PHP

1. Variables en PHP
2. Tipos de Variables
3. Modificar tipos de variables
4. Variables Globales
5. Variables del Sistema
6. Arrays
7. Manejos de Cadenas
8. Parámetros en Funciones en PHP
9. Retornar Valores en Funciones
10. Condiciones IF
11. Bucles

Variables en PHP

Las variables son uno de los primeros temas a conocer en PHP y en la mayoría de los lenguajes de programación.

Conceptos y tipos de datos que podremos encontrar en el lenguaje PHP:

Anteriormente en JavaScript, ya introducimos el concepto de variable. No obstante podemos entender una variable como un dato almacenado en una referencia. Técnicamente una variable apunta a una posición de la memoria, donde se almacena un dato. Las variables se utilizan en los lenguajes de programación para darle un nombre a ese dato, a esa posición de la memoria, de manera que se pueda entender o saber lo que contiene. Al final, esos datos almacenados son los que se utilizan para conseguir los resultados de los programas.

Por su parte, un tipo de datos es la característica de un dato almacenado. Es decir, si el dato posee información numérica, alfanumérica, etc. La mayoría de los lenguajes clasifican de alguna manera los tipos de datos, aunque algunos son más permisivos que otros a la hora de realizar operaciones con variables de distintos tipos.

$ en el nombre de las variables
Para PHP, las variables eran definidas comenzando siempre por el símbolo $.

Las variables siempre deberían tener un nombre descriptivo sobre lo que ellas van a almacenar. Por tanto, al nombre de una variable en PHP le colocaremos el símbolo $ y un nombre que ayude a entender la funcionalidad del dato almacenado en el sistema que se está desarrollando.

Por ej:

<?php $totalticket = 300 ?>

Tipos de datos en PHP
Dependiendo de la información que contenga, una variable puede ser considerada de diferentes tipos:

Variables numéricas
Este tipo de variables almacena cifras, números, que pueden tener dos clasificaciones distintas:

Enteros	$entero=2002;	//Números sin decimales
Reales	$real=3.14159;	//Números con o sin decimal

Variables alfanuméricas
Este tipo de datos almacena textos compuestos, cadenas de caracteres, que pueden contener letras, símbolos y números o cifras.

Cadenas
Almacenan variables alfanuméricas por ej: $cadena="Feliz año 2020!";

Boleanas
Este tipo de variables almacena un valor lógico, que puede valer verdadero o falso. Es muy común en la programación este tipo de variables boleanas.

Boleano verdadero $verdadero = true;
Boleano falso $falso = false;

Matrices, tablas o arrays
Es un tipo de datos en el que, en lugar de tener un dato, podemos almacenar un conjunto de ellos, a los que accedemos a través de índices. Cada una de las casillas de un array o los datos de nuestra matriz a su vez poseen informaciones numéricas, alfanuméricas etc.

Arrays Son las variables que guardan las tablas
$sentido[1]="ver";
$sentido[2]="tocar";
$sentido[3]="oir";
$sentido[4]="gusto";
$sentido[5]="oler";

Objetos
Se trata de conjuntos de variables y funciones asociadas. Presentan una complejidad mayor que las variables vistas hasta ahora pero su utilidad es más que interesante. Entraremos con detalle en los objetos más adelante.

PHP tiene tipado dinámico
A diferencia de otros lenguajes, PHP posee una gran flexibilidad a la hora de operar con variables. En efecto, cuando definimos una variable asignándole un valor, el ordenador le atribuye un tipo. Si por ejemplo definimos una variable entre comillas, la variable será considerada de tipo cadena:

$variable="5"; //esto es una cadena

Sin embargo, si pedimos en nuestro script realizar una operación matemática con esta variable, no obtendremos un mensaje de error sino que la variable cadena será asimilada a numérica (PHP hará todo lo posible por interpretar nuestra operación):

```
<?
$cadena="5"; //esto es una cadena
$entero=3; //esto es un entero
echo $cadena+$entero
?>
```

Este script dará como resultado "8". La variable cadena ha sido asimilada en entero (aunque su tipo sigue siendo cadena) para poder realizar la operación matemática. Del mismo modo, podemos operar entre variables tipo entero y real. No debemos preocuparnos de nada, PHP se encarga durante la ejecución de interpretar el tipo de variable necesario para el buen funcionamiento del programa.

Nota: Los lenguajes como PHP que permiten mayor flexibilidad en los tipos de las variables se dicen que tienen tipado dinámico. En ellos una variable puede tener distintos tipos a lo largo de su vida, es decir, a medida que el programa se ejecuta una variable podrá cambiar de tipo. Generalmente durante el procesamiento del programa se va infiriendo los tipos de las variables, en tiempo de ejecución, según el tipo de datos del valor que se le asigna o las operaciones que se realizan sobre ellas. Otra manera de referirse a este tipo de lenguajes de programación es "levemente tipados", aunque esta segunda denominación es menos correcta, porque puede inducir a

una comprensión errónea, ya que en la realidad las variables siempre tienen tipos, aunque estos puedan variar con el tiempo.

PHP es sensible a las mayúsculas y minúsculas
PHP entiende de manera distinta las mayúsculas y minúsculas. En el caso del nombre que le damos a una variable, no es lo mismo escribirla con mayúscula o minúscula, o mezclando mayúsculas y minúsculas de distinta manera. Por tanto, hay que tener mucho cuidado a la hora de escribir los nombres de variables, y no cambiar mayúsculas por minúsculas, ya que PHP entenderá dos variables distintas aunque nosotros podamos intentar referirnos a la misma. Cuando estamos empezando quizás sea un buen consejo trabajar asignando nombres a las variables siempre en minúsculas, para evitar este tipo de malentendidos a veces muy difíciles de localizar.

En el caso que tengamos una variable con un nombre compuesto de varias palabras, en PHP es una práctica común colocar la variable toda en minúscula y separar las palabras por guiones bajos.

```
<?php $mi_variable = "me gusta PHP" ?>
```

Variables asignadas por referencia
En PHP también podemos asignar variables por referencia, aunque a decir verdad no es muy utilizado. En ese caso no se les asigna un valor, sino otra variable, de tal modo que las dos variables comparten espacio en memoria para el mismo dato.

La notación para asignar por referencia es colocar un "&" antes del nombre de la variable.

```
<?php
$foo = 'Roberto'; // Asigna el valor 'Roberto' a $foo
$bar = &$foo; // Referencia $foo vía $bar.
$bar = "Mi nombre es $bar"; // Modifica $bar...
echo $foo; // $foo también se modifica.
echo $bar;
?>
```
Esto dará como resultado la visualización dos veces del string "Mi nombre es Roberto". Algo como:

```
Mi nombre es  Roberto
Mi nombre es  Roberto
```

Formas en que una variable de PHP puede ver variado su tipo.

Como vimos, PHP no requiere que indiquemos el tipo que va a contener una variable, sino que lo deduce del valor que asignemos a la variable. Asimismo, se encarga de actualizar automáticamente el tipo de la variable cada vez que le asignamos un nuevo valor. Esto es básicamente lo que se llama "tipado dinámico" o "tipado débil", característica no sólo de PHP, sino de muchos otros lenguajes como Javascript.

Por ello, para cambiar el tipo de una variable simplemente le asignamos un valor con un nuevo tipo.

```
$cadena = 'esto es una cadena';
$cadena = 34 //La variable $cadena cambió de tipo
```

Nota: Se excluyen en este caso el cambio de variables a tipo Array.

```
$a = "1";
//$a es una cadena
$a[0] = "f";
//¿Estamos editando el índice de la cadena o forzando a array?
```

Veremos dos tipos posibles de alteración del tipo de variables, más allá del propio que hace PHP con el comportamiento derivado de su tipado dinámico. A esta operación se la conoce habitualmente como "Forzado".

Forzado
Variar el tipo de datos que contiene una variable con el tiempo es una cosa que no siempre es aconsejable, porque si no tenemos certeza de si una variable contiene un dato de un tipo u otro, a veces los resultados obtenidos pueden no ser los esperados.

Para evitar problemas en muchas ocasiones puede ser útil realizar el forzado de una variable a un tipo en concreto, de manera explícita, lo que nos permitirá saber que cuando llega el flujo del programa a un punto dado, aquella variable tendrá el tipo de datos esperado. En PHP existen diversas maneras de forzar una variable a un tipo.

Casting de variables
Hay otra manera de realizar un forzado, para que una variable se comporte como un tipo determinado. Ahora vamos a ver otro mecanismo de forzado que es similar al de otros lenguajes como C o Java.

```
$variable = "23";
$variable = (int) $variable;
```

Los forzados permitidos son:

(int), (integer) - fuerza a entero (integer)
(real), (double), (float) - fuerza a número con decimales (coma flotante)
(string) - fuerza a cadena (string)
(array) - fuerza a array (array)
(object) - fuerza a objeto (object)
(unser) - fuerza a null
(binary) - fuerza a "binary string"

Cuando hay una incongruencia de tipos PHP siempre intenta hacer lo más adecuado con el código que ejecuta, pero no siempre la solución que toma el lenguaje es la que estamos necesitando. En esos casos, el forzado será realmente importante.

Ámbito de las variables en PHP

En cualquier lenguaje de programación las variables tienen un ámbito, que es el lugar o lugares donde tienen validez. El ámbito varía en función de donde se hayan creado esas variables, pudiendo ser globales o locales.

En PHP, todas las variables creadas en la página, fuera de funciones, son variables globales a la página. Por su parte, las variables creadas dentro de una función son variables locales a esa función.

Las variables globales se pueden acceder en cualquier lugar de la página, mientras que las variables locales sólo tienen validez dentro de la función donde han sido creadas. De modo que una variable global la podemos acceder dentro de cualquier parte del código, mientras que si intentamos acceder a una variable local fuera de la función donde fue creada, nos encontraremos con que esa variable no tiene contenido alguno.

Si intentamos acceder a una variable global dentro de una función, en principio también nos encontraremos con que no se tiene acceso a su valor. Esto es así en PHP por motivos de claridad del código, para evitar que se pueda prestar a confusión el hecho de usar dentro de una función una variable que no ha sido declarada por ningún sitio cercano.

Entonces, si queremos utilizar una variable global a la página dentro de una función, tenemos que especificar de alguna manera que esa variable que vamos a utilizar es una global. Existen en PHP maneras de utilizar variables globales a la página dentro de una función. Son las siguientes:

Matriz GLOBALS
Existe un array en PHP llamado $GLOBALS, que guarda una referencia a todas las variables creadas de manera global a la página. Es una matriz o array asociativo, de los que en lugar de índices numéricos utilizan índices de texto, donde cada índice es el nombre que hemos dado a la variable y cada valor es el contenido de cada variable.

Supongamos que tenemos esta declaración de variables globales a la página, es decir, fuera de cualquier función:

```
$mivariable = "pepe";
$otravariable = 1234;
```
Si queremos acceder a esas variables dentro de una función utilizando el array $GLOBALS tendríamos este código:

```
function mifuncion(){
  //estoy dentro de la función, para aceder a las variables utilizo $GLOBALS
  echo $GLOBALS["mivariable"];
  echo $GLOBALS["otravariable"];
}
```
Como se puede ver, se accede al contenido de las variables globales con el array $GLOBALS, utilizando como índices de la matriz los nombres de variables que deseamos mostrar.

Esto imprimiría por pantalla el texto "pepe1234", el valor de las dos variables uno detrás del otro.

Declaración de uso de variables globales dentro de una función
Otra cosa que podemos hacer para acceder a variables globales dentro de una función es especificar al comienzo de dicha función la lista de variables que vamos a utilizar dentro. Para especificar esas

variables utilizamos la palabra "global" seguida de la lista de variables que se van a utilizar del entorno global.

```
function mifuncion(){
  global $mivariable, $otravariable;
  //con esa línea dentro de la función, declaramos el uso de variables globales
  echo $mivariable;
  echo $otravariable;
}
```

Como vemos, con "global" se especifica que vamos a utilizar unas variables que fueron declaradas como globales a la página. Una vez hecho esto, ya podemos acceder a esas variables globales como si estuvieran declaradas dentro de la función.

Cualquier alteración que hagamos a las variables dentro de la función permanecerá cuando se haya salido de la función, tanto si accedemos a través del array $GLOBALS o declarando con "global" el uso de esas variables.

Variables de sistema en PHP

PHP es un lenguaje que se ejecuta en el servidor, bajo demanda de un cliente. Por tanto, la ejecución de PHP se produce dentro de un marco muy concreto, donde intervienen varios actores, principalmente el cliente (generalmente el usuario que entra usando su navegador) y el servidor (donde se ejecuta el código PHP, que básicamente debe producir la salida que se enviará al cliente).

Dentro de una página PHP tendremos por tanto acceso a toda una serie de variables que nos informan sobre nuestro servidor y sobre el cliente que ha solicitado una determinada página. A estas informaciones, que podemos recoger en forma de variables, las llamamos "variables de sistema".

La información de estas variables es atribuida por el servidor y muchas varían de valor según el momento en que se consultan los datos que almacena.

$_SERVER
La mayoría de las variables de sistema las podemos recibir a partir de un array denominado $_SERVER.

$_SERVER es un array asociativo, cuyos índices son cadenas de texto y no números.

Técnicamente $_SERVER se conoce como una "variable superglobal" y posee una multitud de datos asociados al array $_SERVER, algunos sin utilidad aparente y otros realmente interesantes y con una aplicación directa para nuestras aplicaciones web. Enumeramos algunas de estas variables y la información que nos aportan:

$_SERVER["HTTP_USER_AGENT"] Nos informa principalmente sobre el sistema operativo y tipo y versión de navegador utilizado por el internauta. Su principal utilidad radica en que, a partir de esta información, podemos redireccionar nuestros usuarios hacia páginas optimizadas para su navegador o realizar cualquier otro tipo de acción en el contexto de un navegador determinado.

$_SERVER["HTTP_ACCEPT_LANGUAGE"] Nos devuelve la o las abreviaciones de la lengua considerada como principal por el navegador. Esta lengua o lenguas principales pueden ser elegidas en el menú de opciones del navegador. Esta variable resulta también extremadamente útil para enviar al internauta a las páginas escritas en su lengua, si es que existen.

$_SERVER["HTTP_REFERER"] Nos indica la URL desde la cual el internauta ha tenido acceso a la página. Muy interesante para generar botones de "Atrás" dinámicos o para crear nuestros propios sistemas de estadísticas de visitas.

$_SERVER["PHP_SELF"] Nos devuelve una cadena con la URL del script que está siendo ejecutado. Muy interesante para crear botones para recargar la página.

$_SERVER["HTTP_GET_VARS"] Se trata de un array que almacena los nombres y contenidos de las variables enviadas al script por URL o por formularios GET.

$_SERVER["HTTP_POST_VARS"] Se trata de un array que almacena los nombres y contenidos de las variables enviadas al script por medio de un formulario POST.

$_SERVER["HTTP_COOKIE_VARS"] Se trata de un array que almacena los nombres y contenidos de las cookies. Veremos qué son más adelante.

$_SERVER["PHP_AUTH_USER"] Almacena la variable usuario cuando se efectúa la entrada a páginas de acceso restringido. Combinado con $_SERVER["PHP_AUTH_PW"] resulta ideal para controlar el acceso a las páginas internas del sitio.

$_SERVER["PHP_AUTH_PW"] Almacena la variable password cuando se efectúa la entrada a páginas de acceso restringido. Combinado con $_SERVER["PHP_AUTH_USER"] resulta ideal para controlar el acceso a las páginas internas del sitio.

$_SERVER["REMOTE_ADDR"] Muestra la dirección IP del visitante.

$_SERVER["DOCUMENT_ROOT"] Nos devuelve el path físico en el que se encuentra alojada la página en el servidor.

$_SERVER["PHPSESSID"] Guarda el identificador de sesión del usuario. Veremos más adelante en qué consisten las sesiones.

No todas estas variables están disponibles en la totalidad de servidores o en determinadas versiones de un mismo servidor. además, algunas de ellas han de ser previamente activadas o definidas por medio de algún acontecimiento. Así, por ejemplo, la variable $HTTP_REFERER no tendrá un valor definido, a menos que el usuario acceda al script a partir de un enlace desde otra página.

Si quieres ver cuál es el conjunto completo de las variables del sistema que dispones dentro de $_SERVER en tu entorno, es suficiente con escribir y ejecutar una página PHP que contenga este código:

```
<?php
var_dump($_SERVER);
?>
```

Eso realizará un listado de todo el contenido del array asociativo $_SERVER y lo mostrará como salida en la página web.

Variables superglobales
A partir de PHP 4.1.0, se dispone de un conjunto de variables de tipo array que mantienen información del sistema, llamadas "superglobales" porque se definen automáticamente en un ámbito global y a las que se puede acceder desde cualquier punto del código PHP.

Estas variables ya existían anteriormente en PHP, aunque se accedían desde otros arrays.
La lista de estas variables superglobales de PHP es la siguiente:

$GLOBALS
Contiene una referencia a cada variable disponible en el espectro de las variables del script. Las llaves de esta matriz (índices del array) son los nombres de las variables globales. $GLOBALS existe dese PHP 3.

$_SERVER
Variables definidas por el servidor web ó directamente relacionadas con el entorno en don el script se esta ejecutando. Es equivalente a lo que antes se conocía como $HTTP_SERVER_VARS. Son las variables de sistema que hemos explicado antes en este artículo.

$_GET

Variables proporcionadas al script por medio de HTTP GET. Es equivalente a lo que antes se conocía como $HTTP_GET_VARS.

$_POST
Variables proporcionadas al script por medio de HTTP POST. Es equivalente a lo que antes se conocía como $HTTP_POST_VARS.
$_COOKIE
Variables proporcionadas al script por medio de HTTP cookies. Es equivalente a lo que antes se conocía como $HTTP_COOKIE_VARS.

$_FILES
Variables proporcionadas al script por medio de la subida de ficheros via HTTP . Es equivalente a lo que antes se conocía como $HTTP_POST_FILES.

$_ENV
Variables proporcionadas al script por medio del entorno. Es equivalente a lo que antes se conocía como $HTTP_ENV_VARS.

$_REQUEST
Variables proporcionadas al script por medio de cualquier mecanismo de entrada del usuario. La presencia y el orden en que aparecen las variables en esta matriz es definido por la directiva de configuración variables_order. Esta matriz no tiene un análogo en versiones anteriores a PHP 4.1.0.

$_SESSION
Variables registradas en la sesión del script. Es equivalente a lo que antes se conocía como $HTTP_SESSION_VARS. Vea también la sección Funciones para el manejo de sessiones para más información.

Tablas o Arrays en PHP

Una variable generalmente almacena un dato, ya sea de tipo cadena, numérico, etc. Un array es como una variable capaz de almacenar un conjunto de datos. También los podemos conocer con el nombre de "arreglo", "tabla" o "matriz".

Dado que en un array somos capaces de almacenar varios elementos, es necesario el uso de un índice para poder referirnos a cada uno de ellos. Ese índice a veces se conoce como "clave". Existen en PHP arrays con índices numéricos (los arrays más comunes) y con índices alfanuméricos (también llamados arrays asociativos, muy útiles, pero menos comunes), que veremos en esta unidad.

Arrays comunes, índices numéricos

En capítulos anteriores poníamos el ejemplo de un array llamado sentido que contenía los distintos sentidos del ser humano:

```
$sentido[1]="ver";
$sentido[2]="tocar";
$sentido[3]="oir";
$sentido[4]="gustar";
$sentido[5]="oler";
```

En este caso este array cataloga sus elementos, comúnmente llamados valores, por números. Los números del 1 al 5 son por lo tanto las claves y los sentidos ("tocar", "oir"...) son los valores asociados.

Arrays asociativos

Si lo deseamos, es posible emplear nombres (cadenas) para clasificar los elementos del array. Lo único que deberemos hacer es entrecomillar las llaves alfanuméricas y entonces tendremos un array asociativo:

```
$moneda["argentina"]="Peso";
$moneda["francia"]="Euro";
$moneda["usa"]="Dolar";
```

Otra forma de definir idénticamente este mismo array y que nos puede ayudar para la creación de arrays más complejos es la siguiente sintaxis:

```
<?
$moneda=array("argentina"=> "Peso","francia" => "Euro","usa" => "Dolar");
?>
```

Arrays multidimensionales

Otra forma de almacenar datos es mediante la creación de arrays multidimensionales (tablas o matrices con más de una dimensión).

Por ejemplo: Queremos almacenar dentro de una misma tabla el nombre, moneda y lengua de cada país. Para hacerlo podemos emplear un array llamado país que vendrá definido por estas tres características (claves). Para crearlo, deberíamos escribir una expresión del mismo tipo que la vista anteriormente en la que incluiremos una array dentro del otro. Este proceso de incluir una instrucción dentro de otra se llama anidar y es muy corriente en programación:

```
<?
$pais=array
(
"argentina" =>array
    (
    "nombre"=>"República Argentina",
    "lengua"=>"Castellano",
    "moneda"=>"Pesos"
    ),
"francia" =>array
    (
        "nombre"=>"Francia",
        "lengua"=>"Francés",
        "moneda"=>"Euro"
)
);
echo $pais["argentina"]["moneda"] // Muestra por pantalla: "Pesos"
?>
```

Antes de entrar en el detalle de este pequeño script, comentemos algunos puntos referentes a la sintaxis.

- Como puede verse, en esta secuencia de script, no hemos introducido punto y coma ";" al final de cada línea. Esto es simplemente debido a que lo que hemos escrito puede ser considerado como una sola instrucción. En realidad, somos nosotros quienes decidimos cortarla en varias líneas para, así, facilitar su lectura. La verdadera instrucción acabaría una vez definido completamente el array y es precisamente ahí donde hemos colocado el único punto y coma.

- Por otra parte, podés observar cómo utilizadamos el tabulador para separar del lado izquierdo (indentar) unas líneas más que otras. Esto también lo hacemos por cuestiones de claridad, ya que nos permite ver qué partes del código están incluidas dentro de otras. Es importante acostumbrarse a escribir de esta forma del mismo modo que a introducir los comentarios ya que la claridad de los scripts es fundamental a la hora de depurarlos. Un

poco de esfuerzo a la hora de crearlos puede ahorrarnos muchas horas a la hora de corregirlos o modificarlos meses más tarde.

Pasando ya al comentario del programa, como podéis ver, éste nos permite almacenar tablas y, a partir de una simple petición, visualizarlas un determinado valor en pantalla.

Funciones de Array en PHP

PHP incluye un nutrido conjunto de funciones para trabajar con Arrays. En ellas nos podemos apoyar para realizar toda una serie de operaciones típicas como ordenar elementos por orden alfabético directo o inverso, por claves, contar el numero de elementos que componen el array además de poder movernos por dentro de él hacia delante o atrás.

Array_values(miarray)	Lista los contenidos del array
Asort (miarray) y arsort (miarray)	Ordena el array en funcion de los valores
Count(miarray)	Nos devuelve el numero de elementos
ksort(miarray) y krsort	Ordena en funcion de las claves
list($varuno, $vardos...)=miarray	Asigna a cada variable los valores del array
next(miarray)	Mueve el puntero una posición adelante
prev(miarray), reset(miarray) y end(miarray)	Mueve el puntero atrás y al inicio y al final
each(miarray)	Retorna el puntero actual y lo mueve a la siguiente posición.

Cadenas o strings en PHP

Uno de los tipos de datos más corrientes en la mayoría de los lenguajes son los strings. También podremos conocerlas con el nombre de cadenas o "cadenas de caracteres". No son más que información que contiene texto, con caracteres alfanuméricos, cualquier mezcla de caracteres alfabéticos, símbolos y caracteres numéricos.

Para asignar a una variable un contenido de tipo cadena, lo escribiremos entre comillas, valiendo tanto las comillas dobles como las comillas simples. Por ejemplo:

```
$cadena="Esta es la información la variable de tipo string";
```

Si queremos mostrar en pantalla el valor de una variable o bien un mensaje cualquiera usaremos la instrucción *echo* :

```
echo $cadena; //muestra por pantalla "Esta es la información de mi variable"
```

A la sentencia echo le podemos pasar no solo una variable de tipo cadena, y si no es una cadena hará lo necesario para producir una salida adecuada. Incluso podemos pasarle un literal de cadena:

```
echo "Esta es la información de mi variable"; //daría el mismo resultado
```

Literales de cadena con comillas dobles o comillas simples

Algo característico de PHP es que permite usar tanto comillas simples como comillas dobles y, dependiendo de cómo lo hayamos hecho PHP interpretará las cadenas de manera distinta.

Cadenas con comillas dobles

Si usamos comillas dobles para delimitar cadenas de PHP haremos que el lenguaje se comporte de una manera más flexible y los valores serán analizados para obtener la acción deseada. Lo más destacado es que las variables que coloquemos dentro de las cadenas se sustituirán por los valores. Es mejor verlo con un código.

```
$sitioweb = "Curso FullStack";

$cadena = "Bienvenidos a $sitioweb";

echo $cadena;
```

Ese código producirá como salida "Bienvenidos a Curso FullStack". Es decir, PHP interpolará en la variable $cadena el valor de la variable $sitioweb, sustituyendo $sitioweb por su correspondiente valor.

Dentro de las cadenas delimitadas por comillas dobles hay una gran cantidad de caracteres de escape, mediante los cuales podemos colocar en cadenas de caracteres cosas como saltos de línea,

tabuladores o símbolos "$" que no serían considerados como inicio del nombre de una variable. Luego daremos más detalle sobre esto.

Cadenas con comillas simples

Al delimitar un literal de cadena con comillas simples PHP lo analiza de forma diferente. Ninguna de tus variables se sustituirá por su valor. Por Ejemplo:

```
$sitioweb = 'Curso FullStack’;

$cadena = 'Bienvenidos a $sitioweb';

echo $cadena;
```

Este código fuente es prácticamente igual que el anterior, con la salvedad que estamos usando cadenas delimitadas por comillas simples. La salida es sensiblemente distinta, en este caso nos mostraría "Bienvenidos a $sitioweb", dado que no realiza la interpolación de la variable.

Para obtener el mismo resultado que con comillas dobles tendríamos que concatenar la variable al texto:

```
$sitioweb = 'Curso FullStack’;

$cadena = 'Bienvenidos a ' . $sitioweb;

echo $cadena;
```

¿Qué usar, comillas simples o dobles?

Por lo general se recomienda usar comillas simples, puesto que PHP no realizará el análisis de la cadena.

Concatenación de cadenas

Podemos concatenar varias cadenas utilizando el operador de concatenación de string, que tiene el símbolo punto ".":

```
$cadena1="Curso";

$cadena2=" FullStack";

$cadena3=$cadena1.$cadena2;

echo $cadena3; //El resultado es: "Curso FullStack"
```

Usando comillas dobles podrías colocar esas variables dentro de la cadena. Dejamos aquí otro ejemplo:

```
$a=55;

$mensaje="Tengo $a años";
```

```
echo $mensaje; //El resultado es: "Tengo 55 años"
```

La pregunta que nos podemos plantear ahora es... ¿Cómo hago entonces para que en vez del valor "55" me salga el texto "$a"? En otras palabras, cómo se hace para que el símbolo $ no defina una variable sino que sea tomado tal cual. Esta pregunta es tanto más interesante cuanto que en algunos de scripts este símbolo debe ser utilizado por una simple razón comercial (pago en dólares por ejemplo) y si lo escribimos tal cual, el ordenador va a pensar que lo que viene detrás es una variable siendo que no lo es.

Caracteres de escape

Para incluir el símbolo $, la contrabarra y otros caracteres utilizados por el lenguaje dentro de las cadenas y no confundirlos se usan los caracteres de escape.

Para insertar un caracter de escape tenemos que indicarlo comenzando con el símbolo de la contrabarra (barra invertida) y luego el del caracter de escape que deseemos usar.

Los caracteres de escape disponibles dependen del tipo de literal de cadena que estemos usando. En el **caso de las cadenas con comillas dobles** se permiten muchos más caracteres de escape. Los encuentras en la siguiente tabla:

\n	Salto de Línea
\r	Retorno de Carro
\t	Tabulador horizontal
\v	Tabulador vertical
\e	Tecla de Escape
\f	Tecla de avance de página
\\	Barra invertida
\$	Símbolo dólar
\”	Comillas dobles

Estos cambios de línea y tabulaciones tienen únicamente efecto en el código y no en el texto ejecutado por el navegador. En otras palabras, si queremos que nuestro texto ejecutado cambie de línea hemos de introducir un `echo "<br>"` y no \n.

Nota: El caracter de escape de salto de línea \n sólo cambia de línea en el código HTML creado y enviado al navegador cuando la página es ejecutada en el servidor. Ese salto de línea no tiene valor en el HTML, por lo que solamente lo verías al examinar el código fuente producido al ejecutar el script.
En el **caso de las cadenas expresadas con comillas simples** hay muchos menos caracteres de escape. Primero porque no son necesarios (como el símbolo $, que no puede ser confundido con el inicio de una variable, ya que no las tiene en cuenta) y segundo porque simplemente no se encuentran disponibles.

A continuación la tabla de caracteres de escape permitidos en una cadena encerrada mediante comillas simples:

\’ Comillas simples

\\ Barra Invertida

Sintaxis compleja de las llaves

También podés utilizar arrays entre las comillas.

Como en el siguiente código:

```
$array = array(1, 2, 40, 55);

$cadena = "La posición tres contiene el dato $array[2]";

echo $cadena; //escribe La posición tres contiene el dato 40
```

No surge ningún problema al expandir el valor de la posición 3 del array en la cadena, usando (eso sí) comillas dobles. Incluso aunque el array necesite un índice, PHP sabe que lo que tiene que mostrar ahí es una casilla del array. Veamos el siguiente código:

```
$array = array('uno' => 1, 'dos' => 2, 'tres' => 40, 'cuatro' => 55);

$cadena = "La posición 'tres' contiene el dato $array['tres']"; //esto produce un error!!
```

En este caso nuestro script producirá un error al ser interpretado por PHP, puesto que un array con índice alfanumérico (array asociativo) no es capaz de procesarlo bien cuando lo escribimos dentro de una cadena.

Para salvar esta situación entran en juego la mencionada sintaxis compleja de las llaves. Simplemente vamos a escribir el arraya asociativo que deseamos que PHP sustituya encerrado entre llaves. Así PHP lo reconocerá perfectamente.

```
$array = array('uno' => 1, 'dos' => 2, 'tres' => 40, 'cuatro' => 55);

$cadena = "La posición 'tres' contiene el dato {$array['tres']}"; //Ahora funciona bien

echo $cadena; //escribe La posición 'tres' contiene el dato 40
```

Funciones de cadenas

Las cadenas pueden asimismo ser tratadas por medio de funciones. PHP es un lenguaje muy rico en este sentido, que incluye muchas posibles acciones que podemos realizar sobre ellas con tan solo ejecutar una función: Dividirlas en palabras, eliminar espacios sobrantes, localizar secuencias, remplazar caracteres especiales por su correspondiente en HTML, etc.

Por ejemplo, producir una salida en HTML, en la que cambiamos todos los caracteres especiales de las entidades HTML (útil para evitar que se inyecte código HTML al documento que no queremos que aparezca formateado, sino escrito en la página con sus etiquetas).

```
$cadenaOriginal = '<b>Me gusta FullStack</b>';
```

```
$cadenamodificada = htmlspecialchars($cadenaOriginal);

echo $cadenamodificada; //escribe &lt;b&gt;Me gusta FullStack&lt;/b&gt;
```

Funciones en PHP

Las funciones de PHP nos permiten realizar de una manera sencilla tareas habituales y a la hora de desarrollar una aplicación, pero además nosotros podemos hacer nuevas funciones para resolver todo tipo de tareas más específicas de nuestra aplicación.

Las funciones integradas en PHP son muy fáciles de utilizar y para acceder a las utilidades que hay detrás de una función tan sólo hemos realizar la llamada y especificar los parámetros necesarios para que la función realice su tarea.

Nota: Después de la llegada de PHP 5, en el momento en el que PHP pasó a ser un lenguaje con una orientación a objetos potente, las funciones de la biblioteca del lenguaje tienen en muchos casos alternativas en base a clases y objetos.

Crear nuestras propias funciones en PHP

De una forma general, podríamos crear nuestras propias funciones para conectarnos a una base de datos o crear los encabezados o etiquetas meta de un documento HTML. Para una aplicación de comercio electrónico podríamos crear por ejemplo funciones de cambio de una moneda a otra o de calculo de los impuestos a añadir al precio de articulo. En definitiva, es interesante crear funciones para la mayoría de acciones más o menos sistemáticas que realizamos en nuestros programas.

Aquí daremos el ejemplo de creación de una función que, llamada al comienzo de nuestro script, nos crea el encabezado de nuestro documento HTML y coloca el titulo que queremos a la página:

```
<?

function hacer_encabezado($titulo) {

   $encabezado="<html><head>t<title>$titulo</title></head>";

   echo $encabezado;

}

?>
```

Esta función podría ser llamada al principio de todas nuestras páginas de la siguiente forma:

```
$titulo="Mi web";

hacer_encabezado($titulo);
```

De esta forma automatizamos el proceso de creación de nuestro documento. Podríamos por ejemplo incluir en la función otras variables para construir las etiquetas meta y de esta forma, con un esfuerzo mínimo, crearíamos los encabezados personalizados para cada una de nuestras páginas. De

este mismo modo nos es posible crear cierres de documento o interfaces de la web como podrían ser barras de navegación, formularios de login, etc.

Para crear una función debemos declararla. Para ello usamos la palabra function seguida del nombre de la función. Luego unos paréntesis donde podemos indicar los parámetros que se espera recibir en su invocación y finalmente el bloque de código de la función propiamente dicha, encerrado entre llaves.

Estructurar el código de una aplicación con nuestras propias librerías de funciones

La función ha de ser definida para poder ser utilizada, ya que no se encuentra integrada en PHP sino que la hemos creado nosotros. Si pensamos que en una aplicación web completa podemos tener cientos de funciones definidas por nosotros mismos quizás te asuste que tengas demasiado código de funciones que deben ser definidas antes de ser usadas y que pueden ser incluidas desde un archivo externo. De hecho es muy común que tengamos archivos donde solo colocamos el código de las funciones, almacenando definiciones de las funciones que vayamos creando para realizar un sitio web.

Estos archivos en los que se guardan las funciones se llaman comúnmente librerías. La forma de incluirlos en nuestro script es a partir de la instrucción require o include:

```
require("ruta/a/libreria.php");
```

O si prefieres la alternativa del include:

```
include("ruta/a/libreria.php");
```

Nota: Tanto require() como include() hacen el mismo trabajo, de traerse código que hay en archivos diferentes dentro del servidor, para que podamos utilizarlo al crear una página. La diferencia fundamental entre require e include es que la primera requiere forzosamente algo y la otra no. Es decir, si hacemos un require() de un archivo y éste no se encuentra disponible por cualquier motivo, PHP parará la ejecución del código y devolverá un "Error fatal". Si por el contrario hacemos un include() y el archivo que tratamos de traer no se encuentra disponible, entonces lo que PHP nos mostrará es una señal de advertencia, un "warning", pero tratará de seguir ejecutando el programa. En resumen, cuando usas archivos con código de funciones (librerías) y los incluyes para usarlos desde otras páginas de la aplicación, la cosa quedaría así:

Tendríamos un archivo libreria.php como sigue

```
<?

//función de encabezado y colocación del titulo

function hacer_encabezado($titulo)

{

   $encabezado="<html>n<head>nt<title>$titulo</title>n</head>n";
```

```
   echo $encabezado;

}

?>
```

Por otra parte tendríamos nuestro script principal página.php (por ejemplo):

```
<?

include("libreria.php");

$titulo="Mi Web";

hacer_encabezado($titulo);

?>

<body>

El cuerpo de la página

</body>

</html>
```

Podemos incorporar todas las funciones creamos dentro de un mismo archivo pero resulta más ventajoso ir clasificándolas en distintos archivos por temática: Funciones de conexión a bases de datos, funciones comerciales, funciones generales, etc. Esto nos ayudara a poder localizarlas antes para corregirlas o modificarlas, nos permite también cargar únicamente el tipo de función que necesitamos para el script sin recargar éste en exceso además de reutilizar algunas de nuestras librerías para varios sitios webs distintos.

También puede resultar muy práctico el utilizar una nomenclatura sistemática a la hora de nombrarlas: Las funciones comerciales podrían ser llamadas com_loquesea, las de bases de datos bd_loquesea, las de tratamiento de archivos file_loquesea. Esto nos permitirá reconocerlas enseguida cuando leamos el script sin tener que recurrir a nuestra memoria para descubrir su utilidad.

No obstante, antes de lanzarnos a crear nuestra propia función, merece la pena echar un vistazo a la documentación de PHP para confirmar si dicha función ya existe o podemos aprovecharnos de alguna de las existentes para escribir menos código. Así, por ejemplo, existe una función llamada header que crea un encabezado HTML configurable lo cual nos evita tener que crearla nosotros mismos.

Ejemplo de función

Se trata de hacer una función que recibe un texto y lo escribe en la página con cada carácter separado por "-". Es decir, si recibe "hola" debe escribir "h-o-l-a" en la página web.

La manera de realizar esta función será recorrer el string, caracter a caracter, para imprimir cada uno de los caracteres, seguido de el signo "-". Recorreremos el string con un bucle for, desde el carater 0 hasta el número de caracteres total de la cadena.

El número de caracteres de una cadena se obtiene con la función predefinida en PHP strlen(), que recibe el string entre paréntesis y devuelve el número de los caracteres que tenga.

```
<html>
<head>
<title>funcion 1</title>
</head>
<body>
<?
function escribe_separa($cadena){
        for ($i=0;$i<strlen($cadena);$i++){
                echo $cadena[$i];
                        if ($i<strlen($cadena)-1)
                                echo "-";
                }
        }

escribe_separa ("hola");
echo "<p>";
escribe_separa ("Texto más largo, a ver lo que hace");
?>
</body>
</html>
```

La función que hemos creado se llama escribe_separa y recibe como parámetro la cadena que hay que escribir con el separador "-". El bucle for nos sirve para recorrer la cadena, desde el primer al último carácter. Luego, dentro del bucle, se imprime cada carácter separado del signo "-". El if que hay dentro del bucle for comprueba que el actual no sea el último carácter, porque en ese caso no habría que escribir el signo "-" (queremos conseguir "h-o-l-a" y si no estuviera el if obtendríamos "h-o-l-a-").

Paso de parámetros en funciones PHP

Los parámetros son los datos que reciben las funciones y que utilizan para realizar las operaciones de esa función. Una función puede recibir cualquier número de parámetros, incluso ninguno.

Si la función que estamos construyendo no necesita recibir ningún parámetro, al declararla, simplemente indicamos los paréntesis vacíos en la cabecera. Por ejemplo en la siguiente función mostramos la fecha del día de hoy. Para ello nos apoyamos en otra función incluida en PHP: date().

```
function fecha_hoy() {

      echo date('d/m/Y');

}
```

La intención de la anterior función es mostrar la fecha del día actual. Como siempre mostrará el día de hoy, no necesito pasarle ningún parámetro, siempre hará lo mismo. Las funciones que no requieren parámetros se las invoca indicando los paréntesis vacíos.

```
fecha_hoy();
```

Para implementar una función, en la cabecera, se definen los parámetros que va a recibir.

```
function f1 ($parametro1, $parametro2)
```

Así definimos una función llamada f1 que recibe dos parámetros. Como se puede observar, no se tiene que definir el tipo de datos de cada parámetro. Es decir, la función necesitará que le enviemos dos datos, pero no le importará que sean de un tipo u otro.

Los parámetros tienen validez durante la ejecución de la función. Como en un ámbito local a la función donde se están recibiendo. Cuando la función se termina, los parámetros dejan de existir.

Los parámetros se pasan por valor

El paso de parámetros en PHP se realiza por valor. "Por valor" es una manera típica de pasar parámetros en funciones, quiere decir que el cambio de un dato de un parámetro no actualiza el dato de la variable que se pasó a la función. Por ejemplo, cuando invocamos una función pasando una variable como parámetro, a pesar de que cambiemos el valor del parámetro dentro de la función, la variable original no se ve afectada por ese cambio. Por ejemplo:

```
function porvalor ($parametro1){

   $parametro1="hola";

   echo "<br>" . $parametro1; //imprime "hola"

}
```

```
$mivariable = "esto no cambia";

porvalor ($mivariable);

echo "<br>" . $mivariable; //imprime "esto no cambia"
```

Esta página tendrá como resultado:

hola
esto no cambia

Paso de parámetros por referencia

En contraposición al paso de parámetros por valor, está el paso de parámetros por referencia. En este último caso, el cambio del valor de un parámetro dentro de una función sí afecta al valor de la variable original.

Podemos pasar los parámetros por referencia si, en la declaración de la función, colocamos un "&" antes del parámetro.

```
function porreferencia(&$cadena) {
        $cadena = 'Si cambia';
}

$str = 'Esto es una cadena';

porreferencia ($str);

echo $str; // Imprime 'Si cambia'
```

Este script mostrará por pantalla 'Si cambia'.

Parámetros por defecto

Podemos definir valores por defecto para los parámetros. Los valores por defecto sirven para que los parámetros contengan un dato predefinido, con el que se inicializarán si no se le pasa ningún valor en la llamada de la función. Los valores por defecto se definen asignando un dato al parámetro al declararlo en la función.

```
function pordefecto ($parametro1="pepe";$parametro2=3)
```

Para la definición de función anterior, $parametro1 tiene como valor por defecto "pepe", mientras que $parametro2 tiene 3 como valor por defecto.

Si llamamos a la función sin indicar valores a los parámetros, estos tomarán los valores asignados por defecto:

```
pordefecto () // $parametro1 vale "pepe" y $parametro2 vale 3
```

Si llamamos a la función indicando un valor, este será tenido en cuenta para el primer parámetro.

```
pordefecto ("hola") // $parametro1 vale "hola" y $parametro2 vale 3
```

Retorno de valores en funciones PHP

Las funciones pueden, o no, retornar valores. Es decir, no es obligado que las funciones retornen valor alguno, solo se trata de una posibilidad, que encontrarás de mucha utilidad en el desarrollo en general.

Palabra "return"

Para retornar valores en funciones se utiliza la palabra "return", indicando a continuación el dato o variable que tienen que retornar.

```
function suma($valor1, $valor2) {

	return $valor1 + $valor2;

}
```

La anterior función realiza una operación de suma entre dos valores enviados por parámetro. Para invocarla debemos enviarle los dos valores que debe sumar. Cuando se ejecute la función recibiremos un valor como devolución y podremos hacer cualquier cosa con él. Por ejemplo, en el siguiente código estamos invocando a la función suma, enviando dos valores numéricos y almacenando el valor de devolución en una variable llamada "$resultado".

```
$resultado = suma(3, 6);
```

Una función puede perfectamente tener múltiples palabras return en su código. Sin embargo, aunque esto ocurra, debemos tener en cuenta que una función sólo podrá devolver un único valor. Entre otras cosas esto ocurrirá porque, cuando se usa el return, se termina la ejecución de la función devolviendo el dato indicado.

Por ejemplo:

```
function division($valor1, $valor2) {

	if($valor2 == 0) {

		return 'Error. Divide por cero';

	} else {

		return $valor1 / $valor2;

	}

}
```

En el código anterior, la función evalúa si $valor2 tiene el dato 0 (cero) y en ese caso se devuelve un mensaje "Error. Divide por cero". En caso que $valor2 no fuera un valor de cero, entonces realiza la operación de división y se devuelve el resultado.

En el código de la función anterior aparecen dos return, y solamente uno de ellos se ejecutará, dada la construcción del IF. La función, al encontrar un return se detiene en ese punto de ejecución. Es decir, después de ejecutar un return no se ejecutará ninguna otra línea de código siguiente.

Por ejemplo:

```
function cuadrado($valor) {

	return $valor * $valor;

	echo 'Esto nunca se ejecutará!!';

}
```

Debido al return, el código con la sentencia "echo" nunca se llegará a ejecutar.

Retornar múltiples valores en una función

Lo dicho anteriormente sobre que "una función devuelve un único valor" puede resultar una limitación a la hora de escribir funciones. Aunque no es del todo cierto, en realidad si es posible que las funciones puedan devolver varios valores distintos ya que podemos crear para ello una devolución en un array.

```
function numeros_pequenos()

{

   return array (0, 1, 2);

}

list ($zero, $one, $two) = small_numbers();
```

Con el array devuelto podremos hacer cualquier cosa. Acceder a sus casillas por separado, recorrerlo, etc. Después de esa operación, $zero valdrá 0, $one valdrá 1 y $two valdrá 2.

Además de arrays, también podemos devolver objetos y eso nos ayudará a retornar en las funciones todo tipo de estructuras complejas, con varios datos, solo devolviendo un objeto. Más adelante hablaremos sobre objetos en PHP, con ejemplos.

Condiciones IF

Cuando necesitamos que el programa, llegado a un cierto punto, tome una decisión sobre como seguir en PHP también contamos con el conjunto de instrucciones *if, else* y *elseif*. La estructura de base de este tipo de instrucciones es la siguiente:

```
if (condición)

{

    Instrucción 1;

    Instrucción 2;

    ...

}

else

{

    Instrucción A;

    Instrucción B;

    ...

}
```

Llegados a este punto, el programa verificará el cumplimiento o no de la condición. Si la condición es cierta las instrucciones 1 y 2 serán ejecutadas. De lo contrario (*else*), las instrucciones A y B serán llevadas a cabo.

Esta estructura de base puede complicarse un poco más si tenemos cuenta que no necesariamente todo es blanco o negro y que muchas posibilidades pueden darse. Es por ello que otras condiciones pueden plantearse dentro de la condición principal. Hablamos por lo tanto de condiciones anidadas que tendrían una estructura del siguiente tipo:

```
if (condición1)

{

    Instrucción 1;

    Instrucción 2;

    ...
```

```
}
else
{
    if (condición2)
    {
        Instrucción A;
        Instrucción B;
        ...
    }
    else
    {
        Instrucción X
        ...
    }
}
```

De este modo podríamos introducir tantas condiciones como queramos dentro de una condición principal.

De gran ayuda es la instrucción *elseif* que permite en una sola línea introducir una condición adicional. Este tipo de instrucción simplifica ligeramente la sintaxis que acabamos de ver:

```
if (condición1)
{
    Instrucción 1;
    Instrucción 2;
    ...
}
elseif (condición2)
{
```

```
    Instrucción A;
    Instrucción B;
    ...
}
else
{
    Instrucción X
    ...
}.
```

Control del flujo en PHP: Bucles

PHP propone varios tipos de bucle cada uno con características especificas:

Bucle while

El bucle más utilizado y el más sencillo. Lo usamos para ejecutar las instrucciones contenidas en su interior siempre y cuando la condición definida sea verdadera. La estructura sintáctica es la siguiente.

```
while (condición)

{

    instruccion1;

    instruccion2;

    ...

}
```

Un ejemplo sencillo es este bucle que aumenta el tamaño de la fuente en una unidad a cada nueva vuelta por el bucle:

```
<?

$size=1;

While ($size<=6)

{

    echo"<font size=$size>Tamaño $size</font><br>n";

    $size++;

}

?>
```

Debemos definir el valor de la variable que vamos a evaluar en la condición. En este caso le hemos atribuido un valor de 1 que corresponde a la letra más pequeña.

El paso siguiente es crear el bucle en el que imponemos la condición que la variable no exceda el valor de 6.

La instrucción a ejecutar será imprimir en nuestro documento un código HTML en el que la etiqueta *font* y el mensaje que contiene varían a medida que *$size* cambia su valor.

El siguiente paso es incrementar en una unidad el valor de $size. Esto se puede hacer con una expresión como la mostrada en el bucle ($size++) que en realidad es sinónima de:

```
$size=$size+1
```

El bucle while se suele utiliza cuando no se sabe exactamente cuantas iteraciones se deben realizar antes de acabar.

Ejemplo: recorrer una cadena hasta encontrar un carácter dado. Si lo encuentra, escribir su posición. Si no, escribir que no se ha encontrado.

```
<?

$cadena = "hola a todo el mundo";

//recorro la cadena hasta encontrar una "m"

$i=0;

while ($cadena[$i]!="m" && $i< strlen($cadena)){

        $i++; }

if ($i==strlen($cadena))

        echo "No se encuentra...";
else
        echo "Está en la posición $i";
?>
```

En este ejemplo se define una cadena con el valor "hola a todo el mundo". Posteriormente se recorre esa cadena hasta el final de la cadena o hasta encontrar el caracter "m", utilizando una variable $i que lleva la cuenta de los caracteres recorridos.

Al final del bucle while, si se salió porque se encontró el caracter "m", la variable $i valdrá un número menor que la longitud de la cadema. Si se salió por llegar al final de la cadena, la variable $i valdrá lo mismo que la longitud en caracteres de esa cadena. En el condicional simplemente se comprueba si $i vale o no lo mismo que la longitud de la cadena, mostrando los mensajes adecuados en cada caso.

Bucle do/while

La sintaxis es la siguiente:

```
do
```

```
{

    instruccion1;

    instruccion2;
    …
}
while (condición)
```

La diferencia con respecto a los bucles *while* es que este tipo de bucle evalúa la condición al final con lo que, incluso siendo falsa desde el principio, este bucle se ejecuta al menos una vez.