será el que se visualice en el propio botón de envio //-->
<input type="submit" value="enviar">
<!-- El <input type='reset'>
```

```
permite borrar todos los contenidos
del formulario y reestablecer los valores
por defecto de cada campo //-->
<input type="reset" value="borrar">
<!-- La etiqueta </form>
es la etiqueta de cierre del formulario //-->
</FORM>
</BODY>
</HTML>
```

No incluiremos la opción *Ver ejemplo* hasta que hayamos elaborado los documentos **formu1.php** incluidos en la **action** de este formulario.

Scripts para recoger los datos del formulario anterior

Insertaremos ahora los diferentes tipos scripts utilizables, especificando las condiciones de utilización de cada uno de ellos. Sería buena idea que *experimentaras* con ellos tanto bajo register_globals=ON como cuando esté en modo OFF.

Este primero funcionará tanto cuando el método sea POST como cuando sea GET.

Requiere que el **php.ini** contenga la opción **register_globals=ON**

```
<?
echo "El method que ha usado fué: ",$REQUEST_METHOD,"<br>";
echo $nombre,"<br>";
echo $clave,"<br>";
echo $color,"<br>";
echo $acondicionado,"<br>";
echo $tapiceria,"<br>";
echo $llantas,"<br>";
echo $precio,"<br>";
echo $texto,"<br>";
echo $oculto,"<br>";
?>
```

Este otro requiere que el método especificado en el formulario de envío sea POST.
El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$HTTP_SERVER_VARS[REQUEST_METHOD],"<br>";
echo $HTTP_POST_VARS['nombre'],"<br>";
echo $HTTP_POST_VARS['clave'],"<br>";
echo $HTTP_POST_VARS['color'],"<br>";
echo $HTTP_POST_VARS['acondicionado'],"<br>";
echo $HTTP_POST_VARS['tapiceria'],"<br>";
echo $HTTP_POST_VARS['llantas'],"<br>";
echo $HTTP_POST_VARS['precio'],"<br>";
echo $HTTP_POST_VARS['texto'],"<br>";
echo $HTTP_POST_VARS['oculto'],"<br>";
?>
```

Para utilizar eficazmente este script es necesario que el método especificado en el formulario de envío sea **GET**.
El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$HTTP_SERVER_VARS[REQUEST_METHOD],"<br>";
echo $HTTP_GET_VARS['nombre'],"<br>";
echo $HTTP_GET_VARS['clave'],"<br>";
echo $HTTP_GET_VARS['color'],"<br>";
echo $HTTP_GET_VARS['acondicionado'],"<br>";
echo $HTTP_GET_VARS['tapiceria'],"<br>";
echo $HTTP_GET_VARS['llantas'],"<br>";
echo $HTTP_GET_VARS['precio'],"<br>";
echo $HTTP_GET_VARS['texto'],"<br>";
echo $HTTP_GET_VARS['oculto'],"<br>";
?>
```

Este otro requiere que el método especificado en el formulario de envío sea POST y que la versión de PHP soporte variables **superglobales**. El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$_SERVER[REQUEST_METHOD],"<br>";
echo $_POST['nombre'],"<br>";
echo $_POST['clave'],"<br>";
```

```
echo $_POST['color'],"<br>";
echo $_POST['acondicionado'],"<br>";
echo $_POST['tapiceria'],"<br>";
echo $_POST['llantas'],"<br>";
echo $_POST['precio'],"<br>";
echo $_POST['texto'],"<br>";
echo $_POST['oculto'],"<br>";
?>
```

En este supuesto sería necesario que el método especificado en el formulario de envío sea GET y que la versión de PHP instalada en el servidor que lo aloja soporte variables **superglobales**. El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method que ha usado fué: ",$_SERVER[REQUEST_METHOD],"<br>";
echo $_GET['nombre'],"<br>";
echo $_GET['clave'],"<br>";
echo $_GET['color'],"<br>";
echo $_GET['acondicionado'],"<br>";
echo $_GET['tapiceria'],"<br>";
echo $_GET['llantas'],"<br>";
echo $_GET['precio'],"<br>";
echo $_GET['texto'],"<br>";
echo $_GET['oculto'],"<br>";
?>
```

La variable $_REQUEST

PHP también dispone –a partir de su versión 4.1.0– de la variable **$_REQUEST** (de tipo superglobal) que aúna las funcionalidades de **$_GET** y **$_POST** y que recoge en variables del tipo **$_REQUEST['***nombre***']** tanto los valores transferidos mediante el método **GET** como mediante **POST. $_REQUEST**, a diferencia de $_GET y $_POST, no dispone de equivalentes en versiones anteriores. Ello quiere decir, **no existe** una variable (no superglobal) del tipo $HTTP_REQUEST_VARS con $HTTP_GET_VARS o con $HTTP_POST_VARS.

En este ejemplo hemos incluido dos scripts que solo solo se diferencian en el *method* especificado en el formulario. **Al ejecutarlos podremos comprobar que, independientemente del método usado, $_REQUEST** recoge los valores transferidos desde el formulario.

```
<?
/* Al ejecutar por primera vez el script la variable pepe será nula
ya que no se ha transferido aún el formulario. Al pulsar sucesivamente
en el botón Enviar iremos visualizando los valores que se vayan
transfiriendo */
print "He recibido la variable pepe con valor: ".$_REQUEST['pepe'];
/* al enviar el formulario se recargará este mismo documento
ya que hemos puesto action="" */
?>
<form name="prueba" method="post" action="">
<input type="text" name="pepe" value=''>
<input type="submit" value="enviar">
```

Capítulo 9: Operaciones aritméticas

Operaciones aritméticas

En páginas anteriores hemos podido ver que PHP permite utilizar un tipo de variables –las numéricas– cuyos valores puedan ser *operados* de la misma forma que se hace con los números en la vida cotidiana.

Los resultados de las operaciones pueden utilizarse de *forma directa* o ser recogidos en una *nueva variable*. Si asignamos a una nueva variable el resultado de una operación el valor contenido en ella no se modifica, aunque cambien los de las variables que intervinieron su creación.

Sintaxis de print y echo

Si queremos *encadenar* en una sola instrucción –*echo* ó *print*– el resultado de una operación junto con otras variables (o cadenas) es **imprescindible** poner entre paréntesis las instrucciones de la operación.

Esta norma, solo tiene dos excepciones: en caso de que el *print solo contenga la propia operación* o cuando utilicemos *echo* y el *separador* sea una **coma**.

Operadores aritméticos

Suma

$a + $b

Diferencia

$a - $b

Producto

$a * $b

Cociente

$a / $b

Cociente entero

(int)($a / $b)

Resto de la división

$a % $b

Raíz cuadrada

Sqrt($a)

Potencia a^b

pow($a,$b)

Raíz (de índice b) de a

pow($a,1/$b)

Redondeo de resultados

PHP tiene **tres** opciones de redondeo:

Redondeo por defecto

floor($z)

Redondeo por exceso

ceil($z)

Redondeo tradicional

round($z)

Al realizar una operación cuyo resultado **no es un número real** PHP devuelve la cadena **-1.#IND**.

Orden de operación

Cuando una misma instrucción contiene una secuencia con varias operaciones el orden de ejecución de las mismas sigue los mismos criterios que las matemáticas. No se realiza una ejecución secuencial sino que se respeta el *orden de prioridad matemático*. Es decir, las potencias y raíces tienen prioridad frente a los productos y los cocientes, y estos, son prioritarios respecto a la suma y las diferencias.

Igual que en matemáticas se pueden utilizar los paréntesis para *modificar* el orden de ejecución de las operaciones, e igual que *allí* PHP también permite encerrar paréntesis dentro de paréntesis.

Operaciones aritméticas

```
<?
# definamos dos variables numéricas asignandoles valores
$a=23; $b=34;
/* hagamos una suma y escribamos directamente los resultados
utilizando las instrucciones print y echo
con todas sus posibles opciones de sintaxis */
print("La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b)."<br>");
print "La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) ."<BR>";
print ("La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) ."<BR>");
echo "La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) ."<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b . "=" , ($a+$b) ."<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b , "=" , $a+$b ,"<BR>";
# guardemos ahora el resultado de esa operación en una nueva variable
$c=$a+$b;
/*ahora presentemos el resultado utilizando esa nueva variable
adviertiendo el la salida */
print ("Resultados recogidos en una nueva variable<br>");
print "La suma de $a + $b es: " . $a . "+" . $b . "=" . $c ."<BR>";
print ("La suma de $a + $b es: " . $a . "+" . $b . "=" . $c ."<BR>");
echo "La suma de $a + $b es: " . $a . "+" . $b . "=" . $c ."<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b . "=" , $c ."<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b , "=" , $c ,"<BR>";
/* modifiquemos ahora los valores de $a y $b comprobando que el cambio
no modifica lo contenido en la variable $c */
$a=513; $b=648;
print ("<br> C sigue valiendo: " . $c ."<br>");
# experimentemos con los paréntesis en un supuesto de operaciones combinada
# tratemos de sumar la variable $a con la variable $b
# y multiplicar el resultado por $c.
# Si escribimos print($a+$b*$c) nos hará la multiplicación antes que la suma
print "<br>No he puesto paréntesis y el resultado es: ".($a+$b*$c);
# Si escribimos print(($a+$b)*$c) nos hará la suma y luego multiplicará
print "<br>He puesto paréntesis y el resultado es: ".(($a+$b)*$c);
?>
```

Capítulo 10: Números Aleatorios

El origen de la semilla

El valor *Unix Epoch* El conocido como *tiempo UNIX* –o también *Unix Epoch*- es un sistema referencia de tiempo cuya unidad son los segundos y que tiene su valor cero a las **0:00:00 horas (GMT)** del día **uno de enero de 1970**.

La funcion *time()*

La función time() devuelve **una cadena** con el número de segundos transcurridos desde el comienzo de la **Unix Epoch**. *¿Que cuantos son?* Pues mira, hasta este mismo instante han transcurrido: 1 243 523 762 segundos desde el comienzo del *tiempo UNIX*.

La funcion *microtime()*

Usando la función microtime() se obtiene **una cadena** a la que, además del número de segundos transcurridos desde el comienzo de la **Unix Epoch**, se añade – al comienzo de ella y separado por un espacio– la parte decimal de ese tiempo expresada en microsegundos.

Este es el valor de la cadena devuelta por microtime() en este instante:

0.39062800 1243523762

donde, como verás, aparece la fracción decimal del tiempo *Unix Epoch* delante de su valor entero.

(double)microtime()

Dado que con **microtime()** obtenemos *una cadena*, es posible convertirla en número de coma flotante anteponiendo a esa función **(double)**. Una vez cambiado el tipo de variable el valor devuelto por **microtime** se convierte en:

0.390663

que como puedes observar es un número con *seis decimales* (si las últimas cifras son ceros no se visualizarán).

Si se multiplica el valor anterior por **1.000.000** obtenemos un número de **seis cifras** y valor *entero* que puede cambiar de valor cada **millonésima *de segundo.***

390683

¡Fíjate que ha cambiado! La diferencia no es otra cosa que el tiempo transcurrido entre los instantes en que se ejecutaron ambas instrucciones.

Este número entero es utilizado por **srand** y **mt_rand** como *semilla* generadora de números aleatorios.

Números aleatorios

PHP dispone de dos funciones capaces de generar números aleatorios. Se trata de la función **rand()** y de la *función mejorada* **mt_rand()**.

El valor **mínimo** del intervalo que contiene los números aleatorios que pueden ser generados es **CERO** en ambos casos y los valores **máximos** de cada opción pueden determinarse mediante las funciones **getrandmax()**, para el primer generador, y **mt_getrandmax()**, para el segundo.

Veamos cuales son esos valores en cada uno de los casos.

Valores máximos de los generadores de números aleatorios			
Generador **rand()**		Generador **mt_rand()**	
Sintaxis	Valor máximo	Sintaxis	Valor máximo
echo getrandmax()	**32767**	**echo mt_getrandmax()**	**2147483647**

Según las librerías que esté usando PHP, puede ocurrir que los valores máximos con ambos generadores sean iguales.

La forma más simple

La forma más simple -y más desaconsejable- de generación de un número aleatorio es esta:

Generación de un número aleatorio			
Generador **rand()**		Generador **mt_rand()**	
Sintaxis	Nº aleatorio	Sintaxis	Nº aleatorio
echo rand()	**26474**	**echo mt_rand()**	**299791159**

Una *semilla* que mejora la aleatoriedad

Si antes de escribir la función **rand()** en un caso, o **mt_rand()** en el otro escribimos: **srand((double)microtime()*1000000)**, en el primer caso y/o **mt_srand((double)microtime()*1000000)**, en el segundo estaremos introduciendo una *semilla* (al margen comentamos la forma en que se genera) que mejora sustancialmente la aleatoriedad del los números obtenidos.

Aquí tenemos un ejemplo:

Generación de un número aleatorio con *semilla*	
Generador **rand()**	
Sintaxis	Nº aleatorio
srand((double)microtime()*1000000); **echo rand()**	**11892**
Generador **mt_rand()**	
Sintaxis	Nº aleatorio
mt_srand((double)microtime()*1000000); **echo mt_rand()**	**1186193619**

Manteniendo la sintaxis anterior -no te olvides de las *semillitas famosas*- se pueden generar números aleatorios comprendidos dentro del intervalo que preestablezcamos.

Bastaría con añadir los valores de los extremos de ese intervalo como parámetros de la función.

La sintaxis sería esta:

rand(*extremo inferior* , *extremo superior*)

y para la **función mejorada**

mt_rand(*extremo inferior* , *extremo superior*)

Generación de un número aleatorio delimitando intervalos	
Generador **rand()**	
Sintaxis	Nº aleatorio
srand((double)microtime()*1000000); **echo rand(1,300)**	**104**
Generador **mt_rand()**	
Sintaxis	Nº aleatorio
mt_srand((double)microtime()*1000000); **echo mt_rand(1,300)**	**75**

Capítulo 11: Array escalar y asociativo

¿Qué es un array?

Un *array* es sencillamente una tabla de valores. Cada uno de los elementos de esa *tabla* se identifica por medio de un **nombre** (común para todos) y un **índice** (que diferenciaría a cada uno de ellos).

La sintaxis que permite definir elementos en un array es esta:

$nombre[*indice***]**

$nombre utiliza exactamente la misma sintaxis empleada para definir variables, con la única particularidad de que ahora deben añadírsele los corchetes y los índices. El *índice* puede ser *un número* (habría que escribirlo dentro del corchete *sin comillas*), *una cadena* (que habría que poner en el corchete encerrada entre *comillas sencillas* –'–), o una variable PHP en cuyo caso tampoco necesitaría ir entre comillas.

Cuando los *índices* de un array son *números* se dice que es **escalar** mientras que si fueran *cadenas* se le llamaría array **asociativo**.

Arrays escalares

Los elementos de un *array* escalar puede escribirse con una de estas sintaxis:

$a[]=*valor*

ó

$a[xx]=*valor*

En el primero de los casos PHP asigna los índices de forma automática atribuyendo a cada elemento el valor **entero** *siguiente* al último asignado.

Si es el **primero** que se define le pondrá índice **0** (CERO). En el segundo de los casos, seremos nosotros quienes pongamos (**xx**) el **número** correspondiente al **valor del índice**.

Si ya existiera un elemento con ese índice, se cambiaría el valor de su contenido, en caso contrario creará un nuevo elemento del *array* y se le asignaría como valor lo especificado detrás del signo igual, que –de las misma forma que ocurría con las variables– debería ir entre comillas si fuera una cadena o sin ellas, si se tratara de números.

Arrays asociativos

Los elementos de un *array* asociativo pueden escribirse usando la siguiente sintaxis:

$a[*'indice'*]=*valor*

En este caso estamos obligados a escribir el nombre del índice que habrá de ser una **cadena** y debe ponerse entre comillas. Tanto en este supuesto como en el anterior, es posible –y bastante frecuente– utilizar como índice el contenido de una variable. El modo de hacerlo sería:

$a[$ind]=*valor*

En este caso, sea cual fuere el valor de la variable *$ind*, el nombre de la variable **nunca** se pone entre comillas.

Tablas (*arrays*) unidimensionales

Mediante el uso de arrays podemos utilizar el **mismo nombre** para varias variables **diferenciándolas entre sí** mediante **índices distintos**

Tablas unidimensionales					
Array escalar			Array asociativo		
Variable	Indice	Valor	Variable	Indice	Valor
$a[0]	0	Domingo	$a['Primero']	Primero	Domingo
$a[1]	1	Lunes	$a['Segundo']	Segundo	Lunes
$a[2]	2	Martes	$a['Tercero']	Tercero	Martes
$a[3]	3	Miércoles	$a['Cuarto']	Cuarto	Miércoles
$a[4]	4	Jueves	$a['Quinto']	Quinto	Jueves
$a[5]	5	Viernes	$a['Sexto']	Sexto	Viernes
$a[6]	6	Sábado	$a['Septimo']	Septimo	Sábado

Uso de arrays

```php
<?
# Crearemos un array escalar (basta con definir un elemento)
$a[2]="Este elemento es el segundo del array";
# creemos un nuevo elemento de ese array
# esta vez de forma automática
# si ponemos corchetes vacíos va añadiendo índices automaticamente
$a[]="¿Será este tercero?";
# comprobemos que le ha puesto índice 3
echo "El elemento ".$a[3]." tiene indice 3 (siguiente a 2) <br>";
# ahora insertemos un nuevo elemento con indice 32
$a[32]="Mi indice es 32";
# insertemos otro elemento de forma automática
$a[]= "¿Irá a parar al indice 33 este elemento?";
# la inserción se hará con indice 33, comprobémoslo
print "Vemos que contiene el elemento de indice 33 ...".$a[33]."<br>";
# ¿qué ocurrirá si pido que imprima el elemento 21 que nadie ha definido
# seguramente estará vacio, ¡¡comprobémoslo!!
print ("Aqui--> ". $a[21]. "<--- si es que hay algo<br>");
# ahora crearemos un nuevo array llamado $b
# insertémosle de forma automatica su PRIMER elemento
$b[]="Estoy empezando con el array b y mi indice será cero";
# comprobemos que efectivamente ha empezado con indice CERO
print ($b[0]."<br>");
# veamos ahora eso de los arrays asociativos
# creemos uno llamado $c con varios elementos
$c["objeto"]="coche";
$c["color"]="rojo";
$c["tamaño"]="ideal";
$c["marca"]= "Ferrari";
$c["precio"]="prohibitivo para un humilde docente";
#encadenemos variables para hacer una salida
# pondremos cadenas " " para que no aparezcan los textos
# pegados unos a otros..
$salida="<H2> El ". $c["objeto"] ." ".$c["marca"]." ".$c["color"];
$salida .=" tiene el tamaño ideal ".$c["tamaño"];
$salida .=" y su precio es ".$c["precio"];
$salida .="</H2>";
print $salida;
# sigamos experimentando ahora
# ¿qué ocurriría si nos olvidamos de poner nombre al indice
# e insertamos un corchete vacio ¿lo crearía?¿que indice pondria?
# probemos ....
$c[]="¿creará un array escalar nuevo y le pondrá indice cero?";
# tratemos ahora de visualizar esa variable
# probemos a escribir $c[0] porque PHP
# habrá entendido que queremos un array escalar
# y como no existe ninguno con ese nombre empezará por cero
# comprobémoslo
echo $c[0];
?>
```

Capítulo 12: Arrays bidimensionales

Arrays bidimensionales

Los *arrays bidimensionales* pueden entenderse como algo muy similar a una *tabla de doble entrada*. Cada uno de los elementos se identifica –sigue siendo válido el nombre único que se usaba en los unidimensionales – por un nombre ($*nombre*) seguido de dos ([]) que contienen los *índices* (en este caso son dos índices) del array. Los *índices* pueden ser de tipo **escalar** -*equivalen al número de fila y columna que la* ***celda*** *ocupa en la tabla*– o puede ser **asociativos** lo que equivaldría en alguna medida a usar como índices los *nombres de la fila y de la columna*.

¡Cuidado!

No dejes de tener en cuenta lo que hemos advertido al hablar de arrays unidimensionales. En este supuesto, también, se empiezan a numerar los arrays escalares a partir de **CERO**.

Arrays escalares

Los elementos de un *array* **bidimensional** escalar pueden escribirse usando una de estas sintaxis:

$a[][]=*valor*

o

$a[xx][]=*valor*

o

$a[][xx]=*valor*

o también

$a[xx][yy]=*valor*

En el primero de los casos PHP asigna automáticamente como **primer índice** el valor que sigue al último asignado y, si es el **primero** que se define, le pondrá como índice **0** (CERO). Sea cual fuere el valor de primer índice al **segundo** se le asignará **cero** ya que es en este mismo momento cuando se habrá creado el **primero** y, por tanto, *aún carecerá de elementos*.

En el segundo de los casos, asignamos un valor al **primer índice** (**xx**) y será el segundo quien se incremente en **una unidad** respecto al de valor más alto de todos aquellos cuyo **primer índice** coincide con el especificado.

La tercera opción es bastante similar a la anterior. Ahora se modificaría automáticamente el primer índice y se escribiría el contenido (xx) como valor del segundo.

En la cuarta de las opciones se asignan libremente cada uno de los índices (**xx** e **yy**) poniéndoles valores numéricos.

Arrays asociativos

Los elementos de un *array* asociativo **bidimiensional** se pueden escribir usando la siguiente sintaxis:

$a[*"indice1"*][*"indice2"*]=*valor*

En este caso, los índices serán **cadenas** y se escribirán entre comillas.

Arrays mixtos

PHP permite utilizar también arrays *mixtos*. Sería este el caso de que uno de ellos fuera escalar y el otro asociativo. Igual que ocurría con los unidimensionales, también aquí podemos utilizar valores de variables como índices.

Arrays bidimensionales

Como ejemplo de array bidimensional emplearemos una tabla de resultados de una *liga de fútbol* en la que intervienen **cinco equipos** que –como en toda liga que se precie– se juega a *doble partido*.

En este primer supuesto utilizaremos *arrays* **escalares**, por lo tanto los equipos serán identificados con números desde **cero** hasta **cuatro**.

```
<?
# rellenamos el array desde [0][0] hasta [0][4]
# la insercion automatica haria que este primero fuera [0][0]
$a[][]=" ";
# ahora pondremos cero como indice del primer array y dejemos que PHP
# nos vaya insertando automaticamente el segundo
$a[0][]="3-2";$a[0][]="5-3";$a[0][]="7-1";$a[0][]="0-2";
#ahora desde [1][0] hasta [1][4]
```

```
#este primero lo dejamos como automático en ambos indices
# de esta forma el primero tomará valor uno (siguiente al anterior)
# de forma automática
$a[][]="0-11";
# repetimos el proceso anterior
$a[1][]=" ";$a[1][]="2-1";$a[1][]="1-0";$a[1][]="1-2";
# y repetimos de nuevo, ahora crearia 2 como primer indice
$a[][]="0-0";
#insertariamos los restantes valores de indice 2
$a[2][]="1-3";$a[2][]=" ";$a[2][]="1-4";$a[2][]="2-0";
# nuevo incremento del primer indice
$a[][]="1-0";
# rellenamos
$a[3][]="6-3";$a[3][]="14-3 ";$a[3][]=" ";$a[3][]="1-0";
# nuevo y ultimo incremento de primer indice
$a[][]="1-1";
# rellenamos de nuevo
$a[4][]="2-3";$a[4][]="0-1 ";$a[4][]="1-1";$a[4][]="";
# como verás el proceso no tiene complicaciones, pero ... pesadillo si es
# ¿verdad que si tuviéramos una base de datos sería más fácil?
# estamos en ello, todo se andará...
# tendríamos que ver esos valores pero.. escribir "a mano"
# una tabla puede ser una tortura, así que mejor introducimos
# una bucle, otro recurso que estudiaremos pronto
# para esa labor repetitiva de mostrar en una tabla
# todos los datos del array
# Sería algo como esto
# creamos la etiqueta de apertura de una tabla
print ("<TABLE BORDER=2>");
# ahora dos bucles anidados (rojo uno, magenta el otro)
# para rellenar las celdas de cada fila (el magenta)
# y para insertar las etiquetas <TR> utilizaremos el rojo
for ($i=0;$i<5;$i++){
print("<tr>");
for($j=0;$j<5;$j++) {
print("<td>".$a[$i][$j]."</td>");
}
}
#ponemos la etiqueta de cierre de la tabla
print("</table>");
?>
```

Capítulo 13: Operadores bit a bit

Incluimos la sintaxis de este tipo de operadores a título meramente informativo. Rara vez será necesario utilizarlos en nuestras aplicaciones PHP.

Su utilidad suele limitarse a la gestión de periféricos y algunas operaciones de cálculo de carácter muy reiterativo en la que se puede conseguir un rendimiento muy superior a los operadores tradicionales.

En el ámbito propio del PHP pueden tener algún interés a la hora de elaborar rutinas para *encriptar* el código fuente de algunos scripts que por su importancia pueden requerir ese tipo de protección. Los que sí han de resultarnos de gran interés serán el resto de los operadores. Los iremos viendo en páginas sucesivas.

Operadores bit a bit

$A & $B

El operador **&** compara los valores binarios de cada uno de los bits de las cadenas $A y $B y devuelve **1** en el caso que ambos sean **1**, y **0** en cualquier otro caso.

Cuando las variables **$A** y **$B** son **cadenas** compara los valores binarios de los códigos ASCII de sus caracteres y devuelve los caracteres ASCII correspondientes al resultado de esa comparación.

$A | $B

Funciona de forma idéntica al anterior y *devuelve* **1** cuando **al menos** el valor de uno de los bits comparados es **1**, y devolverá **0** cuando **ambos** sean **0**.

$A ^ $B

Devuelve **1** cuando los bits comparados son **distintos**, y **0** cuando son **iguales**.

$A << $B

Realiza la operación **$A** * $2^{\$B}$. Hace el cálculo añadiendo **$B CEROS** (binarios) a la derecha de la cadena binaria $A.

$A >> $B

Divide el valor $A entre $2^{\$B}$. Hace la operación en la cadena *binaria* quitando **$B CEROS** (por la derecha) de la cadena $A.

~ $A

Invierte los valores de los bits de la cadena **$A** convirtiendo los CEROS en UNO y los UNO en CERO.

Manejando operadores bit a bit

Desarrollamos aquí algunos ejemplos de manejo de los operadores bit a bit.

El operador **&**					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
$a=12	1100	$A="12"	110001110010	$A1="Rs"	10100101110011
$b=23	10111	$B="23"	110010110011	$B1="aZ"	11000011011010
$a&$b=4	**100**	**$A&$B=02**	**110000110010**	**$A1&$B1=@R**	**10000001010010**
En los casos de cadenas hemos diferenciado en rojo el valor binario correspondiente al primer carácter. Esos valores binarios corresponden a la forma binaria del código ASCII de cada uno de los caracteres					

Puedes observar que el tratamiento es distinto cuando los mismos valores numéricos se asignan como entero y como cadena. Al asignarlos como cadena *opera* los valores binarios de los códigos ASCII de los caracteres, mientras que cuando se trata de números compara los valores de las expresiones binarias de los valores de cada uno de ellos

El operador **|**					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
$a=12	1100	$A="12"	110001110010	$A1="Rs"	10100101110011
$b=23	10111	$B="23"	110010110011	$B1="aZ"	11000011011010
$a\|$b=31	**11111**	**$A\|$B=33**	**110011110011**	**$A1\|$B1=s{**	**11100111111011**
Se comporta de forma idéntica al anterior en lo relativo a números y cadenas.					

El operador **^**					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
$a=12	1100	$A="12"	110001110010	$A1="Rs"	10100101110011
$b=23	10111	$B="23"	110010110011	$B1="aZ"	11000011011010
$a^$b=27	**11011**	**$A^$B=**	**000011000001**	**$A1^$B1=3)**	**01100110101001**
Los criterios de tratamiento de números y cadenas coinciden con los operadores anteriores.					

El operador **<<**					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
$a=12	1100	$A="12"	110001110010	$A1="Rs"	10100101110011
$b=2	10	$B=2	10	$B1=2	10
$a<<b=48	**110000**	**$A<<$B=48**	**110000**	**$A1<<$B1=0**	

El operador << **multiplica** el valor de la primera cadena por *2 elevado al valor de la segunda*.

Al ser un operador *matemático* solo tiene sentido cuando ambas variables son números naturales.

En las **cadenas alfanuméricas extrae los números que pudiera haber al comienzo** y, en **caso de no haberlos, toma valor cero**.