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, pueden observar cómo utilizamos 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 pueden ver, éste nos permite almacenar tablas y, a partir de una simple petición, visualizarlas con 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 número 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 "
" 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
```