



estudios abiertos

SEAS

GRUPO SAN**VALERO**



PHP

3. Conceptos básicos de PHP

ÍNDICE

OBJETIVOS	117
INTRODUCCIÓN	118
3.1. Sintaxis PHP	119
3.1.1. Estructura y sintaxis en PHP	119
3.1.2. Encajar PHP con HTML	120
3.1.3. Editores de PHP	121
3.1.4. Mi primer Script	123
3.2. Variables	124
3.2.1. Concepto	124
3.2.2. Tipos de variables	124
3.2.3. Averiguar el tipo de una variable	127
3.2.4. Chequeando variables	127
3.2.5. Constantes	128
3.2.6. Variables por referencia	129
3.2.7. Variables de variables	130
3.2.8. Variables del sistema	130
3.2.9. Variables superglobales	131
3.2.10. Ámbito de las variables	132
3.3. Operadores	135
3.3.1. Operadores aritméticos	135
3.3.2. Operadores de concatenación	135
3.3.3. Operadores de asignación	136
3.3.4. Operadores de incremento y decremento	137
3.3.5. Operadores condicionales	138
3.3.6. Operadores de ejecución	138
3.3.7. Operadores lógicos	139
3.4. Estructuras de control	140
3.4.1. IF	140
3.4.2. IF ... ELSE	140
3.4.3. ELSEIF	141
3.4.4. Sintaxis alternativa de estructura de control	142
3.4.5. SWITCH	143
3.4.6. WHILE	145
3.4.7. DO ... WHILE	146
3.4.8. FOR	147
3.4.9. FOREACH	147
3.4.10. BREAK	148
3.4.11. CONTINUE	148

3.5. Arrays	150
3.5.1. Crear un array	150
3.5.2. Arrays asociativos	152
3.5.3. Arrays multidimensionales	152
3.5.4. Uso de arrays en bases de datos	153
3.5.5. Recuperar el contenido de un array	154
3.6. Funciones	158
3.6.1. Creación de funciones	158
3.6.2. Llamadas a funciones	158
3.6.3. Parámetros de las funciones	159
3.6.4. Paso de parámetros	159
3.6.5. Por valor	160
3.6.6. Operador return	162
3.6.7. Librerías de funciones para estructurar tu aplicación web	162
3.7. Include y Require	164
3.8. Manejo de fechas y tiempo	167
3.9. Redirección de página	172
3.10. Funciones isset() y unset()	173
RESUMEN	175

OBJETIVOS

- Conocer la sintaxis básica de PHP.
- Conocer los diferentes tipos de variables que podemos usar.
- Definir los distintos operadores y cómo nos pueden ayudar cada uno.
- Conocer las estructuras de control de flujo de un programa.
- Explicar el concepto de array y sus diferentes tipos, así como la manera de recorrerlos.
- Aprender a utilizar funciones para agrupar trozos de código.
- Realizar operaciones con fechas y horas.




3.1. Sintaxis PHP

En este capítulo vamos a dar los primeros pasos para ir conociendo la forma de trabajo del lenguaje PHP y lo finalizaremos con el desarrollo de nuestro primer script básico de PHP.

3.1.1. Estructura y sintaxis en PHP


El código de un archivo puede estar formado por los siguientes elementos:

- **HTML.** Código HTML para generar contenido estático.
- **Etiquetas PHP.** Le indicarán al servidor web donde empieza nuestro código PHP y donde va a termina, de tal forma que todo el código que se encuentre incluido entre ellas se interpretará como código PHP, y lo que se quede fuera se interpretará como código HTML. En el siguiente apartado veremos las diferentes etiquetas que podemos utilizar.
- **Instrucciones PHP.** Sirve para indicar al intérprete PHP qué es lo queremos que haga, mediante el uso de instrucciones propias de PHP. Una cuestión muy importante es que siempre tendrán que terminar con punto y coma “;”.
- **Espacios en blanco.** Pueden ser líneas nuevas (retornos del carro), espacios o tabuladores, por ejemplo. Serán ignorados por los navegadores y por el intérprete PHP.
- **Comentarios.** Para realizar aclaraciones sobre el código y una mejor comprensión del mismo. Suelen ser también utilizados para indicar la fecha de la última actualización del script. Podremos utilizar dos tipos de comentarios:
 - Utilizando los caracteres reservados /* y */. Haciendo uso de ellos, podremos hacer comentarios de varias líneas.

 EJEMPLO

```
/* Autor: Cristian
Script actualizado a fecha de 13 de Agosto de 2011
Este es un comentario de prueba
*/
```

- Utilizando los caracteres reservados # o //. Haciendo uso de ellos, podremos hacer comentarios en una sola línea.

 EJEMPLO

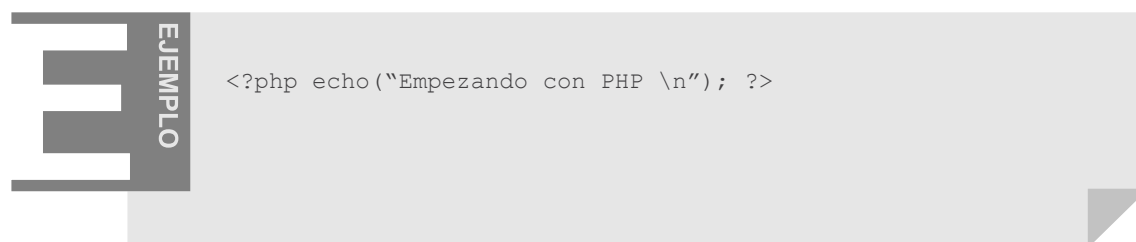
```
echo "Hola"; // comentario hasta final de línea
echo "Hola de nuevo"; #otro comentario hasta final de línea
```

3.1.2. Encajar PHP con HTML

Existen varias formas de poder incrustar el código PHP en un script de HTML. Para poder incluir el código necesitaremos hacer uso de **diferentes etiquetas PHP**. Veámoslas con unos cuantos ejemplos:

Estilo XML

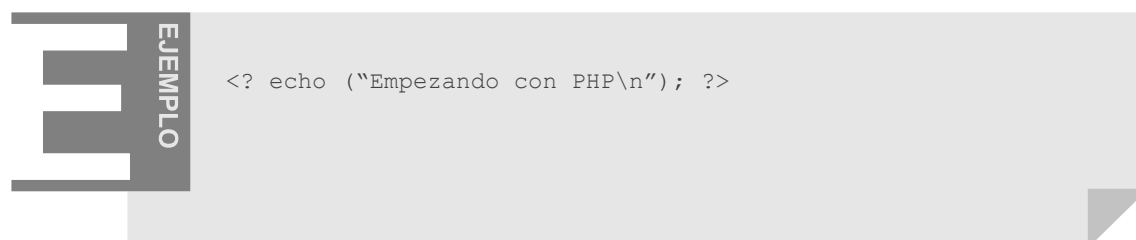
Suele ser el preferido porque no puede ser desactivado por el administrador del servidor, como veremos que sí que ocurre con la siguiente etiqueta, y esto nos asegurará su exitosa ejecución. Se denomina estilo XML porque puede ser utilizado en documentos XML.



Estilo corto

Para poder utilizar esta etiqueta tendremos que habilitar el parámetro ***“short_open_tag”*** en el archivo de configuración PHP.INI y volver a arrancar el servidor Apache. También podremos habilitarlas compilando PHP con la opción ***“enable-short-tags”*** en ***configure***.

Esta etiqueta no es recomendable porque a diferencia de la anterior, al necesitar esta configuración especial en los servidores donde estuviera alojado el script que lo incluyera en su programación, no se puede asegurar el éxito en la ejecución.



Estilo script

Esta etiqueta es algo más elaborada y, por lo tanto, más costosa de desarrollar, aunque si estás familiarizado con lenguajes como JavaScript o VBScript, estarás habituado a usarla.

R RECUERDA

```
<script language="php">
echo ("Empezando con PHP ");
</script>
```

Estilo ASP

Recibe este nombre porque tiene el mismo formato que para el lenguaje de programación ASP o ASP.NET, pero al igual que la etiqueta de estilo corto, será necesario tener activado un parámetro. En este caso, el parámetro **"asp_tags"**:

E EJEMPLO

```
<% echo ("Empezando con PHP "); %>
```

Por último, comentar que también podremos integrar código HTML en el interior de código PHP, como podemos ver en el siguiente ejemplo:

E EJEMPLO

```
<% echo ("Empezando con <strong>PHP</strong>"); %>
```

3.1.3. Editores de PHP

Antes de comenzar a estudiar y, por lo tanto, experimentar con código PHP, es fundamental encontrar la complicidad de un editor de código PHP que nos ayude en la tarea de desarrollo de código y en la localización de posibles errores.

La lista de editores actualmente es muy extensa, pero vamos a proponer algunos de ellos:

- **PHPedit.** Editor IDE (*Integrated Development Environment*) de PHP para Windows con coloreado de sintaxis, depurador php integrado, generador de ayudas, atajos de teclado personalizables, plantillas de teclado, generador de informes de tareas pendientes y diversos plugins.

<http://www.waterproof.fr/>

- **Maguma.** Maguma estudio es un IDE para PHP a nivel de código que ofrece ayudas y funcionalidades para acelerar el desarrollo de aplicaciones web.

<http://www.maguma.com/>

- **PHPdesigner.** Este entorno de desarrollo combina muchas potentes mejoras como los esquemas de sintaxis personalizable para PHP, HTML, XHTML, CSS, Perl, C#, JavaScript, VB, Java and SQL (Ingres, Interbase, MSSQL, MySQL, Oracle, Sybase and Standard SQL), un navegador de clases, interprete de PHP para depurar y probar los scripts, manual de php integrado, autosangrado de paréntesis, cierre automático de paréntesis y acceso a código común y librerías integrado.

<http://www.mpsoftware.dk/phpdesigner.php>

- **NuSphere PHPed.** Entorno de desarrollo profesional para PHP, con depurador, análisis de errores, ayudas para la localización de cuellos de botella en el código, publicación segura de código en servidores e integración con herramientas de terceros.

<http://www.nusphere.com/products/>

- **Dzsoft PHP editor.** Es una herramienta manejable y potente para escribir y probar páginas PHP y HTML. Su interfaz es cómoda y sencilla para novatos y programadores experimentados, haciendo el desarrollo de PHP fácil y productivo.

<http://www.dzsoft.com/>

- **PHPcoder.** Es un IDE especialmente diseñado para programadores de PHP, potente, reducido y libre. Dispone de un intérprete integrado y documentación que permite ahorrar tiempo en el desarrollo, coloreado de sintaxis HTML y PHP, autocompleta estructuras comunes, gestor de proyectos, undo ilimitado, copy y paste, búsquedas ...

<http://www.phpide.com/php-editors/php-coder/>

- **Eclipse Editor PHP.** Sin duda uno de los mejores entornos de desarrollo para PHP, Java, C/C++ y otros lenguajes, dispone de una potente administración de proyectos y ficheros, un gran editor con coloreado del lenguaje, detección y resaltado de errores sintácticos y de estructuras. Se trata de un entorno IDE cuya principal ventaja es la visualización de los errores de escritura, de inclusión de cabeceras, etc. Además, dispone del manual de PHP integrado y de rápido acceso.

<http://www.eclipse.org/downloads/index.php>

- **Macromedia Dreamweaver.** Adobe Dreamweaver es una aplicación en forma de estudio (basada en la forma de estudio de Adobe Flash) que está destinada a la construcción y edición de sitios y aplicaciones web basados en estándares. Creado inicialmente por Macromedia (actualmente producido por Adobe Systems) es el programa de este tipo más utilizado en el sector del diseño y la programación web, por sus funcionalidades, su integración con otras herramientas como Adobe Flash y, recientemente, por su soporte de los estándares del World Wide Web Consortium.

<http://www.adobe.com/la/products/dreamweaver>

3.1.4. Mi primer Script

Para terminar con este primer capítulo de introducción, vamos a poner en práctica las primeras cosas que hemos aprendido. Es decir, a insertar código PHP en HTML, introducir comentarios, visualizar una frase sencilla, e incluso llamar a una función típica de PHP que servirá para mostrarnos en pantalla información sobre la versión del intérprete PHP que estamos ejecutando, del servidor Apache y de MySQL, así como otra serie de funciones extras de PHP. Asimismo, al final del documento, nos encontraremos información acerca de la licencia de PHP, que como ya sabemos, se trata de un software gratuito.

Es importante conocer que todos los scripts que desarrollemos a partir de este momento, para que el servidor Apache pueda localizarlos e interpretarlos, deberán de almacenarse en una ruta en concreto. Como nosotros hemos instalado XAMPP, esta ruta es el directorio **htdocs**. Por ejemplo, en Windows sería **C:\XAMPP\HTDOCS**.

De la misma forma, tenemos que conocer que para poder ver el resultado de nuestros scripts, tendremos que introducir la ruta <http://localhost>. Esta dirección web enlaza directamente con el directorio **htdocs**, y muestra el fichero **index.html** de XAMPP por defecto.

Para comenzar, si, por ejemplo, estamos trabajando con XAMPP en Windows, deberíamos de crear el siguiente código en un archivo denominado **PHPINFO.PHP**, de tal forma que su ruta absoluta fuera **C:\XAMPP\HTDOCS\PHPINFO.PHP**:

EJEMPLO

```
<html>
<head><title>Mi primer script en PHP</title></head>
<body>
<p>Esto es una línea PHP</p>
<?php
echo 'Esto también es una línea de PHP';//comentario
/* la función phpinfo()
nos muestra información sobre el intérprete PHP que estamos
utilizando en el servidor*/
phpinfo();
?>
</body>
</html>
```

3.2. Variables

En este capítulo vamos a conocer las variables, qué tipos diferentes de variables podremos utilizar, así como otros aspectos relacionados con ellas.

3.2.1. Concepto

Una variable es un símbolo que es utilizado para almacenar un valor en memoria.

En PHP, todas las variables tendrán por delante de la palabra que utilicemos para definir la variable el símbolo “\$”. Por ejemplo, \$nombre. De esta forma, sabremos que estamos definiendo una variable.

Un lenguaje de programación no tendría sentido sin el uso de variables para poder realizar operaciones de todo tipo: aritméticas, con cadenas, etc.

Los **identificadores** serán los nombres de estas variables, y hay que tener en cuenta una serie de consideraciones respecto a ellos:

- Podrán tener cualquier longitud e incluir letras, números y guiones bajos.
- No podrán comenzar por un número. Por ejemplo, **\$23nombre**, sería un error.
- En PHP, se distinguirá entre mayúsculas y minúsculas. Por lo tanto, no será lo mismo **\$valor** que **\$VALOR**.
- Una variable sí podrá tener el mismo nombre que una función, pero como puede dar lugar a confusiones, es mejor evitarlo.
- Hay que tener especial cuidado en no utilizar palabras reservadas por PHP. Por ejemplo, sería un error definir la variable **\$os**, ya que es una variable predefinida y no puede ser utilizada.

En PHP no tiene por qué ser necesario concretar el tipo de variables, pero sí que será importante, por ejemplo, tener claro cuando utilizar comillas puesto que, a la hora de crearlas, en función de si las utilizamos o no, estaremos hablando de una cadena o no. Por lo tanto, no tenemos porque definir las de forma previa a su utilización, puede ser realizado en el mismo momento.

Para asignarles un valor, tendremos que utilizar el símbolo “=”.

3.2.2. Tipos de variables

Mediante los diferentes tipos de variables, estaremos haciendo referencia al tipo de datos que van a poder almacenar en su interior.

PHP admite los siguientes tipos de datos:

- **Entero**. Para números enteros, para números octales, hexadecimales o negativos:

EJEMPLO

```
$a = 1234;  
$a = 01234;  
$a = 0x1234;  
$a = -1234;
```

- **Flotante** (o **Doble**). Para números reales o números en punto flotante (double), como en los siguientes ejemplos:

EJEMPLO

```
$a = 1.234;  
$a = 1.2e3;
```

- **Cadena**. Para cadenas de caracteres. Para las cadenas de caracteres podremos utilizar dos tipos de delimitadores: comillas simples (') y comillas dobles (").

Si utilizamos las comillas dobles para encerrar una cadena, las variables que se encuentren en su interior serán expandidas. Por ejemplo, el carácter de barra invertida (\), puede ser usado para identificar caracteres especiales como \n para nueva línea o \\$ para poder representar el símbolo de dólar.

EJEMPLO

```
$pais="España";  
$ciudad='Zaragoza';
```

- **Booleano**. Para valores verdaderos o falsos.
- **Arrays**. Para almacenar conjuntos de datos todos del mismo tipo. Pueden ser unidimensionales, multidimensionales o asociativos. Los veremos en profundidad más adelante.
- **Objeto**. Para almacenar instancias de clases. Los veremos en la unidad 4.

Existen también otros tipos de variables especiales.

- Variables de tipo **NULL**, que serán aquellas variables a las que no se les ha asignado un valor, no están definidas o se les ha asignado el valor NULL.
- Variables con tipo de **recurso**, que representan recursos externos, como conexiones a bases de datos, que son devueltas por funciones o deben pasarse como parámetros a otras funciones.

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución en función del contexto en el que se utilice esa variable. Si, por ejemplo, a la variable **\$a** le asignamos un valor de cadena, se convierte en tipo cadena. Si, a continuación, le asignamos un valor entero, la habremos convertido a tipo entero.

Sin embargo, en determinadas ocasiones nos puede interesar forzar el valor concreto de una variable, por ejemplo, para validar su contenido. Los tipos de forzados que podemos realizar en PHP serán los siguientes:

- **(int), (integer)** - fuerza a entero (integer).
- **(real), (double), (float)** - fuerza a doble (double).
- **(string)** - fuerza a cadena (string).
- **(array)** - fuerza a array (array).
- **(object)** - fuerza a objeto (object).

De esta forma, podríamos realizar las siguientes conversiones:

EJEMPLO

```
<?php
$numero=10;//tipo entero
$numero2=(double) $numero;//$numero2 es tipo double
$a='hola';
$array=(array) $a;// $array será tipo array y su primer valor
será una cadena
echo $array[0];// tendremos como salida "hola"

$objeto=(object) $a; //se convierte a tipo objeto
$var1=11.1;
$var2=(int)$var1;//la convertimos a tipo entero con valor 11
echo $var2;
?>
```

Mediante la función **settype()** podremos también establecer el tipo de dato que vamos a guardar en una variable. La función necesita de dos argumentos: el primero será el nombre de la variable y el segundo, el tipo que queremos establecer. Si el proceso de conversión se realiza con éxito, devolverá el valor *true*, y de lo contrario devolverá *false*.

EJEMPLO

```
<?php
$var=8.5;
echo "$var<br>";
settype($var,"integer"); // convertimos su valor a entero
echo $var;
?>
```

3.2.3. Averiguar el tipo de una variable

Para averiguar el tipo de variables podemos utilizar la función **gettype()**. Esta función devolverá el tipo de dato pasado como parámetros. Los posibles valores a devolver serán los siguientes: *integer*, *double*, *string*, *array*, *object*, *class* y *unknown type* (tipo desconocido).

EJEMPLO

```
<?php
$edad="34";
print("Tipo actual: ".gettype($edad)."<br>");
settype($edad,"integer");
print("Tipo actual: ".gettype($edad)."<br>");
?>
```

3.2.4. Chequeando variables

Podremos también determinar el tipo de una variable pasada como argumento con las siguientes funciones en función del tipo de dato que queramos evaluar:

- **is_array()** - Comprueba si una variable es una matriz.
- **is_bool()** - Comprueba si una variable es de tipo booleano.
- **is_double()** - Comprueba si el tipo de una variable es flotante.
- **is_float()** - Comprueba si el tipo de una variable es flotante.
- **is_int()** - Comprueba si el tipo de una variable es un número entero.
- **is_integer()** - Comprueba si el tipo de una variable es un número entero.
- **is_long()** - Comprueba si el tipo de una variable es un número entero.
- **is_null()** - Comprueba si una variable es NULL.
- **is_numeric()** - Comprueba si una variable es un número o una cadena numérica.

- **is_object()** - Comprueba si una variable es un objeto.
- **is_real()** - Comprueba si el tipo de una variable es flotante.
- **is_resource()** - Comprueba si una variable es un recurso.
- **is_scalar()** - Comprueba si una variable es escalar.
- **is_string()** - Comprueba si una variable es de tipo string.

Veamos un ejemplo del uso de alguna de estas funciones:

EJEMPLO

```
<?php
$edad=34;
if(is_int($edad))echo $edad." es un entero";
else echo $edad. " no es un entero";
?>
```

3.2.5. Constantes

La principal diferencia que existe entre constantes y variables, es que las constantes van a tener un valor fijo y no va a poder ser modificado durante la ejecución de una página.

Para poder definir una constante tendremos que utilizar la instrucción “define”, que tendrá la siguiente sintaxis: **define(“nombre de constante”, “valor de constante”)**.

Para hacer uso de ella no hará falta incluir el símbolo “\$”. Los nombres de las constantes es conveniente definirlos siempre con mayúsculas para diferenciarlos de las variables. Esta convención no es obligatoria, pero ayuda a que la lectura y el mantenimiento del código sea más sencillo.

Veamos un par de ejemplos:

EJEMPLO

```
<?php
define ("CIUDAD", "Zaragoza");
define ("PAIS", "España");
echo "Vivo en ".CIUDAD."<br>";
echo "Y mi país es ".PAIS;
?>
```

Además de las constantes que nosotros mismos podemos definir, PHP incorpora una serie de constantes propias, que podremos visualizar a través de la función “**phpinfo()**”.

Algunas de estas constantes predefinidas son las siguientes:

- **__FILE__**. Nos mostrará el nombre del fichero de comandos que está siendo interpretado en ese momento.
- **__LINE__**. Nos mostrará el número de línea dentro del fichero que está siendo interpretado en ese momento.
- **PHP_VERSION**. Nos mostrará la versión del intérprete PHP.
- **PHP_OS**. Nos mostrará la versión del sistema operativo en la que está siendo ejecutado el intérprete PHP.
- **TRUE**. Valor verdadero.
- **FALSE**. Valor falso.
- **E_ERROR**. Denota un error distinto de un error de interpretación del cual no es posible recuperarse.
- **E_WARNING**. Denota una condición donde PHP reconoce que hay algo erróneo, pero continuará de todas formas.
- **E_PARSE**. El intérprete encontró una sintaxis no válida en el fichero de comandos.
- **E_NOTICE**. Ocurrió algo que pudo ser o no un error, pero la ejecución continúa.

EJEMPLO

```
<?php
echo __LINE__.".- VERSION de PHP:".PHP_VERSION;
echo "<br>";
echo __LINE__.".- SISTEMA Operativo del Servidor: ".PHP_OS;
?>
```

3.2.6. Variables por referencia

En PHP también podremos realizar asignaciones de valores por referencia. En esta forma de trabajo, una variable no recibirá un valor, sino que le será asignada otra variable, de tal forma que las dos variables van a compartir el mismo espacio en memoria para almacenar el mismo dato.

Para representar las asignaciones por referencia se utilizará el símbolo "&" por delante del nombre de la variable. Veamos un ejemplo de funcionamiento:

EJEMPLO

```
<?php
$var1 = 'Silvia'; // Asigna el valor 'Silvia' a $var1
$var2 = &$var1; // Referencia $var1 vía $var2.
$var2 = "Mi nombre es $var2 <br>"; // Modifica $var2...
echo $var1; // $var1 también se modifica.
echo $var2;
?>
```

3.2.7. Variables de variables

PHP nos suministra también otro tipo de variable denominada variable de variables. Con las variables de este tipo podremos cambiar el nombre de una variable de forma dinámica.

De esta forma, podremos utilizar el valor de una variable como nombre para la otra. Veamos un ejemplo:

EJEMPLO

```
<?php
$var="ciudad";
$$var="Zaragoza";
echo $ciudad."<br>"; //el resultado es Zaragoza
echo $$var; //el resultado es Zaragoza
?>
```

En la primera línea establecemos el valor “ciudad” para la variable **\$var**. En la segunda línea usaremos el valor de la primera línea para crear una segunda variable **\$\$var**, que además recibirá el valor de “Zaragoza”.

3.2.8. Variables del sistema

PHP será capaz de facilitarnos el acceso a una información recogida en unas variables especiales, denominadas variables del sistema, que nos darán, por ejemplo, información sobre el servidor, el cliente, etc.

Estos valores nunca podrán ser modificados, puesto que su información está atribuida al servidor. Incluso puede que algunas de estas variables no funcionen en todos los servidores, y otras necesitarán de su activación en el servidor.

Veamos un listado con las principales:

- **\$HTTP_USER_AGENT**. Información sobre el sistema operativo y tipo y versión de navegador utilizado por el cliente.
- **\$HTTP_ACCEPT_LANGUAGE**. Nos devolverá la o las abreviaciones del idioma del navegador.

- **\$HTTP_REFERER**. Nos va a suministrar la URL desde la cual el visitante ha tenido acceso a la página.
- **\$PHP_SELF**. Nos va a suministrar una cadena con la URL del script que está siendo ejecutado en ese momento.
- **\$HTTP_GET_VARS**. Esta variable es un array que va a guardar los nombres y los contenidos de las variables enviadas al script por URL o por formularios utilizando el modo GET.
- **\$HTTP_POST_VARS**. Esta variable es un array que va a guardar los nombres y contenidos de las variables enviadas al script por medio de un formulario utilizando el modo POST.
- **\$HTTP_COOKIES_VARS**. Esta variable es un array que va a guardar los nombres y contenidos de las cookies.
- **\$PHP_AUTH_USER**. Esta variable va a guardar la variable usuario cuando visitamos páginas de acceso restringido. Combinado con **\$PHP_AUTH_PW** resulta ideal para controlar el acceso a las páginas internas de nuestra web.
- **\$PHP_AUTH_PW**. Esta variable va a guardar la variable password cuando visitemos páginas de acceso restringido. Combinado con **\$PHP_AUTH_USER** resulta ideal para controlar el acceso a las páginas internas del sitio.
- **\$REMOTE_ADDR**. Esta variable nos facilitará la dirección IP del visitante.
- **\$DOCUMENT_ROOT**. Nos devolverá la ruta en el servidor donde se encuentra alojada la página web.
- **\$PHPSESSID**. Guarda el identificador de sesión del usuario.

3.2.9. Variables superglobales

Algunas variables predefinidas en PHP son “superglobales”, lo que significa que están disponibles en todos los ámbitos a lo largo del script. Por lo tanto, no será necesario definir la variable como “*global \$variable*” para acceder a ellas dentro de las funciones o métodos.

Las variables superglobales son:

- **\$GLOBALS**. Va a almacenar referencias a las variables que se estén utilizando en el ámbito global del script.
- **\$_SERVER**. Mediante el uso de este array podremos obtener información que es creada por el servidor web. No es seguro que funcione en todos los servidores, puesto que dependerá de la configuración de cada uno de ellos.

Con esta variable superglobal podremos utilizar como índices las variables explicadas anteriormente como variables de sistema:

EJEMPLO

```
<?php
echo "Tu dirección IP es: " . $_SERVER['REMOTE_ADDR'] . " <br>";
echo "Tu directorio de proyectos es : " . $_SERVER['DOCUMENT_
ROOT'] . " <br>";
?>
```

- **\$_GET**. Array asociativo que va a almacenar las variables que se han suministrado al script como parámetros de una URL o mediante el método GET.
- **\$_POST**. Array asociativo que va a almacenar las variables que se han suministrado al script mediante el método POST.
- **\$_FILES**. Array asociativo que va a almacenar las variables donde podremos obtener información de los archivos que han sido subidos mediante formularios.
- **\$_COOKIE**. Array asociativo que va a almacenar las variables que se han suministrado al script mediante el método Cookies HTTP.
- **\$_SESSION**. Array asociativo que va a almacenar las variables que están disponibles para el script.
- **\$_REQUEST**. Array asociativo que va a almacenar por defecto el contenido de \$_GET, \$_POST y \$_COOKIE.
- **\$_ENV**. Array asociativo que va a almacenar las variables que se han suministrado al script mediante el método del entorno.

3.2.10. Ámbito de las variables

Con el ámbito de una variable nos estamos refiriendo al lugar dentro de una secuencia de comandos en los que resultará visible dicha variable.

Para concretar estos ámbitos, hay que tener en cuenta las siguientes reglas:

- Las variables superglobales serán siempre visibles dentro de una secuencia de comandos.
- Las constantes que hayan sido declaradas son visibles de forma global y podrán ser utilizadas tanto fuera como dentro de una función.
- Las variables globales que hayan sido declaradas en una secuencia de comandos serán visibles en dicha secuencia, pero no en el interior de una función.
- Las variables que sean utilizadas dentro de una función y que han sido declaradas como globales, harán referencia a la variable global que tiene el mismo nombre.
- Las variables que hayan sido creadas dentro de una función y que hayan sido declaradas como estáticas serán invisibles desde el exterior de la función.

- Las variables que sean creadas en el interior de una función tendrán restringido su ámbito a la propia función y dejarán de existir al finalizar la función.

Existen unas variables denominadas **superglobales**, que tienen sus propias reglas de ámbito, pero las explicaremos en su apartado.

Veamos unos cuantos ejemplos para comprender los ámbitos.

EJEMPLO

```
<?php
$a = 1; /* ámbito global */
Function prueba () {
    echo $a; /* referencia a una variable de ámbito local */
}
Test ();
?>
```

Si ejecutamos este script, nos dará un error, puesto que dentro de la función, estamos intentando mostrar el contenido de una variable que no ha recibido ningún valor dentro de su ámbito local.

En el siguiente ejemplo, sí que tendremos como salida el valor “3”, ya que dentro de la función estamos declarando las variables a nivel global, y por lo tanto, los cambios se realizan también a nivel global.

EJEMPLO

```
<?php
$var1 = 1;
$var2 = 2;
Function suma () {
    global $var1, $var2;
    $var2 = $var1 + $var2;
}
Suma ();
echo $var2;
?>
```

Otra posibilidad que tenemos es declarar una variable como “static” dentro de una función, y mantendrá siempre su valor dentro de su ámbito. Por ejemplo, si quisiéramos utilizar una función para realizar una cuenta atrás, donde por cada llamada a la función se disminuyera en 1 el valor de la variable estática, tendríamos que realizarlo de la siguiente forma:

EJEMPLO

```
<?php
Function Cuentaatras () {
static $a = 10;
echo $a."<br>";
$a--;
}
Cuentaatras();
Cuentaatras();
?>
```

3.3. Operadores

Los operadores son símbolos que podrán ser utilizados para manipular valores y variables de tal forma que podamos realizar operaciones con ellos.

Hasta ahora hemos utilizado operadores como “=” para realizar asignaciones y el operador “.” para concatenar texto y variables en las salidas del comando “echo”. A continuación, vamos a explorar todas las posibilidades de estos operadores.

3.3.1. Operadores aritméticos

Son los operadores matemáticos de uso más común:

- “+” para la suma. Su sintaxis sería: \$a + \$b.
- “-” para la resta. Su sintaxis sería: \$a - \$b.
- “*” para la multiplicación. Su sintaxis sería: \$a * \$b.
- “/” para la división. Su sintaxis sería: \$a / \$b.
- “%” para el módulo. Devuelve el resto de la división. Su sintaxis sería: \$a % \$b.

3.3.2. Operadores de concatenación

En PHP encontramos un operador cuya misión va a ser la de encadenar variables de tipo cadena o string, o incluso valores literales utilizados para mostrar mensajes concretos. De esta forma, si disponemos de unas cuantas variables de este tipo, las podremos concatenar con el símbolo “.”:

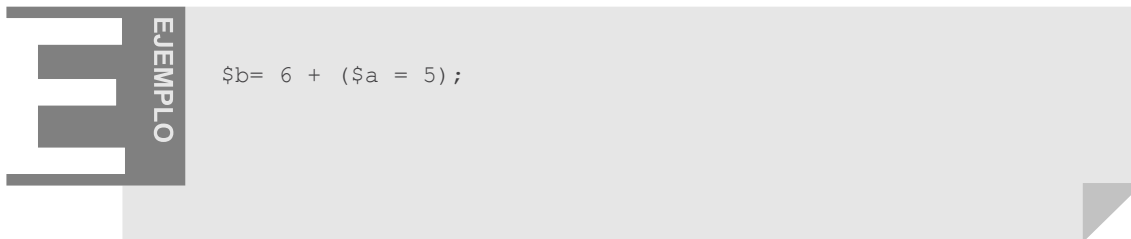
EJEMPLO

```
<?php
$a = "Yo trabajo en ";
$b = $a . "mi casa"; // ahora $b contiene "Yo trabajo en mi casa"
echo $b;
$a = "Yo trabajo en ";
$a .= "mi casa"; // ahora $a contiene "Yo trabajo en mi casa"
echo $a;
?>
```

3.3.3. Operadores de asignación

Como ya hemos comentado, ya hemos utilizado el operador básico de asignación “=”.

Se puede utilizar en expresiones algo más complejas con operaciones como la siguiente:



Al utilizar los paréntesis, damos la prioridad a que primero la variable \$a se le asigne un valor, que luego será utilizado para que finalmente la variable \$b le sea asignado el valor 11.

Además de este operador básico, podremos utilizar los denominados “**operadores combinados**” para poder realizar operaciones aritméticas o de manipulación de cadenas que sean binarias. De esta forma, podremos realizar en una sola instrucción el uso de una variable en una expresión para luego establecer un nuevo valor a dicha variable como resultado de la expresión.

Podremos utilizar los siguientes:

Operador	Uso	Equivalencia
+=	<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>
- =	<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>
*=	<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>
/=	<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>
%=	<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>
.=	<code>\$a .= \$b</code>	<code>\$a = \$a . \$b</code>

Figura 3.66. Operadores combinados.

Veamos un ejemplo:

EJEMPLO

```
<?php
$a = 2;
$a += 4; // establece $a a 6, al igual que $a = $a + 4;
echo $a;
$a = 4;
$a /= 2; // establece $a a 2, al igual que $a = $a / 2;
echo $a;
?>
```

3.3.4. Operadores de incremento y decremento

Son una variante de los operadores matemáticos que nos permitirán aumentar o disminuir en 1 el valor de una variable:

- “++” para aumentar el valor de la variable en 1.
- “--” para disminuir el valor de la variable en 1.

Si ponemos estos operadores por delante de la variable estaremos incrementando o decrementando en uno el valor de la variable antes de utilizarla.

Si los ponemos a continuación de la variable, primero usaremos la variable y a continuación, aumentaremos o decrementamos su valor en uno.

Veamos un ejemplo:

EJEMPLO

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

echo ++$a . "<br>\n";
echo $a . "<br>\n";
echo "<h3>Postdecremento</h3>";
$a = 5;
echo $a-- . "<br>\n";
echo $a . "<br>\n";
echo "<h3>Predecremento</h3>";
$a = 5;
echo --$a . "<br>\n";
echo $a . "<br>\n";
?>
```

3.3.5. Operadores condicionales

Los operadores condicionales los utilizaremos para comprobar el resultado de una determinada operación. Este resultado, podrá ser verdadero (**True**) o falso (**False**). Para comprender mejor estos operadores vamos a suponer que tenemos la variable \$x=7 y la variable \$z=5. Podremos utilizar los siguientes operadores:

Operador	Operación	Sintaxis	Resultado
==	Igual (\$x y \$z tienen el mismo valor)	\$x==\$z	False
===	Idéntico (\$x y \$z tienen el mismo valor y además son del mismo tipo)	\$x=== \$z	False
!=	Diferente (\$x y \$z son de diferente valor)	\$x!=\$z	True
<	Menor (\$x menor que \$z)	\$x<\$z	False
>	Mayor (\$x mayor que \$z)	\$x>\$z	True
<=	Menor o igual (\$x menor o igual que \$z)	\$x<=\$z	False
>=	Mayor o igual (\$x mayor o igual que \$z)	\$x>=\$z	True

Figura 3.67. Operadores condicionales.

3.3.6. Operadores de ejecución

Con PHP podremos hacer uso de un operador de ejecución mediante el cual intentará ejecutar la instrucción que se encuentre en el interior de los operadores de ejecución como si fuera un comando. Estos operadores se representan mediante el apóstrofe invertido “`<`”. Veamos un ejemplo:

EJEMPLO

```

<?php
$var = `ls -al`; // para sistemas Unix
echo "<pre>".$var."</pre>";
$var = `dir c:`; // para sistemas Windows
echo '<pre>' . $var . '</pre>';
?>

```

3.3.7. Operadores lógicos

Los operadores lógicos son usados para combinar los resultados de condiciones lógicas.

Operador	Sintaxis	Resultado
&&	<code>\$a && \$b</code>	True (si <code>\$a</code> y <code>\$b</code> son verdaderos)
AND	<code>\$a AND \$b</code>	Igual que && pero con prioridad más baja
	<code>\$a \$b</code>	True (si <code>\$a</code> o <code>\$b</code> son verdaderos)
OR	<code>\$a OR \$b</code>	Igual que pero con prioridad más baja
XOR	<code>\$a XOR \$b</code>	Es matemáticamente igual a (<code>a AND (NOT b)</code>) OR ((<code>NOT a</code>) and <code>b</code>).
!	<code>!\$a</code>	True (si <code>\$a</code> no es verdadero)

Figura 3.68. Operadores lógicos.

Por ejemplo, si quisiéramos comprobar que el valor de la variable `$a` se encuentre entre los valores 0 y 50, tendremos que utilizar una estructura condicional que veremos más adelante donde utilizaríamos la expresión (`$a > 0 AND $a < 50`).

3.4. Estructuras de control

Mediante la utilización de estructuras de control, podremos ejecutar instrucciones variando el orden secuencial inicial de las mismas en función de unas condiciones y unos valores que determinarán la alteración del flujo de un programa.

3.4.1. IF

Mediante la instrucción IF podremos ejecutar una serie de instrucciones que vendrán a continuación, si se cumple la condición que lo acompaña.

Esta es la forma más sencilla de estructura de control. Con un IF sencillo evaluaremos una condición que se encontrará obligatoriamente encerrada entre paréntesis, de tal forma que si se cumple la condición, se ejecuta el código que sigue a continuación y si no, es ignorado. Veamos un ejemplo:

EJEMPLO

```
<?php
$a = 44;
$b = 77;
if ($a <= $b) {
    echo "Esta parte se ejecuta";
}
?>
```

La secuencia de acciones que estén encerradas dentro de las estructuras de control tendrán que empezar y terminar con unas llaves “{}”.

Podremos anidar tantas sentencias IF en el interior como consideremos necesario.

3.4.2. IF ... ELSE

En la anterior estructura no contemplamos la posibilidad de que la condición pueda no cumplirse. Podemos añadir a la estructura anterior una nueva posibilidad de ejecutar el código en el caso de que no se cumpla la condición inicial.

Veamos un ejemplo cambiando el operador “<=” por el operador “>=”:

EJEMPLO

```
<?php
$a = 44;
$b = 77;
if ($a >= $b) {
    echo "Esta parte no se ejecuta";
}
else {
    echo "Esta parte se ejecuta";
}
?>
```

3.4.3. ELSEIF

Mediante esta estructura, que es una combinación de IF y ELSE, podremos extender las posibilidades de las estructuras condicionales para poder ejecutar unas instrucciones diferentes, en el caso de que la condición de IF sea falsa. Pero a diferencia de ELSE, las instrucciones que se encuentran dentro del bloque ELSEIF solo se ejecutarán si su condición es TRUE. Veámoslo con un ejemplo:

EJEMPLO

```
<?php
$a = 44;
$b = 77;
if ($a > $b) {
    echo "a es mayor que b";
}
elseif ($a == $b){
    echo "a es igual que b";
}
else {
    echo "a es menor que b";
}
?>
```

Podremos anidar tantos ELSEIF dentro de una sentencia IF como consideremos oportuno.

La sentencia ELSEIF se ejecuta solo si la expresión IF precedente y cualquier expresión ELSEIF precedente se evalúan como FALSE, y la expresión ELSEIF actual se evalúa como TRUE.

3.4.4. Sintaxis alternativa de estructura de control

Como ya hemos explicado, en las estructuras condicionales anteriores, las secuencias de instrucciones a ejecutar dentro de una condición, tenían que estar encerradas entre llaves “{ }”.

PHP ofrece una alternativa y se trata de sustituir la llave inicial “{” por los dos puntos “:”, y la llave final “}” por la palabra reservada “**endif**”.

Veamos el mismo ejemplo anterior, pero con estas alternativas:

EJEMPLO

```
<?php
$a = 44;
$b = 77;
if ($a > $b):
echo "a es mayor que b";
elseif ($a == $b):
echo "a es igual que b";
else :
echo "a es menor que b";
endif;
?>
```

PHP ofrece, además, otra sintaxis alternativa que puede ser utilizada.

Si escribiéramos el siguiente código:

EJEMPLO

```
<?php
$var=5;
($var==5)?(echo"$var es igual a 5"):(echo "$var es diferente
de 5);
php?>
```

Sería equivalente a escribir lo siguiente:

EJEMPLO

```
<?php
If($var==5)
{
    Echo "$var es igual a 5";
}
Else
{
    Echo "$var es diferente a 5";
}
php?>
```

3.4.5. SWITCH

La instrucción SWITCH es parecida a una instrucción IF en cuanto a que evalúa condiciones, pero SWITCH nos va a permitir evaluar varias posibilidades de condiciones de una manera más clara y sencilla que anidar varios IF..ELSE IF.. ELSE.

En las instrucciones **IF** se evalúa constantemente si un valor es verdadero y sino si es falso, sin embargo, la instrucción **SWITCH** se utiliza para un dato que podrá tener múltiples valores.

Habrà que incluir una instrucción **CASE** para cada uno de los valores que se quiera evaluar, y de manera opcional, un último valor que actúe como predeterminado en el caso de que no se cumpla la condición de las demás.

Solo podrán evaluarse valores simples como números, punto flotante o cadenas de texto. No podrán utilizarse arrays ni objetos.

Para cada uno de los valores con los que será comparada la variable, se ejecutarán un conjunto de instrucciones. Este conjunto deberá concluir con la instrucción "**break**". Esta instrucción, que la explicaremos en capítulos posteriores, básicamente forzarà el salto hacia el final de CASE para permitirnos seguir con la ejecución secuencial del script.

Veamos un ejemplo:

EJEMPLO

```
<?php
$ciudad="Zaragoza";
switch ($ciudad) {
case "Zaragoza":
echo "La comunidad es Aragón";
break;
case "Barcelona":
echo "La comunidad es Cataluña";
break;
case "Castellón":
echo "La comunidad es Valencia";
break;
default:
echo "En alguna parte de España";
}
?>
```

Aquí también podrá ser utilizada la sintaxis alternativa para las estructuras de control, de tal forma que tendría la siguiente sintaxis:

EJEMPLO

```
<?php
$ciudad="Zaragoza";
switch ($ciudad):
case "Zaragoza":
echo "La comunidad es Aragón";
break;
case "Barcelona":
echo "La comunidad es Cataluña";
break;
case "Castellón":
echo "La comunidad es Valencia";
break;
default:
echo "En alguna parte de España";
endswitch;
?>
```


3.4.6. WHILE

El uso del bucle **WHILE** es el tipo más sencillo de utilizar PHP, recordando en parte a una sentencia **IF** en su sintaxis:

EJEMPLO

```
WHILE (condición) {  
    //instrucciones a ejecutar mientras se cumpla la condición  
}
```

La condición que se comprueba devolverá un resultado de tipo booleano, de tal forma que mientras la condición sea verdadera, la ejecución seguirá entrando en el bucle a ejecutar las sentencias que forman parte de él.

La característica que lo diferencia de otros bucles es que la condición es evaluada antes de entrar en el bucle, por lo que podrá darse el caso de que ni siquiera se ejecute una sola vez. Veamos un ejemplo con un contador de 1 hasta 10:

EJEMPLO

```
<?php  
$i = 1;  
while ($i <= 10) {  
    echo $i++;  
}  
?>
```

Como podemos observar, al llegar al **WHILE** la variable tiene valor 1, y por lo tanto, cumple la condición, por lo que entra al bucle y al salir habrá aumentado su valor en 1. Al llegar al valor 11 la condición se volverá **FALSA**, por lo que habrá terminado la ejecución del bucle.

Obviamente será necesario que en el interior del bucle se produzca una variación en la variable que es evaluada, puesto que si no, nos estaríamos encontrando ante un bucle infinito.

Como con la sentencia **IF**, se pueden agrupar múltiples sentencias dentro del mismo bucle **WHILE**, encerrando un grupo de sentencias con llaves o usando la sintaxis alternativa: **while (expr): sentencia ... endwhile;**

En determinadas ocasiones, nos puede interesar que el valor de esa variable sea modificado en la propia condición. Esto podrá ser posible gracias a que el valor 0 es considerado como **FALSO**.

Veamos un ejemplo con un contador de cuenta atrás:

EJEMPLO

```
<?php
$i = 10;
while (--$i) :
    echo $i;
endwhile;
?>
```

En este ejemplo, antes de comprobarse la condición del WHILE, el valor de `$i` es decrementado, y por lo tanto, su valor será 9. Cuando su valor sea 0, será considerado como falso y no se ejecutará el bucle de nuevo.

3.4.7. DO ... WHILE

Esta sentencia es muy parecida al bucle WHILE, pero con la diferencia de que aquí se entrará dentro del bucle, al menos, en una ocasión, ya que la condición se evalúa a la salida del bucle. Veamos un ejemplo en el que, por lo menos, nos aseguramos la entrada una vez en el bucle:

EJEMPLO

```
<?php
$i = 15;
do {
    echo $i."<br>";
    $i--; //sólo se ejecuta una vez porque no se cumple la condición.
} while ($i < 10)
?>
```

En un bucle DO..WHILE también podremos aprovecharnos en ocasiones de que el valor 0 sea considerado como FALSO. Veamos un ejemplo de esto con un contador hasta 0:

EJEMPLO

```
<?php
$i = 10;
do {
    echo $i."<br>";
    $i--;
} while ($i)
?>
```

Como resultado en cada línea nos mostrará el valor de la variable desde 10 hasta 1, pero nunca visualizará el 0, ya que será la condición de salida del bucle, y por lo tanto, ya no volverá a entrar en él.

3.4.8. FOR

El bucle FOR realizará la repetición de un conjunto de instrucciones un número determinado de ocasiones. Tiene tres fases:

- En la primera se inicializa la variable que se va a utilizar para evaluar la condición.
- En la segunda se establece la condición que finalizará el bucle.
- En la tercera se modificará el valor de la variable de la condición.

Veamos un ejemplo de nuevo con un contador de 1 a 10, ambos inclusive:

EJEMPLO

```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
?>
```

PHP también soporta la sintaxis de dos puntos alternativa para bucles FOR. Tendría la siguiente sintaxis:

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

3.4.9. FOREACH

La sentencia FOREACH nos va a ser de ayuda para recorrer los valores de un array, lo cual nos puede resultar útil, por ejemplo, para efectuar una lectura rápida del mismo. Su estructura es la siguiente:

EJEMPLO

```
Foreach ($array as $clave=>$valor)
{
    instruccion1;
    instruccion2;
    ...;
}
```

Veamos un ejemplo donde primero inicializamos el array con una serie de valores y luego lo recorremos mostrando el contenido de todos sus valores:

EJEMPLO

```
<?php
$a = array(3, 4, 8, 17, 20);
foreach($a as $c) {
    print "Valor actual de \$a: $c.\n";
}
?>
```

3.4.10. BREAK

Podremos utilizar la sentencia BREAK para escapar de estructuras de control como FOR, WHILE o SWITCH.

Como opción avanzada, BREAK admite un parámetro opcional mediante el cual podremos indicarle de cuantas estructuras queremos escapar.

Veámoslo con un ejemplo:

EJEMPLO

```
<?php
$i = 0;
while (++$i)
{
    switch ($i)
    {
        case 5:
            echo "Con el valor 5 salimos del switch<br>\n";
            break 1; /* Salimos del switch */
        case 10:
            echo "Con el valor 10 salimos del while<br>\n";
            break 2; /* Salimos del while */
        default:
            echo $i."<br>";
            break;
    }
}
?>
```

3.4.11. CONTINUE

Podremos utilizar la sentencia CONTINUE para saltarnos el resto de las sentencias de la iteración en la que nos encontremos, y volver a continuar con la ejecución al comienzo de la siguiente iteración.

Veámoslo con un ejemplo en el que queremos mostrar solo aquellos números que sean múltiplos de 3:

EJEMPLO

```
<?PHP
$i = 0;
while ($i++ < 17) {
    #condición de no múltiplo de 3
    if ($i % 3 !=0){
        continue ;
    }
    echo "El valor de i es: ".$i."<br>";
}
?>
```

En el ejemplo anterior, cada vez que se entra en el bucle, mediante la estructura IF vamos a comprobar si el número es múltiplo de 3. Si NO es múltiplo, se ejecuta la instrucción “continue”, que nos permitirá no visualizar dicho número, terminando esta iteración y volviendo de nuevo al WHILE. De esta forma, solo serán visualizados aquellos números que sí sean múltiplos de 3.

3.5. Arrays

En capítulos anteriores, ya hemos introducido los conceptos de variables y tímidamente el de array como un tipo de variable.

Hemos visto que una variable la podemos identificar con un nombre y que va a almacenar algún tipo de dato en concreto, pero solo un valor.

Sin embargo, con un array, podremos almacenar muchos valores y de diferentes tipos si lo estimamos oportuno. Y todos identificados con el mismo nombre.

De esta forma, un array tendrá tres elementos principales:

- Los elementos que formarán el array.
- El índice, que va a identificar el lugar que va a ocupar cada elemento dentro del array.
- El valor que va a contener cada elemento.

Una consideración muy a tener en cuenta, es el primer elemento de un array, estará localizado en la posición 0, y no en la 1 como podríamos pensar.

3.5.1. Crear un array

Podremos crear un array de diferentes maneras. Vamos a explicarlo con ejemplos que se verá más claro:

EJEMPLO

```
$nombres[]="Paco";
$nombres[]="Silvia";
$nombres[]="Cristian";
```

En este primer ejemplo, creamos un array para almacenar nombres. Como no especificamos los índices, estos valores se irán almacenando en el array con las posiciones consecutivas empezando por el 0: 0,1,2,3...

En el siguiente ejemplo, vamos a indicar de forma expresa el valor del índice, y por lo tanto, el lugar donde queremos que dichos valores se almacenen en el array, de tal forma que no tienen por qué tener un valor secuencial:

EJEMPLO

```
$nombres[2]="Paco";
$nombres[5]="Silvia";
$nombres[10]="Cristian";
```

En este array, solo tenemos almacenados tres elementos, aunque el valor de los índices nos pueda causar alguna confusión.

Podremos hacer uso de la función **count()** para conocer el número de elementos que contiene dicho array. Su sintaxis es la siguiente:

EJEMPLO

```
int count(string array)
```

Esto significa que le tendremos que pasar como argumento el nombre del array, y la función nos devolverá un valor entero, que será el número de elementos del array.

Si añadimos un nuevo valor de una forma no secuencial al array anterior, la posición que ocupará será el siguiente índice a la última posición secuencial. Veamos un ejemplo:

EJEMPLO

```
<?PHP
$nombrres=array("Paco","Silvia","Cristian","Victor");
echo $nombrres[2];
?>
```

De esta forma, si posteriormente quisiéramos modificar el valor de un índice, tendríamos que escribir el valor del nuevo índice y, a continuación, el símbolo "=", junto con el valor que modificará su índice:

EJEMPLO

```
<?PHP
$nombrres=array("Paco",3=>"Silvia","Cristian","Victor");
echo $nombrres[4];
?>
```

3.5.2. Arrays asociativos

Hasta ahora hemos creado arrays mediante las asignaciones índice=>valor, donde los índices han sido numéricos. Pero también podemos utilizar cadenas de texto como índices. A éstos se les llama **arrays asociativos**. Veamos un ejemplo:

EJEMPLO

```
<?PHP
$notas = array(
    "lengua">8,
    "matematicas">6,
    "programacion">3);
echo $notas['programacion'];
?>
```

También podremos utilizar la versión clásica sin tener que utilizar el constructor “array”:

EJEMPLO

```
<?PHP
$notas["lengua"]=8;
$notas["matematicas"]=5;
$notas["programacion"]=4;
echo $notas['programacion'];
?>
```

3.5.3. Arrays multidimensionales

Al principio de este capítulo, hemos comentado que los arrays pueden almacenar todo tipo de valores. Eso incluye también que pueden almacenar otro array. De esta manera, estaremos creando un array de varias dimensiones o arrays multidimensionales.

Veamos cómo podemos crear un array asociativo multidimensional:

EJEMPLO

```
<?php
$flores = array( array( "Titulo" => "rosa",
                      "Precio" => 2,
                      "Numero" => 13
                    ),
                array( "Titulo" => "gardenia",
                      "Precio" => 1,5,
                      "Numero" => 10
                    ),
                array( "Titulo" => "clavel",
                      "Precio" => 0,5,
                      "Numero" => 25
                    )
            );
echo $flores [0] ["Titulo"];
echo "<br>";
echo $flores [1] ["Titulo"];
echo "<br>";
echo $flores [2] ["Titulo"];
?>
```

3.5.4. Uso de arrays en bases de datos

En la unidad 5 aprenderemos a cómo trabajar con bases de datos MySQL mediante instrucciones en lenguaje PHP. Como veremos entonces, el uso de arrays para almacenar resultados de consultas, por ejemplo, será fundamental.

Vamos a ver un pequeño ejemplo de cómo podremos utilizar los arrays en contexto de bases de datos, aunque aún no sepamos muy bien cómo funcionan las funciones de bases de datos que veremos a continuación:

EJEMPLO

```
<?php
$consulta ="SELECT nombre, apellidos FROM alumnos";
$query=mysql_query($consulta,$connect);
$fila= mysql_fetch_array($query);
echo $fila["nombre"];
echo $fila[1]; //mostramos el campo apellidos
?>
```

Ahora vamos a analizarlo:

Mediante la instrucción `mysql_query()` podremos realizar una consulta a la base de datos a la que nos hayamos conectado mediante la variable de conexión `$connect`. La consulta que enviamos la hemos guardado previamente en la variable `$consulta`.

El resultado de esa consulta es almacenado en la variable **\$query**, y para extraer su contenido utilizaremos la instrucción **mysql_fetch_array()**.

En este ejemplo los datos devueltos por la consulta han sido almacenados en el array **\$fila**. Para poder acceder a ellos, podremos utilizar como índice el nombre del campo que lo identifica, o bien su posición en la selección.

Esto es solo un ejemplo del uso de arrays con operaciones de bases de datos, pero hay que tenerlo muy en cuenta porque en la unidad 5 haremos ejemplos parecidos de forma constante.

3.5.5. Recuperar el contenido de un array

Para recorrer los arrays de tal forma que podamos extraer su contenido, nos podremos ayudar de bucles ya conocidos como **FOR**, **FOREACH** o **WHILE**, y también de instrucciones como **list()** y **each()**.

Para empezar, veamos dos formas de utilizar el bucle **FOREACH()**:

EJEMPLO

```

<?php
$flores = array( array( "Titulo" => "rosa",
                        "Precio" => 2,
                        "Numero" => 13
                      ),
                  array( "Titulo" => "gardenia",
                        "Precio" => 1,5,
                        "Numero" => 10
                      ),
                  array( "Titulo" => "clavel",
                        "Precio" => 0,5,
                        "Numero" => 25
                      )
                );

foreach($flores as $item)
{
    echo $item['Titulo'];
    echo " ";
    echo $item['Precio'];
    echo " ";
    echo $item['Numero'];
    echo "<br>";
}
?>

```

Otra forma de utilizar el bucle **FOREACH** sería la siguiente:

EJEMPLO

```
<?php
foreach($flores as $item)
{
    foreach($item as $indice => $valor)
    {
        echo $indice; // Indice de la variable
        echo " ";
        echo $valor; // Su valor
        echo "<br>";
    }
}
?>
```

Ahora vamos a probar a recorrer nuestro array multidimensional con el uso del bucle **while()** y la instrucción **each()**.

La instrucción **each()** devuelve el par clave/valor actual para el array multidimensional y avanza el cursor de la misma. Esta pareja se devuelve en una array multidimensional de cuatro elementos, con las claves **0**, **1**, **key**, y **value**. Los elementos **0** y **key** contienen el nombre de clave del elemento de la matriz, y **1** y **value** contienen los datos.

Si el puntero interno para la matriz apunta pasado el final del contenido de la matriz, **each()** devuelve FALSE.

Veamos un ejemplo, en este caso empleando un array de una sola dimensión:

EJEMPLO

```
<?PHP
$flores=array("rosa","gardenia","clavel");
while($item = each($flores))
{
    echo $item['key']; // Indice de la variable
    echo "-";
    echo $item['value']; // Su valor
    echo "<br>";
}
?>
```

La instrucción **each()** para cada elemento del array, extrae por un lado el valor del índice que está recorriendo en ese momento, junto con el valor que se está almacenando.

Pero tenemos una forma más elegante de realizar esta tarea, y será mediante la instrucción **list()**, que almacenar los dos valores que proporciona la función **each()** en dos variables independientes:

EJEMPLO

```
<?php
$flores = array( array( "Titulo" => "rosa",
                        "Precio" => 2,
                        "Numero" => 13
                      ),
                array( "Titulo" => "gardenia",
                        "Precio" => 1.5,
                        "Numero" => 10
                      ),
                array( "Titulo" => "clavel",
                        "Precio" => 0.5,
                        "Numero" => 25
                      )
              );
while (list($clave, $valor ) = each($flores))
{
    echo "Posición del array multidimensional: ".$clave." ";
    echo $valor['Titulo'];
    echo "-";
    echo $valor['Precio'];
    echo "-";
    echo $valor['Numero'];
    echo "<br>";
}
?>
```

Por último, podríamos utilizar los bucles **FOR** y **WHILE** para recorrer el mismo array multidimensional. En este caso, nos apoyaremos en la función **count()** que hemos explicado previamente, y que le va a indicar al bucle **FOR** las veces que va a tener que iterar hasta llegar al final de la primera dimensión del array multidimensional:

EJEMPLO

```
<?php
$flores = array( array( "Titulo" => "rosa",
                      "Precio" => 2,
                      "Numero" => 13
                    ),
                array( "Titulo" => "gardenia",
                      "Precio" => 1.5,
                      "Numero" => 10
                    ),
                array( "Titulo" => "clavel",
                      "Precio" => 0.5,
                      "Numero" => 25
                    )
              );

for ($i = 0; $i < count($flores); $i++) {
    while ($elemento = each($flores[$i])) {
        echo $elemento[0]."=";
        echo $elemento[1]."<br>\n";
    }
    echo "<br>\n";
}
?>
```

3.6. Funciones

Las funciones son bloques de código que podremos utilizar en cualquier momento a lo largo de todo el código que estemos desarrollando.

Con la utilización de funciones, el objetivo fundamental es el ahorro de código desarrollando funciones para procesos que se puedan repetir en varias ocasiones, evitándonos tener que escribirlos cada vez que vayamos a necesitarlos.

PHP ofrece una gran variedad de funciones predefinidas, que pueden ser consultadas en el manual oficial on line: <http://es.php.net/manual/es/funcref.php>

3.6.1. Creación de funciones

Podremos crear funciones mediante la sentencia FUNCTION. Su sintaxis sería la siguiente:

EJEMPLO

```
FUNCTION nombre_funcion($variable)
{
    código
}
```

Para poder desarrollar un código más limpio y ordenado, siempre será recomendable situar la definición de las funciones en la parte superior de cualquier script.

3.6.2. Llamadas a funciones

Para llamar a una función bastará con escribir su nombre y, entre paréntesis, los argumentos que van a ser necesitados. Si no necesitáramos ningún argumento, bastaría con escribir la función solo con los paréntesis.

EJEMPLO

```
suma(); // llamada a función sin argumentos
suma(5); // llamada a función con un argumento
suma(13,8,6); // llamada a función con tres argumentos
```

Una vez que la función ha recibido los valores de los argumentos y han sido procesados, la función devolverá un resultado.

Este resultado podremos aprovecharlo para otras operaciones, para otras funciones o simplemente para mostrarlo en pantalla. También podremos desarrollar funciones cuyo objetivo no sea el de devolver un resultado, puesto que lo generarán en su interior.


El código que incorpore la función será leído por el intérprete PHP al cargar el script, pero no será ejecutado hasta que encuentre en el código un llamamiento a la función.

Una combinación habitual es la de definición de funciones que utilizamos de forma frecuente en otros ficheros, como por ejemplo, operaciones con bases de datos, y luego utilizarlas mediante las funciones `include()` y `require()` que explicaremos en el siguiente capítulo.

3.6.3. Parámetros de las funciones

Si declaramos una función mediante la sentencia ***FUNCTION nombre_funcion(\$variable)***, a ***\$variable*** se le denomina **argumento**. Podremos pasar tantos argumentos a las funciones como sea necesario, separándolos con comas. Siempre tendrán que coincidir el número de parámetros que se envían a la función en el momento de su llamada, con el número de parámetros que tenga definida la función.

Veamos un ejemplo sencillo, donde definimos una función a la cual pasamos 2 valores y nos devuelve como valor el resultado de la operación suma:

 EJEMPLO

```
<?PHP
function suma ($x, $y)
{
    $z = $x + $y;
    return $z; // la función devuelve el valor de la suma
}
$resultado = suma (13,8); /*pasamos 2 valores a la función
el valor que devuelve la función se almacena en la variable
$resultado*/
echo $resultado; //visualiza el valor devuelto por la función
?>
```

3.6.4. Paso de parámetros

Como acabamos de ver, los argumentos serán un método que emplearemos para pasar información relevante a la función. Veamos qué tipos de argumentos podemos utilizar.

- Por valor.
- Por referencia.
- Por defecto.

3.6.5. Por valor

El paso de parámetros por defecto es el paso por valor. Esto significa que al pasar un argumento a una función, ésta realiza una copia de ese valor para trabajar con él en el interior de la función.

Si posteriormente, en el interior de la función, el valor del parámetro cambia, esto no va a significar que el valor que contiene la llamada a la función vaya a cambiar.

EJEMPLO

```

<?PHP
function cuadrado ($a)
{
    $a = $a*$a;
    return $a; // la función devuelve el valor del cuadrado de
    la variable
}
$a=5;
echo $a; //visualiza el valor antes de llamar a la función
echo "<br>";
echo cuadrado($a); //visualiza el valor devuelto por la fun-
ción
echo "<br>";
echo $a; //visualiza el después de llamar a la función y no
ha variado
?>

```

Como podemos observar, a pesar de haber alterado el valor de la variable en el interior de la función, fuera de la función su valor sigue siendo el mismo.

3.6.5.1. Por referencia

En el caso del paso de una variable por referencia, su valor original sí que se verá modificado. Para ello, será necesario añadir el símbolo "&" delante de la variable que va a recibir el argumento, para de esta forma indicarle que la estamos pasando por referencia y deseamos que se modifique su valor si es necesario. Comprobémoslo con el mismo ejemplo, pero pasando la variable por referencia:

EJEMPLO

```
<?PHP
function cuadrado (&$a)
{
    $a = $a*$a;
    return $a; // la función devuelve el valor de la suma
}
$a=5;
echo $a; //visualiza el valor antes de llamar a la función
echo "<br>";
echo cuadrado($a); //visualiza el valor devuelto por la función
echo "<br>";
echo $a; //visualiza el después de llamar a la función y no ha variado
?>
```

Si lo ejecutamos, podremos comprobar que en el último comando que envía el valor de la variable a la pantalla, la variable ha cambiado de valor.

3.6.5.2. Por defecto

Podremos utilizar en las funciones valores por defecto que puedan ser utilizados si se realiza una llamada a una función sin argumentos, cuando ésta en el interior los va a necesitar. Este valor tendrá que ser un valor constante.

EJEMPLO

```
<?PHP
function micasa ($color = "amarillo") {
    return "Mi casa es de color $color.\n";
}
echo micasa ();
echo "<br>";
echo micasa ("rojo");
?>
```

Si ejecutamos el script, podremos ver que en la visualización del resultado de la primera llamada a la función, al no pasarle ningún valor en la llamada, visualizará el valor por defecto que tiene definida la función. En la segunda llamada, visualizará el color que le hemos pasado en la llamada a la función.

Otro aspecto importante a destacar es que cuando usemos parámetros por defecto, tendrán que estar siempre situados a la derecha de otros parámetros que hayamos pasado sin valor por defecto. Veamos con el siguiente ejemplo una manera correcta de realizarlo:

EJEMPLO

```
<?PHP
function micasa ($color1, $color2 = "amarillo") {
    return "Mi casa es de color $color1 y color $color2.\n";
}
echo micasa ("rojo");
?>
```

Si cambiáramos el orden de las variables en la definición de la función, obtendríamos un error en la ejecución del script.

3.6.6. Operador return

Como ya hemos podido comprobar, los valores que devuelve una función serán enviados mediante la instrucción opcional **"return"**.

Mediante esta instrucción podremos devolver cualquier tipo de valor, pero no podremos devolver múltiples valores desde una función. Podríamos saltarnos esta prohibición formal, devolviendo un array, que de por sí, incluirá varios valores. Veamos un ejemplo:

EJEMPLO

```
<?PHP
function varios_valores() {
    return array (2, 3, 6, 8, 14);
}
$miarray=varios_valores();
for($i=0;$i<=4;$i++)
{
    echo $miarray[$i];
    echo "<br>";
}
?>
```

3.6.7. Librerías de funciones para estructurar tu aplicación web

Como ya hemos explicado, uno de los principales objetivos en la utilización de funciones, es la reutilización del código con la consiguiente optimización y ahorro de código que ello supone.

El objetivo ideal de nuestros desarrollos será la combinación del código nuevo junto con el uso de otro código que ya hayamos desarrollado, y que habremos guardado previamente en otros ficheros, donde podremos almacenar funciones que usemos habitualmente.

En nuestro proceso creativo de desarrollo web, siempre nos va a costar más tiempo y esfuerzo el modificar, probar y documentar un código que el crearlo. Por lo tanto, una buena práctica siempre será reutilizar este código que ya sabemos de su perfecto comportamiento, más que incorporarlo completo con nuevas revisiones en el nuevo código.

Por ejemplo, tendremos ficheros que almacenarán el código con distintas funcionalidades. Los siguientes pueden ser alguno de estos ejemplos:

- Fichero con funciones para manejo de hora y fechas.
- Fichero para administración y mantenimiento de bases de datos.
- Fichero para validación de datos introducidos por el usuario.
- Fichero de conexión a una base de datos.

La lista podría ser interminable.

En el siguiente capítulo vamos a ver dos funciones, que nos van a permitir reutilizar este código diseñado en otros ficheros, y que nos ayudarán a obtener una estructura mejor y más clara de nuestro código.

3.7. Include y Require

PHP nos va a ofrecer mediante estas dos funciones la posibilidad de poder leer y ejecutar el código que no va a estar localizado en el script desde el que realicemos la llamada a estas funciones, sino que se encontrará en un fichero diferente. Este fichero podrá ser otro script PHP o podrá ser cualquier otro fichero de tipo HTML, CSS, TXT, por ejemplo.

Esto nos supone una gran ventaja en cuanto a optimizar el código, ya que podremos emplazar en esos ficheros, por ejemplo, funciones u otro tipo de código, de tal forma que no tengan que ser reescritos cada vez que los necesitemos, pudiendo ser utilizados por diferentes scripts.

Ahora veamos el funcionamiento de cada una.

Con la función REQUIRE() se reemplazará el contenido del fichero al que estemos realizando la llamada. Suele ser utilizado para definir constantes o funciones que no van a ser modificadas en el script que realiza la llamada.

Por ejemplo, si tuviéramos un fichero al que llamamos “**constantes.php**” y en el interior escribimos el siguiente código:

EJEMPLO

```
<?PHP
define ("CIUDAD", "Zaragoza");
define ("COMUNIDAD", "Aragon");
?>
```

Y en el script con nombre “**ejemplo.php**” quisiéramos hacer uso de ellas, tendríamos que hacerlo de la siguiente forma:

EJEMPLO

```
<?PHP
require ( 'constantes.php' );
echo (CIUDAD);
echo (COMUNIDAD);
?>
```

Es importante que el archivo al que llamamos cumpla correctamente todas las reglas de sintaxis de PHP, sobre todo las etiquetas de comienzo y fin de código.

A diferencia de INCLUDE() que veremos a continuación, REQUIRE() siempre va a leer el archivo al que se realiza la llamada, incluso si la línea en la que lo estamos llamando nunca es ejecutada. No obstante, si dicha línea no se ejecuta, tampoco se ejecuta el código del archivo al que estamos llamando.

La principal desventaja de `REQUIRE()` es que no puede ser utilizada en un bucle para llamar a diferentes ficheros.

Mediante la función `INCLUDE()` también se va a acceder a un fichero externo, pero al contrario de la función `REQUIRE()`, sí será capaz de procesar el código que contiene cada vez que se realice la llamada a este fichero. También podrá ser utilizada dentro de bucles. Veamos un ejemplo:

EJEMPLO

```
<?PHP
$archivos = array('primero.inc', 'segundo.inc', 'tercero.
inc');
for ($i = 0; $i < count($archivos); $i++) {
include $archivos[$i];
}
?>
```

En el anterior ejemplo, definimos un array que incluye los nombres de los archivos a los que realizaremos la llamada a través de la función `INCLUDE()`.

A continuación, en cada iteración del bucle `FOR`, que se repetirá tantas veces como archivos hayamos incluido en el array, se realizará una llamada a cada uno de estos archivos, ejecutando el código que se encuentre en su interior.

Para finalizar con este capítulo, vamos a mostrar un ejemplo que suele ser muy utilizado por los programadores. Una de las operaciones más frecuentes es definir por una parte, lo que se quiere mostrar en la cabecera de una página web, y por otra parte, escribir lo que se quiere mostrar en el pie de página. Para realizar estas operaciones de una forma estructurada, se escribe el código de la cabecera en un fichero `.php` (por ejemplo, `header.php`), y el pie de página, en otro fichero `.php` (por ejemplo, `pie.php`).

De esta forma, cada vez que quieran ser utilizados, bastará con insertar un `include`, pasándole como parámetro el nombre del archivo que queramos mostrar (`header.php` o `pie.php`).

Veamos un ejemplo de código para estos dos archivos, y un tercero que será el que realice las llamadas a los otros dos.

El fichero **HEADER.PHP** podría quedar de la siguiente forma:

EJEMPLO

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" dir="ltr"
lang="es">
    <head>
    <title><?php echo $titulo ?></title>
    <meta .. />
    </head>
    <body>
    <h1>Comienzo de CABECERA</h1>
    <ul id="menu">
    <li><a>Inicio</a></li>
    <li><a>Menu</a></li>
    </ul>
```

El fichero **PIE.PHP** podría quedar de la siguiente forma:

EJEMPLO

```
<p id="pie">PIE DE PAGINA</p>
</body>
</html>
```

El fichero que tendríamos que utilizar para realizar las llamadas podría quedar de la siguiente forma:

EJEMPLO

```
<?php
    $titulo = "Título de la página";
    include('header.php');
    ?>
    <p> Texto a incluir en cada pagina</p>
    <?php
    include('pie.php');
    ?>
```

3.8. Manejo de fechas y tiempo


Veamos unas cuantas funciones predefinidas que ayudarán a realizar operaciones con las fechas y horas.

Función `checkdate(int mes, int día, int año)`

Nos devolverá un valor verdadero en el caso de que la fecha sea válida, y si no, devolverá uno falso. Dicha comprobación la llevará a cabo a través de los argumentos que le facilitemos. Las condiciones que comprueba para cada uno de los siguientes argumentos son las siguientes:

- **Año.** Que esté comprendido entre 0 y 32767, ambos incluidos.
- **Mes.** Que esté comprendido entre 1 y 12.
- **Día.** El rango de días en función del mes pasado como argumento, teniendo en cuenta años bisiestos.

Veamos un ejemplo:

 EJEMPLO

```
<?php
$mes=8;
$día=2;
$año=2011;
IF (checkdate ($mes, $día, $año))
{
echo "La fecha es correcta";
}
ELSE
{
echo "La fecha no es correcta";
}
?>
```

Función `string date(string format [, int timestamp])`

Con esta función podremos dar formato a la fecha y hora local en función del formato que le hayamos pasado como cadena, utilizando el valor de *timestamp* dado o la hora local actual si no hay parámetro. El resultado final será que la función devuelve una cadena formateada.



NOTA

En la mayoría de las funciones de fecha/hora de PHP utilizaremos el término *timestamp*, que puede ser traducido como “registro de hora”. Se trata del tiempo transcurrido en segundo desde una fecha de referencia (1 de enero de 1970).

Podremos elegir entre mostrar solo la hora, solo la fecha o ambas opciones e incluso mostrar los días y meses, pero en inglés.

Podemos utilizar los siguientes códigos.

- a - “am” o “pm”.
- A - “AM” o “PM”.
- d - día del mes, de “01” a “31”.
- D - día de la semana, en texto, con tres letras; por ejemplo, “Fri”.
- F - mes, en texto, completo; por ejemplo, “September”.
- h - hora, de “01” a “12”.
- H - hora, de “00” a “23”.
- g - hora, sin ceros, de “1” a “12”.
- G - hora, sin ceros; de “0” a “23”.
- i - minutos; de “00” a “59”.
- j - día del mes sin cero inicial; de “1” a “31”.
- l (“L” minúscula) - día de la semana, en texto, completo; por ejemplo, “Monday”.
- L - “1” o “0”, según si el año es bisiesto o no.
- m - mes; de “01” a “12”.
- n - mes sin cero inicial; de “1” a “12”.
- M - mes, en texto, tres letras; por ejemplo, “Sep”.
- s - segundos; de “00” a “59”.
- S - sufijo ordinal en inglés, en texto, dos caracteres; por ejemplo, “th”, “nd”.
- t - número de días del mes dado; de “28” a “31”.
- U - segundos desde el valor de “epoch”.
- w - día de la semana, en número, de “0”(domingo) a “6”(sábado).

- **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").

Veamos un ejemplo de su utilización:

EJEMPLO

```
<?PHP
echo "Hola a todos!!! <br><br>";
echo "Hoy es " . " ", date ("d/m/Y") , " y la hora actual es"
. " ",
date ("h:i:s");
echo "<br><br>O se lo prefieres de esta forma: <br><br>";
echo (date("l dS of F Y h:i:s A"));
?>
```

Función `array_getdate(int timestamp)`

Cuando ejecutemos esta función se devolverá un array con la información de la fecha en formato timestamp con los siguientes elementos posibles:

- **"seconds"** - segundos.
- **"minutes"** - minutos.
- **"hours"** - horas.
- **"mday"** - día del mes.
- **"wday"** - día de la semana, en número.
- **"mon"** - mes, en número.
- **"year"** - año, en número.
- **"yday"** - día del año, en número; por ejemplo, "302".
- **"weekday"** - día de la semana, en texto, completo; por ejemplo, "Monday".
- **"month"** - mes, en texto, completo; por ejemplo, "September".

Veamos un ejemplo de su utilización:

EJEMPLO

```
<?PHP
$fecha = getdate();
echo ("Día: " . $fecha["mday"]."<br>");
echo ("Mes: " . $fecha["mon"]."<br>");
echo ("Año: " . $fecha["year"]."<br>");
echo ("Hora: " . $fecha["hours"]."<br>");
echo ("Minutos: " . $fecha["minutes"]."<br>");
echo ("Segundos: " . $fecha["seconds"]."<br>");
?>
```

Función `int mktime (int hour, int minute, int second, int month, int day, int year [, int is_dst])`

Mediante esta función, pasándole, por ejemplo, como parámetros la fecha o la hora, nos devolverá el valor timestamp para que de esta forma podamos utilizarlo en otras funciones.

Se podrán eliminar argumentos en orden de derecha a izquierda; en los argumentos omitidos se toma el valor de la fecha y hora locales.

Para probarlo, podemos introducir la fecha 13-08-2011 19:15:00:

EJEMPLO

```
<?PHP
$fecha = mktime(19,15,00,8,13,2011);
echo $fecha;
echo "<br>";
echo date("d-M-Y",mktime(19,15,00,8,13,2011));
?>
```

También nos puede ser muy útil para realizar la comprobación de una fecha, de tal forma que si es incorrecta, realiza una aproximación a la correcta. Por ejemplo, vamos a probar la fecha 32 de Agosto de 2011:


EJEMPLO

```
<?PHP
echo date("d-M-Y",mktime(0,0,0,8,32,2011));
?>
```

Función `int time()`

Esta función nos devolverá la hora de la fecha actual en formato timestamp.

Veamos un ejemplo:

EJEMPLO

```
<?PHP
$hora=time();
echo $hora;
echo "<br>";
$hora=getdate(time());
echo $hora["hours"].":".$hora["minutes"].":".$hora["seconds"];
?>
```

3.9. Redirección de página

De entre las muchas funciones que nos suministra PHP predefinidas, hay una de ellas que podremos utilizar con el objetivo de poder redireccionar nuestro script a una página web en concreto. Se trata de la función **header()**.

La función `header()` es usada para enviar encabezados HTTP sin formato. De entre las diferentes posibilidades que nos ofrece, tenemos un parámetro denominado "Location", a través del cual podremos especificar la página web a la que queremos redirigir nuestra página. Veamos un ejemplo para comprobar su funcionamiento:

EJEMPLO

```
<?PHP
header("Location: http://www.seas.com/"); /* Redirección del
navegador */
?>
```

3.10. Funciones isset() y unset()

Podremos utilizar la función `isset()` para determinar si una variable ha sido inicializada con un valor, de tal forma que si ha sido asignado un valor, devolverá `true`.

Podremos utilizar la función `unset()` para destruir una variable, y por lo tanto, liberar los recursos dedicados a la misma. Como único parámetro recibirá el nombre de la variable que queremos destruir.

Veamos un ejemplo de ambos:

EJEMPLO

```
<?php
$Nombre = "Silvia";
if (isset($Nombre)) {
    echo ("La variable Nombre tiene un valor: ".$Nombre);
}
unset($Nombre); /*Podemos comprobar qué pasa si liberamos la
variable $Nombre */
if (isset($Nombre)) {
    echo ("La variable Nombre aún existe.");
}
else {
    echo ("La variable Nombre ya no existe.");
}
?>
```

En este capítulo vamos a dar los primeros pasos para ir conociendo la forma de trabajo del lenguaje PHP y lo finalizaremos con el desarrollo de nuestro primer script básico de PHP.

RESUMEN

- Hemos conocido los diferentes tipos de variables que podemos utilizar en nuestro código.
- Hemos practicado la forma de averiguar el tipo de dato que contiene una variable.
- Hemos definido los ámbitos de una variable, así como el uso de variables por referencia, las variables de sistema y las variables superglobales.
- Hemos explicado cómo utilizar operadores de asignación, concatenación, aritméticos, incremento, decremento, condicionales, de ejecución y lógicos.
- Hemos conocido las principales estructuras de control, tanto condicionales como IF.. ELSE o SWITCH, a bucles como FOR, FOREACH, WHILE y DO..WHILE.
- Hemos visto la forma de utilizar arrays, tanto asociativos como multidimensionales y las diferentes técnicas que podemos utilizar para recorrerlos.
- Hemos explicado el uso y propósito de las funciones para la optimización y ahorro de nuestro código, así como la forma de pasar argumentos a estas funciones por valor, por referencia o por defecto.
- Hemos visto como estructurar mejor nuestro código, utilizando el código localizado en otros ficheros mediante las funciones include() y require().
- Hemos conocido y practicado las principales funciones para el manejo de fechas y horas.

