Resumen PHP

Este es un resumen de lo visto en el manual de PHP de Aprende Web

http://aprende-web.net/php

1. Preparación

1.1. Definición.

PHP es un lenguaje para programar páginas de contenido dinámico. Este es un lenguaje del lado del servidor, el servidor lee el lenguaje php, y manda los datos transformados en lenguaje HTML.

El lenguaje PHP se incorpora al lenguaje normal de la página (HTML) mediante scripts.

Fue creado originalmente por Rasmus Lerdorf en 1995. Se ha ido ampliando con el tiempo, actualmente va por la versión 5, y trabaja conjuntamente con otros programas como son la base de datos MySQL y el servidor Apache.

Para diseñar páginas con PHP debemos saber previamente los lenguajes HTML y CSS con los que se crean las páginas web.

PHP permite realizar algunas acciones que no pueden realizarse en una página estática: recoger datos del usuario, trabajar con bases de datos, crear sesiones de usuarios, restringir páginas con contraseñas, creación de foros, etc.

1.2. Sofware necesario

1.2.1. Navegadores

Es conveniente tener instalados varios navegadores para ver los resultados de nuestro trabajo. A veces un mismo código no se ve igual en todos los navegadores, por lo que debemos tener los más usuales: Mozilla Firefox, Google Chrome, Opera, Safari y Internet Explorer.

Explicamos dónde encontrarlos y cómo instalarlos en la sección *Complementos / programas / navegadores: (http://aprende-web.net/comple/programas/programas1.php).*

En todo caso es necesario tener instalado el navegador Mozilla Firefox, ya que tiene una serie de complementos que nos ayudarán a crear nuestras páginas, tal como se explica en los manuales de HTML y CSS.

1.2.2. Editor de textos

Aunque el editor de textos más simple es el block de notas de Windows, aquí recomendamos el **HTML-Kit**. Es un editor de textos bastante completo y de sofware libre, En la sección *(http://aprende-web.net/comple/programas/programas2.php) Complementos / programas /HTML-Kit* se explica donde encontrarlo y como instalarlo; y donde encontrar el plugin para verlo en español.

Aunque aquí se recomienda este programa, puede usarse cualquier otro editor de textos.

1.2.3. Descargar XAMPP

El sofware visto hasta ahora es el mismo que para crear páginas web estáticas. Para crear páginas con PHP debemos convertir nuestro ordenador en un servidor local, de esta manera podremos ver los resultados de nuestro trabajo antes de subirlo a la red.

Tener un servidor local en el ordenador requiere tener instalados varios programas, los cuales trabajan conjuntamente. Hasta no hace muchos años debían instalarse por separado y configurarlos después.

Por suerte hoy en día tenemos el paquete integrado **XAMPP** que contiene ya todos los programas y los instala y configura de una manera sencilla.

Puedes descargarte el paquete integrado XAMPP en la siguiente página:

http://www.apachefriends.org/es/xampp.html

Desde ahí eliges la versión de XAMPP que coincide con tu sistema operativo (windows, linux, etc.). Esto te lleva a otra página desde donde puedes descargar la versión. Ésta está en dos formatos "Installer": para instalar como un programa o "Zip" para descargarse los archivos. Elegimos la primera, ya que nos lo instalará como si fuera un programa.

Una vez descargado lo abrimos y lo instalamos como cualquier programa.

Una vez instalado, al abrir el programa nos encontramos con un panel de control. Deben estar marcados ahí los programas "Apache" y "MySQL" para que XAMPP funcione correctamente.

1.3. El servidor local

Una vez instalado XAMPP tenemos ya un servidor local. Ahora debemos saber cómo utilizarlo.

En primer lugar debemos tener siempre el panel de control abierto cuando trabajemos con el servidor local, y con los dos primeros programas (apache y MySQL) funcionando.

Para comprobar que el servidor local funciona abrimos cualquier navegador y en la barra de direcciones escribimos *http://localhost*. Se nos abre la página de inicio de XAMPP, donde la primera vez elegimos el idioma, y nos lleva a la página de inicio en nuestro idioma.

La instalación de XAMPP ha añadido una serie de carpetas y archivos a nuestro ordenador. En Windows estos están en el directorio *C / XAMPP*. Dentro de esta carpeta buscamos otra carpeta llamada *htdocs*. Es en esta carpeta donde guardaremos todos los sitios web que hagamos con PHP como subcarpetas de la misma.

Dentro de la carpeta *htdocs* vemos el archivo *index*. Éste es el que por defecto se abre al ir con el navegador a la página "localhost". No debemos cambiarlo, ya que es el que proporciona la información de XAMPP.

Si lo que queremos es crear uno o varios sitios web en php, lo que haremos será crear dentro de la carpeta *htdocs* una o varias subcarpetas que se correspondan a cada sitio. Por ejemplo, creamos dentro de *htdocs* una subcarpeta llamada *misitio*. Dentro de ella estarán todos los archivos y carpetas de este sitio, entre ellos la página principal, llamada *index.php*. Para verlos en el navegador debemos escribir la ruta a la página desde la dirección *http://localhost.* En este ejemplo escribiremos *http://localhost/misitio* para visualizar la página principal o index.php del sitio.

1.4. La carpeta htdocs

Esta carpeta será el directorio de trabajo, donde se guardarán todos los sitios y archivos que creemos en PHP.

Dentro de esta carpeta pueden crearse subcarpetas donde guardemos pruebas, proyectos de sitios, carpetas raíz de sitios, etc. Todo el trabajo con PHP debe guardarse dentro de esta

carpeta. De esta manera lo podremos ver el navegador utilizando la ruta *http://localhost/ (..seguir ruta en htdocs ..)* .

Tal vez nos resulte más cómodo acceder a la carpeta "htdocs" desde la carpeta de "mis documentos" o desde otra más habitual. Para ello no tenemos más que crear un acceso directo a la carpeta "htdocs".

Para escribir los nombres de los archivos que creemos con PHP debemos seguir las mismas reglas descritas para los archivos en HTML. Recordemos que éstos son básicamente las siguientes:

- Texto en minúsculas: Todo el texto del nombre de archivo se escribirá en minúsculas.
- No poner acentos: Escribiremos todas las palabras sin acento, aunque los tengan.
- No empezar por un número: El primer carácter o signo que escribamos no debe ser un número, pues algunos programas podrían interpretar mal el nombre del archivo
- No utilizar otros caracteres que no sean letras o números: no utilizar barras, paréntesis, puntos, dos puntos, comas, comillas, signo más, etc.
- Separar las palabras con guión normal - o con guión bajo _ : Esto nos asegura un sólo espacio entre palabras.

2. Sintaxis

2.1. Scripts en PHP

El código PHP está incrustado entre medio del código HTML. Para distinguirlo lo escribimos siempre dentro de las etiquetas:

```
<?php ...código php... ?>
```

La etiqueta `<?php` indicará que empezamos a escribir en PHP, y la etiqueta ?> indica que acaba el código PHP

También podemos usar las etiquetas:

```
<script language="php">.... codigo php ... </script>
```

Hay otras etiquetas, pero es posible que no funcionen bien en todas las versiones de PHP, estas son:

```
<? .... codigo php ... ?>
<% .... codigo php ... %>
```

2.2. Escribir en la página

La instrucción `echo` permite visualizar en la página lo que escribamos detrás. Para visualizar un texto debemos escribirlo entre comillas:

```
<?php
echo "Hola mundo";
?>
```

Dentro de las comillas podemos poner no sólo texto, sino código HTML (con sus etiquetas):

```
<?php
echo "<p>Hola mundo</p>";
```

```
?>
```

La instrucción `print` funciona exactamente igual que la instrucción `echo`, podemos usar una u otra indistintamente:

```
<?php
print "<p>Hola mundo</p>";
?>
```

2.3. Normas de escritura

Al escribir el código tenemos normas similares a los lenguajes HTML, CSS o Javascript, es decir: no se tiene en cuenta más de un espacio en blanco, los saltos de línea o las tabulaciones. Todos ellas son considerados como un sólo espacio en blanco.

Cada instrucción o sentencia de PHP debe acabar por un punto y coma (;). El cambio de línea no supone un cambio de instrucción.

El código Sí distingue entre mayúsculas y minúsculas, considerando elementos distintos a una palabra en minúsculas y su correspondiente en mayúsculas.

2.4. Variables

2.4.1. Definición

Una variable es un elemento en el que se puede almacenar un valor o dato. Para ello basta con asignarle a una variable un valor.

En PHP todas las variables van precedidas del signo dólar $. Este signo delante de cualquier texto indica que lo que viene detrás es una variable.

2.4.2. Asignar valor a una variable

Las variables pueden guardar valores, datos, u otros elementos. Al crearlas es conveniente asignarles un valor o dato. Esto se hace mediante el signo = (igual).

```
$texto = "Hola mundo";
```

Escribir el nombre de la variable es lo mismo que escribir su valor o dato almacenado. En el ejemplo anterior las dos instrucciones siguientes nos darán el mismo resultado:

```
echo $texto;
echo "Hola mundo";
```

Si escribimos el nombre de una variable dentro de una cadena de texto, no veremos el nombre de la variable, sino su valor, por ejemplo:

```
echo "<p>$texto, bienvenido a mi página.</p>";
```

Dará como resultado:

Hola mundo, bienvenido a mi página.

2.4.3. Tipos de variables

Dependiendo del valor que almacenen las variables pueden ser de distintos tipos:

- **Números enteros (integer):** Números enteros sin decimales.

  ```
  num1 = 43;
  ```

- **Números Reales (real):** Números reales con decimales.

  ```
  $num2 = 12.56
  ```

- **Cadenas de texto (string):** Texto escrito con caracteres alfanuméricos. Van siempre delimitadas por comillas al principio y al final de la cadena.

  ```
  $texto = "Esta variable es un texto."
  ```

- **Arrays o listas de elementos: (array):** Conjunto de elementos que están agrupados bajo una única variable, y forman una lista.

  ```
  $array = array("primavera","verano","otoño","invierno");
  ```

- **Objetos (object):** Otros elementos más complejos que pueden crearse con el lenguaje PHP

2.4.4. Cambiar el tipo de variable

Las variables cambian automáticamente de tipo al asignarles un nuevo valor de otro tipo.

Podemos forzar también a una variable a que cambie de tipo sin cambiarle el valor. Utilizamos la instrucción:

```
$variable = (tipo_variable) $variable;
```

En `tipo_variable` podemos escribir las siguientes palabras:

- `int, integer` : Convertir a entero.
- `real, double, float` : convertir a número real.
- `string` : Convertir a cadena de texto.
- `array` : Convertir a array.
- `object` : Convertir a objeto.

Donde hemos puesto más de una palabra podemos elegir cualquiera de ellas. En este caso la palabra la pondremos dentro del paréntesis, y sin comillas.

Podemos usar también la instrucción:

```
setType($variable,"nuevo_tipo");
```

En este caso en `"nuevo_tipo"` pondremos cualquiera de las palabras anteriores entre comillas.

2.4.5. Caracteres de escape

Al igual que en otros lenguajes de programación, hay una serie de caracteres que se utilizan en la propia programación, y que no pueden escribirse directamente para verlos en la pantalla.

Si queremos mostrar alguno de estos caracteres utilizaremos los caracteres de escape.

Todo carácter de escape consta de una barra inclinada inversa `\` seguida del carácter a mostrar o de una referencia a lo que se quiere ejecutar.

El más usado es `\$` que muestra el nombre de la variable con el signo dólar delante. por ejemplo:

```
$texto = "Esto es una variable";
echo "la variable \$texto tiene el valor: $texto";
```

El código anterior lo veremos en el navegador así:

la variable $texto tiene el valor: Esto es una variable

Los caracteres de escape más comunes son:

- `\n` : nueva línea.
- `\t` : tabulación.
- `\\` : barra invertida.
- `\$` : signo dólar.
- `\"` : comillas dobles.
- `\'`: comillas simples.
- `\&` : signo ampersand.
- `\?` : signo de interrogación.

Los caracteres de escape de nueva línea y de tabulación, sólo se producen en el código fuente de la página, es decir en el código HTML, y no en la página en sí.

2.4.6. Comentarios

Lo comentarios son anotaciones que hace el programador al margen del código. No forman parte del código, pero ayudan a entender lo que se está haciendo. Su sintaxis es la misma que en javascript. Tenemos dos tipos de comentarios: los de una línea y los de varias líneas.

Los comentarios de una línea empiezan con dos barras inclinadas: //. Todo lo que se ponga en la línea después de las dos barras inclinadas será el comentario:

```
$a = "hola"; //definir la variable a
```

Después de la instrucción hemos puesto un comentario de una línea para indicar lo que estamos haciendo.

Los comentarios de varias líneas empiezan por `/*` y terminan por `*/`. todo lo que escribamos entre estos signos será un comentario:

```
/*página de inicio
Creada por Anyelguti.*/
```

2.4.7. Concatenar variables

Esto consiste en que el valor de una variable se convierta en otra variable. Para ello utilizamos dos signos dólar seguidos. Vemos el siguiente ejemplo:

```
$v="tema1"; //variable inicial
$$v="El Quijote"; //variable concatenada
echo $tema1; // nueva variable: $tema1 = "El Quijote"
```

La nueva variable (`$tema1`) tiene como nombre del valor de la variable inicial, y como valor el valor de la variable concatenada (`"El Quijote"`).

2.4.8. Variables vinculadas.

Si hacemos una copia de una variable en otra, las dos tendrán el mismo valor. Pero si cambiamos después el valor de la primera, la segunda seguirá con el valor antiguo. Estas son dos variables iguales, pero no están vinculadas:

```
$v1="rojo"; //Primera variable $v1
$v2=$v1; //Segunda variable $v2
$v1="azul"; //Cambiamos el valor de la primera variable
echo $v1; //texto en pantalla : azul
```

```
echo $v2; //texto en pantalla: rojo
```

Para vincular las variables de forma que al cambiar la primera cambie también la segunda, al igualarlas pondremos un signo ampersand (&) delante de la primera variable (la que va detrás del igual): `$v2 = &$v1;`. El ejemplo anterior quedará así:

```
$v1="rojo"; //Primera variable $v1
$v2=&$v1; //Segunda variable $v2
$v1="azul"; //Cambiamos el valor de la primera variable
echo $v1; //texto en pantalla : azul
echo $v2; //texto en pantalla: azul
```

2.5. Constantes

2.5.1. Definición

Una constante es un elemento parecido a la variable, pero con la diferencia de que una vez que se han creado no puede cambiarse su valor. Las constantes se expresan sin el signo dólar ($).

2.5.2. Crear una constante

Para crear una constante utilizamos la función predeterminada `define()`:

```
define("constante1","una constante")
```

Dentro del paréntesis pondremos primero el nombre de la constante (`"constante1"`), y después separado por una coma su valor (`"una constante"`).

Después podemos sacarlas en pantalla igual que las variables:

```
echo constante1;
```

Sin embargo algunas constantes pueden resultar problemáticas, cuando su nombre coincide con el de algún elemento o palabra clave de PHP. En ese caso emplearemos la función `constant("nombre_constante"),` para sacarla en pantalla.

```
define("echo","instrucción echo");
echo constant("echo");
```

En este ejemplo al utilizar una palabra clave para definir la constante, debemos de sacarla en pantalla mediante la función `constant`.

2.6. Operadores aritméticos

Los operadores, como su nombre indica, nos permiten hacer operaciones con las variables, modificarlas, sumar, restar, comprobar si son iguales o distintas, etc.

Los operadores aritméticos nos permiten hacer operaciones aritméticas con las variables, estas deben ser de tipo "numero" aunque algunas operaciones pueden hacerse también con variables de tipo "texto".

2.6.1. Operador de asignación.

Es el más simple y ya lo hemos visto, consta del signo igual = y sirve para asignar o cambiar el valor de una variable.

2.6.2. Operador punto.

El operador punto (se escribe como un punto "."). Concatena dos o más variables. Se usa preferentemente con las variables de texto, aunque puede usarse con cualquier variable.

2.6.3. Operadores aritméticos clásicos.

Los operadores aritméticos clásicos son la suma + , la resta - , la multiplicación * y la división / . A estos hay que añadir el operador módulo % cuyo resultado es el resto de la división entre números enteros y su signo es el del porcentaje (%).

Operamos normalmente guardando el resultado en otra variable:

```
$num3=$num1+$num2;
```

2.6.4. Operadores de incremento.

Los operadores de incremento, aumentan o disminuyen en un número entero el valor de la variable a la que afectan.

Pueden utilizarse tanto con número enteros como reales (con decimales).

Existe un operador de incremento ++ y un operador de decremento -- los cuales aumentan o disminuyen respectivamente en una unidad el valor de la variable a la que se aplican.

Podemos ponerlos antes o después de la variable, (++$n o $n++). El resultado puede ser distinto, ya que si está unido a otra operación, al ponerlo delante primero se incrementa y luego se opera, mientras que si se pone detrás primero se opera, y luego se incrementa.

2.6.5. Operadores de asignación compuestos.

El operador consta de dos partes, la primera es una operación aritmética y la segunda el signo igual (`$n1 += $n2`). Se realiza la operación entre los dos números y el resultado es el nuevo valor del primero de ellos:

`$n1 += $n2;` es lo mismo que poner `$n1 = $n1 + $n2`

Los operadores disponibles son los siguientes:

- **+=** : `$a += $b; // igual que $a = $a + $b;`
- **-=** : `$a -= $b; // igual que $a = $a - $b;`
- ***=** : `$a *= $b; // igual que $a = $a * $b;`
- **/=** : `$a /= $b; // igual que $a = $a / $b;`
- **%=** : `$a %= $b; // igual que $a = $a % $b;`

2.6.6. Operadores condicionales.

Los operadores condicionales comprueban si se cumple o no una condición. Se usan sobre todo en las estructuras condicionales (que veremos en el siguiente tema), aunque también pueden usarse fuera de ellas.

Si la condición se cumple devuelven el número 1 (que indica que es verdadero), y si no se cumple devuelven una cadena vacía (que indica que es falso).

Tenemos los siguientes operadores condicionales:

Operadores condicionales

Nombre	Signo	Descripción	Ejemplo
igual	**==**	Devuelve 1 si los dos elementos son iguales	`$n1 == $n2`
idéntico	**===**	Devuelve 1 si los dos elementos son iguales, pero además los datos deben ser del mismo tipo.	`$n1 === $n2`
No igual	**!=**	Devuelve 1 si los dos elementos son distintos	`$n1 != $n2`
No idéntico	**!==**	Devuelve 1 si los dos elementos son distintos, pero además los datos deben ser del mismo tipo	`$n1 !== $n2`
Mayor que	**>**	Devuelve 1 si el primer valor es mayor que el segundo.	`$n1 > $n2`
Menor que	**<**	Devuelve 1 si el primer valor es menor que el segundo.	`$n1 == $n2`
Mayor o igual que	**>=**	Devuelve 1 si el primer valor es mayor o igual que el segundo.	`$n1 >= $n2`
Menor o igual que	**<=**	Devuelve 1 si el primer valor es mayor o igual que el segundo.	`$n1 <= $n2`

2.6.7. Operadores lógicos

Los operadores lógicos se emplean en combinación con los operadores condicionales y permiten comprobar si se cumplen o no varias condiciones al mismo tiempo.

El resultado es el mismo que para los condicionales, es decir, 1 si el resultado es verdadero, o una cadena vacía si el resultado es falso.

Tenemos los siguientes operadores lógicos.

Operadores lógicos

Nombre	Signo	Descripción	Ejemplo
AND	**and**	Devuelve 1 sólo si las dos condiciones se cumplen, si no es así devuelve una cadena vacía.	`$n1 != $n2 and $n1 != $n3`
OR	**or**	Devuelve 1 si una de las dos condiciones se cumple o si se cumplen las dos. Sólo si las dos no se cumplen devuelve cadena vacía.	`$n1 == $n2 or $n1 == $n3`
XOR	**xor**	Devuelve 1 sólo cuando una de las dos condiciones se cumple. Si no se cumple ninguna o se cumplen las dos devuelve cadena vacía.	`$n1 == $n2 xor $n1 ==$n3`
AND (2)	&&	Exactamente igual que `and`	`$n1 != $n2 && $n1 != $n3`
OR (2)	||	Exactamente igual que `or`	`$n1 == $n2 || $n1 == $n3`
Negación	!	Cambia el valor del elemento al que se le aplica es decir, si `$n1` es verdadero (1) lo cambia a falso (""), y viceversa	`!$n1`

Los operadores condicionales y los operadores lógicos se emplean mayoritariamente con las estructuras condicionales que veremos en el siguiente tema.

3. Estructuras

3.1. Introducción

Las estructuras son trozos de código que conforman un bloque definido y que, o definen un objeto, o realizan una tarea determinada.

Tenemos dos tipos de estructuras, las que definen objetos complejos y las estructuras de control. Del primer tipo tenemos los arrays y las funciones. Y del segundo tipo las estructuras condicionales y los bucles. Veremos todo esto a lo largo de este tema.

3.2. Arrays indexados

Un array es una variable en la que podemos guardar varios datos o variables. Éstos forman una lista con un único nombre. Cada elemento de la lista o array es identificado por su posición o por un nombre que podemos darle.

Hay dos tipos de arrays, los arrays indexados, que son simples listas de elementos, y los arrays asociativos, en los que a cada elemento de la lista se le asigna un nombre o clave.

En este apartado veremos los arrays indexados, que son los más sencillos.

3.2.1. Definir un array indexado

Para definir un array utilizamos la instrucción `aray()`. Para lo arrays indexados, dentro del paréntesis se escriben los elementos del array separados por comas. Éstos pueden ser valores o variables.

```
$miarray = ("Juan",,32,17.5,true,$v1)
```

Dentro del array podemos guardar todo tipo de datos, tanto cadenas como números enteros o decimales, valores booleanos o no poner nada. En un mismo array se pueden guardar distintos tipos de datos.

3.2.2. Leer un array indexado.

En los arrays indexados a cada elemento le corresponde un número. Empezamos a contar por el cero (0), al cual le corresponde el primer elemento, al segundo le corresponde el 1, y así sucesivamente.

Para buscar un elemento de un array utilizaremos su número con el siguiente código:

```
$miarray[num];
```

Donde `$miarray` es el array y `num` es el número del elemento.

Veamos un ejemplo:

```
$est=array("Primavera","Verano","Otoño","Invierno"); //definir el array
echo "$est[0], $est[1], $est[2], $est[3]"; //mostrar elementos en pantalla
```

El resultado del código anterior nos dará algo parecido a lo siguiente:

Primavera, Verano, Otoño, Invierno

3.2.3. Definir un array por sus elementos

Podemos definir un array definiendo directamente sus elementos:

```
$turnos[0]="mañana";
$turnos[1]="tarde";
$turnos[2]="noche";
```

Esto es lo mismo que escribir:

```
$turnos= array("mañana","tarde","noche");
```

Si al definir un elemento no hemos definido los anteriores, los anteriores no existen, pero para leerlo debemos usar el número que le hemos dado.

```
$miaray[5]="hola"
```

Este es un array de un sólo elemento, pero para poder leerlo escribiremos:

```
echo miarray[5]
```

3.2.4. Redefinir elementos

Para darle otro valor a un elemento de un array basta con definirlo de nuevo. Por ejemplo:

```
$verduras=array("lechuga","tomate","cebolla");
$verduras[0]="pepino";
```

Hemos cambiado el primer elemento del array al redefinir su contenido.

3.2.5. Contar los elementos de un array

Para contar los elementos que contiene un array utilizaremos la instrucción `count($miarray)`, donde `$miarray` es la variable que contiene el array.

Podemos usar también la instrucción `sizeof($miarray)`, de la misma manera que la anterior y con idéntico resultado.

3.3. Arrays asociativos

En un array asociativo cada elemento del array se compone de un valor y una clave. El valor será el elemento en sí, mientras que la clave será el índice con el que llamamos al elemento

3.3.1. Definir arrays asociativos

Para definir una array asociativo se define en cada elemento tanto su valor como su clave. Utilizamos el siguiente código:

```
$miarray=array("clave1"=>"valor1","clave2"=>"valor2" ... "claveN"=>"valorN");
```

Cada elemento se compone de dos partes, la clave y el valor. Escribimos primero la clave y luego el valor separados por los signos `=>` .

```
"clave"=>"valor"
```

3.3.2. Leer elementos de arrays asociativos

Para leer elementos en array asociativos nos referiremos al elemento por su clave, la cual cumple la misma función que el índice o número en los arrays indexados.

```
echo $miarray[clave];
```

Hemos escrito sin comillas la clave dentro del corchete. Podemos escribirla con o sin comillas, es indiferente, sin embargo si la escribimos dentro de un texto con comillas la pondremos sin comillas.

```
echo "valor : $miarray[clave]";
```

3.3.3. Recorrer los elementos de un array asociativo:

No siempre se conocen todas las claves, o a veces queremos sacar en pantalla las claves, o puede ser que queramos recorrer los elementos de un array con sus claves y valores.

Para ello disponemos de la función `each()`

Por ejemplo, tenemos el siguiente array:

```
$cap=array("España"=>"Madrid","Portugal"=>"Lisboa","Italia"=>"Roma");
```

Para saber cuál es el primer elemento utilizamos la función `each()`. En primer lugar hacemos:

```
$verCap=each($cap);
```

Obtenemos en `$verCap` un array con los siguientes elementos.

- **`$verCap[0];`** *`// muestra la clave del elemento.`*
- **`$verCap[1];`** *`// muestra el valor del elemento.`*
- **`$verCap["key"];`** *`// muestra la clave del elemento.`*
- **`$verCap["value"];`** *`// muestra el valor del elemento.`*

Este es un array indexado y asociativo al mismo tiempo. Para ver la clave podemos usar tanto `$verCap[0]` como `$verCap["key"]`, y para ver el valor podemos usar `$verCap[1];` o también `$verCap["value"]`.

La primera vez que usamos esta función nos dará el valor y la clave del primer elemento, por ejemplo:

```
echo "La capital de $verCap[0] es $verCap[1].";
```

Lo cual nos dará en pantalla un resultado parecido a esto:

La capital de España es Madrid.

La función `each()` tiene un puntero interno que se desplaza al siguiente elemento del array después de usarla, por lo que para ver el siguiente elemento la utilizamos otra vez: Repetimos en el ejemplo las dos últimas líneas:

```
$verCap=each($cap);
echo "La capital de $verCap[0] es $verCap[1].";
```

Ahora el resultado en pantalla nos dará el siguiente texto:

La capital de Portugal es Lisboa.

El puntero interno se ha movido y nos muestra el siguiente elemento. Podemos recorrer de esta forma todos los elementos del array y extraer sus claves y valores.

3.3.4. Control del puntero interno

Para controlar el puntero interno y poder mostrar los elementos que queramos tenemos una serie de funciones. Basta con aplicarlas de la siguiente manera:

- **`reset($verCap);`** : Lleva el puntero a la primera posición.
- **`end($verCap);`** : Lleva el puntero a la última posición.
- **`next($verCap);`** : Lleva el puntero al elemento siguiente.
- **`prev($verCap);`** : Lleva el puntero al elemento anterior.
- **`current($verCap);`** : Lleva el puntero al elemento actual.

Si aplicamos estas funciones no debemos redefinir la función `$verCap=each($cap);` ya que el puntero ya se ha movido, y lo único que haríamos es moverlo otra vez una posición hacia adelante.

3.4. Funciones

3.4.1. Definición

La ejecución del código en PHP se realiza de manera secuencial. Es decir, se va ejecutando en el mismo orden en el que está escrito. Esto cambia con las funciones, ya que son trozos de código que no se ejecutan cuando se leen, sino que se guardan y se pueden ejecutar más tarde con una llamada a la función desde cualquier parte del código.

Podemos llamar a la función desde cualquier parte posterior del código y tantas veces como haga falta.

Tenemos dos tipos de funciones, las que vienen ya predefinidas en PHP y las que podemos crear nosotros o "personalizadas". Las funciones predefinidas ya están hechas, y sólo tenemos que llamarlas para que realicen una tarea. Las "personalizadas" debemos definirlas primero e indicar qué tareas queremos que realicen, y además sólo pueden usarse en la misma página.

Ya hemos visto algunas de estas funciones predefinidas en PHP (aunque a algunas les hemos llamado "instrucción"). Para llamarlas y que se ejecuten se escribe su nombre seguido de un paréntesis. Dentro del paréntesis a veces pondremos una serie de datos que necesita la función para realizar su tarea, son los argumentos. A veces una función no necesita argumentos, en ese caso se pone el paréntesis vacío.

A las funciones definidas por nosotros o "personalizadas" se les llama de la misma manera.

Las funciones son objetos, por lo que podemos guardarlas en variables. En este caso escribiremos como valor de la variable el nombre de la función sin paréntesis ni argumentos. Por ejemplo en una función llmada `miFuncion` la guardamos en una variable:

```
$funcion1=miFuncion;
```

Sin embargo, si lo que queremos guardar es el resultado obtenido después de ejecutar el código de la función, en el valor de la variable pondremos la llamada a la función, con los argumentos que necesite:

```
$resultado=miFuncion($a,$b);
```

3.4.2. Definir una función

Para crear o definir una función utilizaremos el siguiente código:

```
function nombreFuncion($argumento1,$argumento2) {
      // ...instrucciones de la función ...//
      return $resultado;
      }
```

El código se compone de los siguientes elementos:

- **`function`** : La palabra clave `function`
- **`nombreFuncion`** : Seguimos dándole un nombre a la función. Podemos poner cualquier palabra siempre que no sea palabra reservada.
- **`($argumentos)`** : Después del nombre ponemos un paréntesis, y dentro los argumentos, o datos necesarios para realizar la tarea. Si no hacen falta argumentos se pone el paréntesis vacío.
- **`{ //código// }`** : Escribimos la llave de apertura `{` y después el código PHP que realiza la tarea. Éste normalmente ocupa varias líneas. Una vez escrito el código cerramos el bloque con la llave de cierre `}` .
- **`return $resultado`** : La función puede realizar una tarea directamente o devolver un resultado para utilizarlo más tarde (función con retorno). En este último caso devolvemos el resultado al código poniendo como última línea en la función la palabra clave `return` seguida de la variable que contiene el resultado.

3.4.3. Funciones y variables:

Variables globales y locales

Las variables tienen un ámbito o zona en donde son válidas. Una variable creada dentro del flujo normal del código será válida dentro del flujo normal del código, pero no dentro de una función.

Las variables que se crean dentro de una función funcionan solamente dentro de la misma, no pudiendo usarse fuera de la función.

Esta última clase de variable nos da las variables locales, ya que éstas sólo pueden utilizarse en el ámbito en que han sido creadas.

En principio podemos pasar como argumento las variables que están fuera de la función y que vayamos a utilizar dentro.

Para las variables creadas dentro de la función, las podemos sacar fuera mediante la instrucción `return` (si son varias podemos sacarlas en un array).

Sin embargo estos métodos son limitados, por lo que lo mejor es declarar las variables como globales.

Declarar una variable global

Tenemos varios métodos para declarar una variable como global. PHP dispone de varios arrays internos, a los cuales podemos añadirles elementos. Uno de ellos es el array `$GLOBALS` que guarda todas las variables globales que hay en la página. Mediante este array transformamos una variable local en global:

Dentro del código de la función pondremos:

```
$v1 = $GLOBALS["$v1"];
```

Ahora la variable `$v1` que estaba definida fuera de la función puede usare ahora dentro de la función.

En este método hay que declarar las variables una a una. Hay otro método que es usar la palabra reservada `global` seguido de los nombres de las variables que queremos transformar en globales separadas por comas. Esta línea la pondremos al principio de la función. Ejemplo:

```
$nombre="Juan";
$apellido="Martínez";
function saludo() {
   global $nombre, $apellido;
   $recuerdos = "Recuerdos a  $nombre $apellido"
   echo $recuerdos;
   }
saludo();
```

El código anterior dará como resultado el siguiente texto:

Recuerdos a Juan Martínez

3.4.4. Argumentos

Los argumentos son los datos que le pasamos a la función para que ésta realice su tarea.

Al crear la función definimos los argumentos que se deben pasar, esto se pone dentro del paréntesis.

```
function nombreFunción($arg1,$arg2) { //...
```

Luego al llamar a la función debemos pasarle los argumentos que necesite. Las variables que pasamos al llamar a la función no tienen porqué ser las mismas, pero sí tienen que estar en el mismo orden:

```
$resultado=nombreFuncion($var1,$var2);
```

Siguiendo este ejemplo, dentro de la función la variable `$var1` pasara a ocupar el lugar de `$arg1`, y `$var2` ocupará el lugar de `$arg2`

Al definir la función podemos definir también cómo se pasan los argumentos, hay varia formas:

Pasar argumentos por valor

Es la forma vista hasta ahora:

```
function nombreFunción($arg1,$arg2) { //...
```

Una vez pasados los argumentos, podemos cambiarles el valor dentro de la función, pero las variables que hemos utilizado fuera de la función, seguirán teniendo el mismo valor.

Pasar argumentos por referencia

Aquí al cambiar el valor del argumento dentro de la función, cambiamos también el valor de la variable utilizada fuera de la función, para ello el argumento lleva un signo ampersand (`&`) delante:

```
function nombreFunción(&$arg1,&$arg2) { //...
```

Pasar argumentos por defecto

Al definir el argumento le asignamos un valor predeterminado, de manera que si al llamar a la función no se indica el argumento, éste tomará el valor por defecto.

Para ello al definir el argumento se le asigna un valor.

```
function nombreFunción($arg1=0,$arg2=0) { //...
```

Retorno de valores

Mediante la instrucción `return` al final del código de la función devolvemos el resultado de la función al flujo de la página.

La instrucción `return` no permite devolver más de un valor, sin embargo podemos hacer que devuelva más de un valor poniéndolos todos en un array:

```
return array($resultado1,$resultado2);
```

Luego al llamar a la función lo hacemos mediante la instrucción `list()` de la siguiente manera:

```
list($var1,$var2)=nombreFuncion();
```

La variable `$var1` recogerá el valor de `$resultado1` y la variable `$var2` recogerá el valor de `$resultado2`.

3.5. Estructura condicional if

En una estructura condicional se le indica al programa una condición, si ésta se cumple se ejecuta un determinado código, y si no se cumple puede ejecutarse otro código diferente o no ejecutarse nada.

La estructura condicional más sencilla es la estructura `if` y tiene la siguiente sintaxis:

```
if ( /*condición*/ ) {
   /*código a ejecutar si se cumple la condición*/
   }
   else {
   /*código a ejecutar si la condición no se cumple*/
   }
```

Veamos más detenidamente el código:

- **`if ()`** : Escribimos la palabra reservada `if` seguida de un paréntesis, y dentro del paréntesis escribimos la condición.
- **`condicion`** : Como condición pondremos un código. PHP lo convierte en booleano, y comprueba si es verdadero o falso. Normalmente se pone una sentencia con un operador condicional o con operadores condicionales y lógicos. En este ejemplo se comprueba que el array tenga más de 5 elementos:

  ```
  if (count($array)>5)
  ```

- **`{ /*condición*/ }`** : Escribimos después el código que se ejecutará cuando la condición se cumpla, éste se pone dentro de las llaves. Ejemplo:

  ```
  {echo "El array tiene más de 5 elementos";}
  ```

- **`else { /*condición*/ }`** : Esta parte es opcional, y muestra el código en caso de que la condición no se cumpla. Ponemos en primer lugar la palabra reservada `else` y después, dentro de las llaves el código que se ejecutará si la condición no se cumple:

  ```
  else { echo "El array tiene 5 elementos o menos";}
  ```

Como condición se pone normalmente una sentencia con operadores condicionales o lógicos, pero también podemos poner una variable. En este caso se comprueba que la variable exise y que es distinta de 0 o de cadena vacía.

3.5.1. Forma elseif

Si queremos elegir entre más de una opción, usaremos varias estructuras if de la siguiente manera:

```
if ( /*primera condición*/ )
   { /*código a ejecutar si se cumple la primera condición*/ }
elseif ( /*segunda condición*/ )
   { /*código para la segunda condición*/ }
elseif ( /*tercera condición*/ )
   { /*código para la tercera condición*/ }
   .....
else { /*código cuando no se cumple ninguna de las condiciones anteriores*/ }
```

Lo que hacemos es anidar varias estructuras if de forma que al no cumplirse la primera condición (`else`) vamos al segundo if. Si no se cumple la segunda vamos a la tercera, y así sucesivamente, hasta que al final si no se cumple ninguna vamos al último bloque: `else`.

Podemos escribir tanto `elseif` como `else if`, es decir las dos palabras separadas por un espacio o juntas, el resultado será el mismo.

3.6. Estructura switch

Cuando queremos que el programa ejecute distintas acciones dependiendo del valor que tome una variable, podemos usar también la estructura switch. Eta tiene la siguiente sintaxis:

```
switch ($variable) {
    case "valor1":
        /*código a ejecutar para el valor1;*/
        break;
    case "valor2":
        /*código a ejecutar para el valor2;*/
        break;
    case "valor3":
        /*código a ejecutar para el valor3;*/
        break;
    ..............
    default:
        /*código si la variable no tiene ningún valor de los anteriores;*/
    }
```

En primer lugar se escribe la palabra clave `switch` y después, entre paréntesis la variable a la que afecta. El resto del código correspondiente a esta estructura se escribe entre llaves `{ ... }`

Para cada valor de la variable que podamos prever, escribimos la palabra clave `case` seguida del valor de la variable, y después escribimos dos puntos(:). Escribimos después las instrucciones que se ejecutarán en el caso de que la variable tome este valor. Por último la palabra clave break; hace que salgamos de la instrucción switch una vez que se ha ejecutado el código.

Por último la instrucción `default:` (fíjate en que también aquí acaba con dos puntos) indica el código que se aplicará en caso de que la variable no tenga ningún valor de los indicados anteriormente mediante `case`.

3.7. Concepto de bucle

Un bucle es una estructura que hace que un código determinado se repita mientras una cierta condición se cumpla.

Para que el bucle no se repita indefinidamente, debemos variar las condiciones en cada vuelta, de manera que llegue un momento en que la condición no se cumpla.

Lo normal, si queremos que un bucle se repita un número determinado de veces, es tener una variable que controla el bucle y que iremos cambiando en cada vuelta.

3.8. El bucle while

Esta es la estructura más sencilla para un bucle. Su sintaxis es la siguiente:

```
while ( /*condición*/ ) { /*código a ejecutar*/ }
```

Se escribe la palabra reservada `while`, después entre paréntesis la condición, y después entre llaves el código que debe repetirse en cada vuelta. El siguiente ejemplo crea la tabla de multiplicar del 7 con un bucle while:

```
$i=1; //variable de control del bucle, en estado inicial.
while ($i<=10) { //Condición: la variable debe ser menor o igual que 10.
       echo "7 x $i = ".7*$i."<br/>"; //en pantalla línea de la tabla.
       $i++; //aumentamos la variable en una unidad.
       }
```

3.9. El bucle do ... while

Esta es una variante del bucle `while`. La diferencia es que aquí el código se ejecuta siempre al menos una vez, aunque la condición no se cumpla. Su sintaxis es la siguiente:

```
do { /*código a ejecutar*/ } while ( /*condicion*/ )
```

Aquí ponemos primero el código que debe repetirse después de la palabra clave `do`, y detrás la palabra clave `while` con la condición entre paréntesis.

Veamos un ejemplo:

```
$n=0; //Número de vueltas
$i=1; //variable de control
do {
   echo "$i: Esta frase se repetirá $n veces"; //texto en pantalla
   $i++; //aumentar en una unidad la variable de control
   }
while ($i<=$n) //parar el bucle cuando llegue al número de vueltas.
```

En este ejemplo la condición no se cumple, pero el bucle se ejecutará una vez. Si cambiamos la variable `$n` por otro número el bucle se repetirá el número de veces que le indiquemos.

3.10. El bucle for

Esta estructura es idéntica a la que se utiliza en otros lenguajes como javascript, y su sintaxis es:

```
for ( /*inicialización*/ ; /*condición*/ ; /*actualización*/ ) {
    /* código que se repite */
    }
```

La estructura es similar al bucle `while`. Ponemos la palabra reservada `for` seguida de un paréntesis; después escribimos entre llaves el código que se repetirá. Lo que cambia es lo que ponemos dentro del paréntesis.

Dentro del paréntesis ponemos tres sentencias separadas por punto y coma que son:

- **Inicialización:** Definiremos aquí la variable de control del bucle, y le daremos un valor inicial.
- **Condición:** A la variable anterior le marcamos una condición, la cual, mientras se cumpla el bucle seguirá repitiéndose.
- **Actualización:** Modificamos aquí la variable anterior. De esta manera llegará un momento en que la condición no se cumpla, y saldremos del bucle.

El siguiente ejemplo muestra la tabla de multiplicar del 7.

```
for ($i=1 ; $i<=10 ; $i++) { //ini.:en 1; cond.:<=10; act.:número siguiente.
    $n= $i*7; //multiplicar;
    echo "7 x $i = $n <br/>"; //sacar en pantalla
    }
```

3.11. Bucle foreach para arrays

El bucle foreach es una estructura especializada en recorrer los elementos que contiene un array.

Calcula automáticamente el número de elementos que tiene el array y nos proporciona unas variables donde se muestran en cada repetición el valor y la clave de cada elemento. En el siguiente ejemplo mostramos su sintaxis:

```
foreach ($array as $clave=>$valor) {
        echo "clave: $clave , valor: $valor; ";
        }
```

El código empieza por la palabra clave `foreach` seguido de un paréntesis.

Dentro del paréntesis escribimos en primer lugar la variable del array `$array`, seguido de la palabra clave `as` y después una nueva variable, que aquí llamaremos `$clave`. Seguimos con los signos =>, y otra nueva variable que llamamos `$valor`.

El bucle empieza a leer el array por el primer elemento, las variables `$clave` y `$valor` recogen la clave y el valor del elemento. En cada vuelta del bucle, éste avanza un elemento, y al llegar al final se para.

En arrays asociativos obtenemos la clave y el valor, mientras que en arrays indexados la clave indica el número que el elemento ocupa en el array.

Para arrays indexados, si sólo queremos obtener el valor de los elementos podemos simplificar el código de la siguiente manera:

```
foreach ($array as $valor) {
        echo "$valor, ";
        }
```

4. Traspasar datos

4.1. Incluir archivos

Con PHP podemos hacer parte del código de la página en un archivo externo y después incluirlo en la página.

Esto permite que ciertas secciones que suelen ser comunes a la mayoría de páginas del sitio (por ejemplo cabecera, pie, menús, etc.) no haya que repetirlas en todas las páginas, sino que con incluir una referencia al archivo que las contiene se incluyen en la página.

Los archivos externos deben incluir trozos de código en HTML de la página, y pueden llevar incrustados scripts de PHP, javascript, CSS, etc. Estos se insertan en el punto de la página en el que indicamos su referencia.

La forma de incluir un archivo externo en la página con PHP es la siguiente:

```
<?php
require("ruta_archivo");
?>
```

Donde pone `ruta_archivo` escribiremos la ruta para abrir ese archivo, de la misma manera que haríamos con un enlace o una etiqueta tipo link.

Podemos también usar para incluir un archivo externo el siguiente código:

```
<?php
include("ruta_archivo");
?>
```

Este código funciona igual que el anterior, pero además es el único que PHP admite si está incluido dentro de una estructura condicional

La ventaja de usar archivos aparte es que podemos dividir la página en varias secciones, y trabajar en cada una de ellas independientemente. La página aunque la hayamos construido por partes, llegará al navegador como una sola unidad.

4.2. Formularios

Una de las utilidades de PHP es poder recoger los datos que el usuario proporciona en un formulario. Veremos aquí como recoger los datos y llevarlos como variable de PHP a otra página. En temas posteriores se verá cómo guardar éstos u otros datos de manera permanente.

4.2.1. El formulario

Para poder recoger los datos de un formulario, el código HTML debe tener los elementos que indicamos a continuación.

La etiqueta `form` debe llevar siempre los atributos `action` y `method`.

El atributo `action` tendrá como valor la ruta del archivo que recoge el formulario, el cual será una página en PHP.

El atributo method solo admite los valores `"post"` y `"get"`. Estos son métodos para traspasar datos. En principio utilizaremos el `post`.

Cada campo del formulario debe llevar el atributo `name`. Como valor pondremos una palabra que identifique el campo. Esta palabra es la que utilizará PHP para identificar el campo y recoger el dato que aporta el usuario.

Veamos un ejemplo de cómo puede quedar un formulario:

```
<form action="datos.php" method="post">
  <p> Nombre: <input type="text" name="nombre" /></p>
  <p> Nombre: <input type="password" name="clave" /></p>
      <input type="submit" value="enviar" />
</form>
```

4.2.2. La página de recogida

Un vez completo el formulario, al pulsar en enviar, se nos abre la página de recogida. En esta podemos ver los datos mandados por el usuario mediante el array interno `$_POST` (si hemos usado el método post) o `$_GET` (cuando usemos el método get).

Mediante el siguiente código PHP recogemos los datos en la página de recogida:

```
$tunombre=$_POST["nombre"];
```

Donde `$tunombre` es la variable que guarda los datos y `nombre` es el valor del atributo `name` del código HTML.

De igual manera se recogen los datos con el método GET:

```
$tunombre=$_GET["nombre"];
```

Datos de distintos campos.

En los campos de **texto** (`text, password, textarea`), el dato que se recoge es el texto que nos proporciona el usuario.

En los campos tipo **radio**, todos los botones asociados llevan el mismo valor para el atributo `name`, y deben llevar también el atributo `value` con distinto valor en cada botón. El dato que se recoge es el valor del atributo `value` del campo marcado.

En los campos de tipo **chebox** nos devolverá el valor del atributo `value` si el campo está seleccionado, si no lo está devuelve una cadena vacía. Si no ponemos el atributo `value` devuelve el valor `"on"`.

En las listas tipo **select** el atributo `name` debe ponerse en la etiqueta `select`. El dato que se recoge es el valor del atributo `value` de la opción seleccionada. Si La opción no tiene atributo `value`, el dato recogido es el texto de la opción. En listas de selección múltiple se devuelve un array con los datos seleccionados.

En otros tipos de campo como **hidden** o **button**, se sigue la norma general, es decir el valor de `name` es la referencia en PHP, mientras que el valor de `value` es el dato obtenido.

Los datos obtenidos están en la página de recogida, sin embargo éstos se perderán cuando el usuario cierre la página, a no ser que los guardemos antes. En temas posteriores veremos cómo guardar datos y dónde.

4.2.3. Enviar archivos

El usuario también puede enviarnos archivos mediante formularios, siempre que en el servidor nuestro haya espacio para almacenarlos. El formulario que debemos hacer en HTML para enviar archivos es un poco distinto del de enviar datos:

```
<form action="tuarchivo.php" method="post" name="tuarchivo"
     enctype="multipart/form-data">
<p>manda tu archivo: <input type="file" name="archivo" /></p>
<p><input type="submit" value="mandar archivo" /></p>
</form>
```

En la etiqueta `form` además de los atributos `action` y `method` debemos poner el siguiente atributo con el valor que aquí indicamos:

```
enctype="multipart/form-data"
```

Después pondremos en el formulario un campo `input type="file"` que es el que permite enviar el archivo a la página de recogida. Debemos poner también en este campo el atributo `name` que lo identificará en la página de recogida.

En la página de recogida el código PHP que debemos poner es diferente de cuando se mandan datos será el siguiente:

```
copy($_FILES['archivo']['tmp_name'],$_FILES['archivo']['name']);
```

Aquí `archivo` es el valor que le hemos dado al atributo `name` del campo tipo "file", del formulario.

La función `copy` de PHP copia un archivo de un lugar a otro. Cuando se nos manda el archivo se crea una copia del mismo en un archivo temporal. Esta la localizamos mediante `$_FILES['archivo']['tmp_name']`. Después hacemos una copia permanente mediante `$_FILES['archivo']['name']`. La copia se guardará en la misma carpeta en la que esté la página de recogida, por lo que debemos poner la página de recogida en la misma carpeta donde queremos guardar los archivos que nos envian.

El archivo que nos mandan se guarda con el nombre y extensión que tiene originalmente. Para ver el archivo que nos han mandado, si no sabemos su nombre, podemos obtener la ruta mediante:

```
$elarchivo=$_FILES['archivo']['name'];
```

El nombre del archivo se guarda en la variable `$elarchivo.` A partir de ahí, si sabemos que el archivo es de imagen, podemos sacarlo en pantalla mediante:

```
echo "<img src='$elarchivo' />";
```

En temas posteriores veremos más sobre el tratamiento de archivos con PHP para poder manejar los archivos que nos mandan los usuarios.

4.3. Traspasar variables en enlaces

Podemos pasar una o varias variables a otra página cuando la abrimos con un enlace, mediante el siguiente código:

```
<a href="destino.php?variable1=valor1&variable2=valor2 ...
&variableN=valorN "> pasar variables</a>
```

En el atributo `href` del enlace, después de poner la ruta pondremos un signo de interrogación ? y después los enlaces.

De cada variable pondremos el nombre (sin el signo dolar $) seguido del signo igual = y después su valor. El valor se pone siempre sin comillas, aunque sea una cadena de texto. La separación entre una variable y otra se hace con el signo ampersand &.

En la página de destino (la que se abre con el enlace), recogemos las variables con el método get (array interno `$_GET[]`):

```
$v1=$_GET["variable1"];
$v2=$_GET["variable2"];
```

En este método, tal como hemos visto ahora, las variables se crean y definen a la vez que el enlace. Esto no resulta muy práctico ya que la mayoría de las veces lo que queremos es enviar una variable que ya tenemos definida. Por ejemplo, tenemos la siguiente variable:

```
<?php $saludo="Hola mundo" ?>
```

No podemos enviarla directamente mediante:

```
<a href="destino.php?saludo=$saludo">Recibir saludo</a>
```

Ya que en la página de destino al aplicar el código `$v1=$_GET["saludo"];` obtendremos que `$v1="$saludo"`

Para conservar el valor de la variable debemos escribir el enlace dentro del código PHP, de la siguiente manera:

```
<?php echo "<a href='destino.php?saludo=$saludo'>Recibir saludo</a>"; ?>
```

Ahora en la página de destino mediante el código `$v1=$_GET["saludo"];` obtendremos que `$v1="Hola mundo"` .

5. Funciones predefinidas

5.1. Funciones de cadena

En este apartado veremos las funciones predefinidas de PHP para utilizar con variables de cadena o tipo "texto". Estas funciones no alteran el texto que pasamos en lo argumentos, por lo que los resultados se obtienen igualándolas a una variable.

5.1.1. Funciones generales para cadenas de texto:

Estas son las funciones de tipo general para cadenas de texto.

Funciones generales para cadenas de texto

Nombre	Código y explicación
strlen()	`$num = strlen($texto);` Cuenta el **número de caracteres** de una cadena. Su resultado es el número de caracteres que tiene la cadena pasada en el argumento `$texto`.
chr()	`$caracter = chr($num);` Pasamos como parámetro un número entre 0 y 255, y el resultado es el **carácter** asignado a ese número en el **código ASCII**. ASCII es un código que asocia cada letra, signo o carácter con un número. puedes ver la lista completa en *http://ascii.cl/es/codigos-html.htm*
ord()	`$num = ord("caracter");` Función contraria a la anterior. Pasamos como argumento un carácter y nos devuelve su código ASCII. Si pasamos más de un carácter devuelve únicamente el código ASCII del primero.
count_chars()	`$resultado = count_chars($texto,$modo);` **Analiza los caracteres del código ASCII** que aparecen en el texto. En `$texto` ponemos la cadena a analizar, y en `$modo` pasamos un número del 0 al 4. Según el modo que usemos la función devuelve: **Modo 0:** Un array asociativo donde la clave son todos los números de caracteres de ASCII, y el valor el número de veces que se repite ese carácter. **Modo 1:** Un array asociativo donde la clave nos da sólo los números de caracteres de ASCII que aparecen en el texto, y el valor el número de veces que se repite ese carácter. **Modo 2:** Un array asociativo con los caracteres del código ASCII que no aparecen en el texto. La clave es el número de carácter, y el valor es siempre igual a 0. **Modo 3:** Una cadena de texto en la que aparecen los caracteres que contiene el texto. Estos aparecen una sola vez (aunque estén repetidos), y ordenados según el código ASCII. **Modo 4:** Una cadena de texto en la que aparecen los caracteres del código ASCII que no aparecen en el texto, ordenados según este código.
str_word_count()	`$resultado = str_word_count($texto,$modo);` **Analiza las palabras del texto** y permite hacer varias operaciones según el modo. En `$texto` pasamos el texto a analizar, y en `$modo` pondremos un número del 0 al 2. Según el modo usado la función devolverá: **Modo 0:** El número de palabras que contiene el $texto. **Modo 1:** Un array indexado con todas las palabras que contiene el $texto. **Modo 2:** Un array asociativo donde la clave es el número de carácter de la cadena donde comienza la palabra (se empieza a contar desde 0), y el valor es la palabra en sí.
substr_count()	`$num = substr_count($texto,$subcadena);` Devuelve un número que indica las veces que una determinada `$subcadena` (carácter, palabra, etc.) aparece dentro la cadena `$texto`.

Nombre	Código y explicación
strpos()	`$num = strpos($texto,$subcadena);`
	Busca la `$subcadena` dentro de la cadena `$texto` y devuelve un número que indica la posición dentro de la cadena del primer carácter de la subcadena. Si está más de una vez se indica sólo el de la primera vez. Si no se encuentra devuelve una cadena vacía.
strrpos()	`$num = strrpos($texto,$subcadena);`
	Igual que el anterior pero aquí empieza a buscar por el final. Si la subcadena está varias veces nos da el comienzo de la última posición, empezando a contar por el principio.
stripos()	`$num = stripos($texto,$subcadena);`
	Igual que la función strpos() pero sin tener en cuenta la diferencia entre mayúsculas y minúsculas.
strripos()	`strripos($texto,$subcadena);`
	Igual que la función strrpos() pero sin tener en cuenta la diferencia entre mayúsculas y minúsculas.

5.1.2. Funciones de cambio de mayúsculas / minúsculas.

Las siguientes funciones trabajan con cadenas de texto, y cambian letras mayúsculas y minúsculas:

Funciones de cadena: Cambio mayúsculas / minúsculas

Nombre	Código y explicación
strtoupper()	`$texto2 = strtoupper($texto);`
	Devuelve la cadena de texto pasada en el argumento $texto con todas sus letras en mayúsculas.
strtolower()	`$texto2 = strtolower($texto);`
	Devuelve la cadena de texto pasada en el argumento $texto con todas sus letras en minúsculas.
ucwords()	`$texto2 = ucwords($texto);`
	Devuelve la cadena de texto pasada en el argumento $texto con la primera letra de cada palabra en mayúsculas, siempre que ésta sea un carácter alfabético.
ucfirst()	`$texto2 = ucfirst($texto);`
	Devuelve la cadena de texto pasada en el argumento $texto con la primera letra de la cadena en mayúsculas, siempre que ésta sea un carácter alfabético.

5.1.3. Remplazar o eliminar texto:

Las siguientes funciones remplazan o eliminan un texto o trozo de texto. Se emplean con cadenas.

Funciones de cadena: Remplazar o eliminar texto

Nombre	Código y explicación
str_replace()	`$nuevoTexto = str_replace($buscar,$remplazar,$texto);`
	Remplaza un trozo de texto por otro. Dentro de la cadena `$texto` (tercer argumento), busca la subcadena `$buscar` (primer argumento) y la remplaza por la subcadena `$remplazar` (segundo argumento).

Nombre	Código y explicación
str_ireplace()	`$nuevo_texto = str_ireplace($buscar,$remplazar,$texto);` Igual que la anterior, pero en la búsqueda no tiene en cuenta las diferencias entre letras mayúsculas y minúsculas
substr()	`$porcion = substr($texto,$comienzo,$longitud);` Devuelve una parte de la cadena `$texto` (1º argumento) . El 2º y 3º argumentos son números. el 2º indica en que carácter empieza la cadena devuelta (se empieza a contar en el 0), y el 3º indica su longitud. Si no se indica el 3º (longitud) la cadena devuelta llegará hasta el final. Si en el 2º(comienzo) se pone un número negativo, se empieza a contar desde el final de `$texto`, y si indicamos un número mayor que los caracteres que tiene `$texto` devolverá "false".
substr_replace()	`$txt2 = substr_replace($txt,$sustituir,$comienzo,$longitud);` Devuelve la cadena pasada en `$txt` (1º argumento), en la cual se sustituye una porción por la subcadena `$sustituir`. Los argumentos 3º y 4º son números e indican respectivamente el comienzo y la longitud de la porción de texto a sustituir. Su comportamiento es similar a los argumentos 2º y 3º de la función anterior.
str_pad()	`nuevoTexto = str_pad($texto,$largo,$relleno,$tipo_relleno);` Alarga la cadena `$texto` (1º argumento) con el carácter indicado en `$relleno` (3º argumento) hasta el número de caracteres indicado en `$largo`. El 4º argumento es opcional e indica a qué lado de la cadena se pone el relleno. Como valores podemos poner: `"STR_PAD_RIGHT"`, relleno por la derecha (opción por defecto); `"STR_PAD_LEFT"`, relleno por la izquierda; `"STR_PAD_BOTH"`, relleno repartido a ambos lados.
ltrim()	`$nuevoTexto = ltrim($texto,"lista_caracteres");` Devuelve el `$texto` (1º argumento), en el cual se eliminan los primeros caracteres cuando éstos están en la `"lista_caracteres"` (2º argumento).
rtrim()	`$nuevoTexto = rtrim($texto,"lista_caracteres");` Devuelve el `$texto` (1º argumento), en el cual se eliminan los últimos caracteres cuando éstos están en la `"lista_caracteres"` (2º argumento).
trim()	`$nuevoTexto = trim($texto,"lista_caracteres");` Devuelve el `$texto` (1º argumento), en el cual se eliminan tanto los primeros como los últimos caracteres cuando éstos están en la `"lista_caracteres"` (2º argumento).
strtr()	`$nuevoTexto = strtr($texto,"lista_1","lista_2");` Devuelve el `$texto` (1º argumento), en donde se sustituyen los caracteres de la `"lista_1"` por los de `"lista_2"`, de manera que el 1º carácter de lista_1 se cambia por el 1º de lista_2, el 2º de lista_1 por el 2º de lista_2 y así sucesivamente. Si alguna lista es más larga que la otra los caracteres sobrantes se ignoran.

5.1.4. Otras funciones de cadenas de texto

Otras funciones de cadena de texto

Nombre	Código y explicación
chunk_split()	`$nuevoTexto = chunk_split($texto,$num,$caracter);` Devuelve la cadena `$texto` (1º argumento) en la cual cada cierto número de caracteres `$num` (2º argumento), se inserta el texto indicado en `$caracter` (3º argumento).
str_repeat()	`$repetido = str_repeat($texto,$num);` Devuelve el texto de la cadena `$texto` (1º argumento) repetido tantas veces como se

Nombre	Código y explicación
	indica en $num (2º argumento).
strrev()	`$alreves = strrev($texto);` Devuelve la cadena $texto (argumento) invertida (de derecha a izquierda).
htmlspecialchars()	`$texto2 = htmlspecialchars($texto);` Devuelve la cadena $texto (argumento) en la cual se traducen al lenguaje html los siguientes caracteres especiales: `& = & " = &quot; < = < > = >`
htmlentities()	`$texto2 = htmlentities($texto);` Devuelve la cadena $texto (argumento) en la cual se traducen al lenguaje HTML todos caracteres especiales que se escriben de forma distinta en HTML. Además de los caracteres indicados en la función anterior, se cambian otros como son las vocales con acento o la letra "ñ".
strip_tags()	`$texto2 = strip_tags($texto,"etiq_permitidas");` Elimina todas las etiquetas de HTML y PHP que haya en el $texto (1º argumento). El 2º argumento es opcional e indica las etiquetas permitidas. Éstas se escribirán entre los signos `< ... >`.

5.2. Funciones para arrays

Las siguientes funciones se utilizan con arrays. En muchas de ellas se cambia el array original, por lo que no las igualamos a ninguna variable. En este caso si queremos conservar el array original debemos hacer una copia antes de utilizar la función.

5.2.1. Ordenar arrays

Las siguientes funciones ordenan los arrays. Todas ellas cambian el array original.

Ordenar arrays indexados

Nombre	Código y explicación
sort()	`sort($array);` Ordena el array en orden ascendente
rsort()	`rsort($array);` Ordena el array en orden descendente

Ordenar arrays asociativos

Nombre	Código y explicación
asort()	`asort($array);` Ordena el array en orden ascendente por valor.
arsort()	`arsort($array);` Ordena el array en orden descendente por valor.
ksort()	`ksort($array);` Ordena el array en orden ascendente por clave.
krsort()	`krsort($array);` Ordena el array en orden descendente por clave.

5.2.2. Insertar elementos

Para insertar elementos en arrays asociativos basta con crearlos nuevos, sin embargo en los arrays indexados es más complicado ya que éstos tienen un cierto orden. Es por eso que para ello tenemos las siguientes funciones.

Insertar elementos en arrays indexados

Nombre	Código y explicación
array_push()	`array_push($array,"elemento1","elemento2");` Añade elementos al final del array original `$array` (1º argumento). El 2º argumento y los siguientes son los elementos a añadir.
array_unshift()	`array_unshift($array,"elemento1","elemento2");` Añade elementos al principio del array original `$array` (1º argumento). El 2º argumento y los siguientes son los elementos a añadir. Los demás elementos se reordenan según su posición.
array_pad()	`$nuevoArray = array_pad($array,$tamano,$relleno);` Devuelve un `$nuevoArray` en el que el array del 1º argumento `$array`, amplia su tamaño hasta alcanzar el número indicado en `$tamano` (2º argumento). Los nuevos elementos toman el valor de `$relleno` (3º argumento). Aquí el $array original se conserva.

5.2.3. Eliminar elementos

Las siguientes funciones eliminan elementos de un array:

Eliminar elementos de un array

Nombre	Código y explicación
array_shift()	`$elemento=array_shift($array);` Elimina el primer elemento del array original `$array` (argumento) y lo devuelve en la variable `$elemento`. En arrays indexados los elementos que quedan e reordenan.
array_pop()	`$elemento = array_pop($array);` Elimina el último elemento del array original `$array` (argumento) y lo devuelve en la variable `$elemento`.
array_unique()	`$array2 = array_unique($array);` Devuelve un array (`$array2`) igual que el `$array` (argumento) en el que se eliminan los elementos repetidos dejando sólo uno. El array original se conserva.

5.2.4. Funciones para arrays y cadenas de texto.

Estas son las principales funciones que relacionan las cadenas de texto con los arrays.

Funciones para cadenas de texto y arrays

Nombre	Código y explicación
explode()	`$array = explode($separador,$texto);` Separa los elementos del `$texto` (2º argumento) y los devuelve en un `$array`. El `$separador` (1º argumento) indica el carácter o caracteres que, cada vez que aparezcan en el texto se hará una partición de un elemento. Ej.: $separador=" " dividirá el texto en palabras.

Nombre	Código y explicación
implode()	`$texto = implode($elem_union,$array);` Función contraria a `$explode`. Junta los elementos de un `$array` (2º argumento) en una cadena de `$texto`. En `$elem_union` pondremos uno o varios caracteres que aparecerán entre los elementos del array. Ej.: `$elem_union=", "` muestra los elementos separados por comas.
join()	`$texto = join($elem_union,$array);` Convierte una array en una cadena de texto, exactamente igual que la anterior. Los argumentos que pasamos son los mismos y funciona de igual manera.
chunk_split()	`$array = chunk_split($texto,$num);` Devuelve un array en el que cada elemento consiste en un trozo de la cadena `$texto` (1º argumento) de tantos caracteres como se indica en `$num` (2º argumento).

5.2.5. Otras funciones para arrays

Veremos aquí otras funciones útiles para trabajar con arrays, aunque aquí no están todas, éstas son las más habituales:

Otras funciones para arrays

Nombre	Código y explicación
array_reverse()	`$array2 = array_reverse($array);` Devuelve un array `$array2` con los mismos elementos que el array original `$array` (argumento) pero en orden inverso.
in_array()	`$resultado = in_array($array,$valor);` Comprueba i un `$valor` (2º argumento) es un elemento del `$array` (1º argumento). Devuelve un valor booleano (verdadero 1, falso "").
array_slice()	`$array2 = array_slice($array,$posicion,$tamano);` Extrae parte de los elementos de un array indexado `$array` (1º argumento) en un nuevo array `$array2`. `$posición` (2ºargumento) es un número que indica el índice a partir del cual se crea el nuevo array, y en `$tamano` se indica el número de elementos que tendrá el nuevo array. De no ponerse este último el nuevo array irá hasta el final.
array_values()	`$array2 = array_values($array);` Devuelve un array indexado `$array2` cuando le pasamos un array asociativo `$array` (argumento). Las claves se eliminan, y en su lugar tenemos los índices.
array_keys()	`array_keys($array);` Devuelve un array indexado `$array2` que consiste en las claves del array asociativo `$array` (argumento).
array_merge()	`$array3 = array_merge($array1,$array2);` Devuelve un array `$array3` que es la suma de los dos arrays pasados como argumentos. A los elementos de `$array1` (1º argumento) se le añaden al final los elementos de `$array2` (2º argumento).

5.3. Funciones matemáticas

Las funciones matemáticas nos permiten hacer variaciones y operaciones con los números. Aunque tenemos los operadores que permiten hacer operaciones, hay otra serie de operaciones más complejas que haremos mediante estas funciones.

5.3.1. Constantes matemáticas.

Estas son una serie de números especiales usados por los matemáticos (PI, e, constante de Euler, etc.). En PHP accedemos a ellos mediante una serie de constantes, las constantes matemáticas.

Constantes matemáticas

Constante	valor	Descripción
M_PI	`3.1415926535898`	Número pi, relación circunferencia/diámetro.
M_E	`2.718281828459`	Número e, base de logaritmos naturales.
M_EULER	`0.57721566490153`	Constante de Euler.
M_SQRT2	`1.4142135623731`	Raíz cuadrada de 2.
M_SQRT3	`1.7320508075689`	Raíz cuadrada de 3.
M_SQRTPI	`1.7724538509055`	Raíz cuadrada del número pi.
M_LOG2E	`1.442695040889`	Logaritmo en base 2 del número e.
M_LOG10E	`0.43429448190325`	Logaritmo en base 10 del número e.
M_LN2	`0.69314718055995`	Logaritmo neperiano (base e) de 2.
M_LN10	`2.302585092994`	Logaritmo neperiano (base e) de 10.

5.3.2. Funciones de cálculo

Estas funciones permiten hacer ciertas operaciones algo más complejas que las que nos permiten los operadores.

Funciones de cálculo

Nombre	Código y explicación
pow()	`$pot = pow($base,$exp);` **Potenciación:** Eleva el número indicado en `$base` (base de la potencia), al número indicado en `$exp` (exponente).
sqrt()	`$raiz2 = sqrt($num);` **Raíz cuadrada:** calcula la raíz cuadrada del número pasado como argumento.
log()	`$log = log($num,$base);` **Logaritmo:** Devuelve el resultado del logaritmo para `$num` (1º argumento). El 2º argumento es opcional, e indica la base del logaritmo. Si no se especifica el logaritmo será en base e (logaritmo natural).
log10()	`$log10 = log10($num);` **Logaritmo en base 10:** Devuelve el resultado del logaritmo en base 10 para el argumento pasado.
decbin()	`$bin = decbin($num);` **Conversión a binario:** Convierte el número pasado en el argumento en número binario (en base 2).
bindec()	`$dec = bindec($num);` **Conversión a decimal:** Función contraria a la anterior. Le pasamos un número binario en el argumento y lo convierte a decimal.
decoct()	`$oct = decoct($num);` **Conversión a octal:** Convierte el número pasado en el argumento en número octal (en base 8).

Nombre	Código y explicación
dechex()	`$hex = dechex($num);` **Conversión a hexadecimal:** Convierte el número pasado en el argumento en número hexadecimal (en base 16).
base_convert()	`$num2 = base_convert($num,$base1,$base2);` **Conversión de una base a otra:** Convierte el `$num` (1º argumento), e una base a otra. La `$base1` (2º argumento) indica la base inicial en la que esta `$num`, y `$base2` es la bse a la que se convierte. Las bases deben ser números entre el 2 y el 36 (ambos inclusive). En bases mayores de 10 se usarán las letras minúsculas del alfabeto.
max()	`$maximo = max($array);` **Máximo valor:** Devuelve el valor más alto de todos los que se le pasan en una lista, ésta puede estar en forma de array o pasar los números directamente como varios argumentos.
min()	`$minimo = min($array);` **Mínimo valor:** Devuelve el valor más bajo de todos los pasados en una lista, éstos pueden pasarse en forma de array o pasarlos directamente como varios argumentos.

5.3.3. Redondeo y valor absoluto

Las siguientes funciones modifican el número que se les pasa para redondearlo o mostrar su valor absoluto.

Funciones de redondeo y valor absoluto

Nombre	Código y explicación
abs()	`$num2 = abs($num);` **Valor absoluto:** devuelve el valor absoluto del número pasado como argumento, siempre positivo, sin tener en cuenta el signo.
round()	`$num2 = round($num,$precision);` **Redondeo:** Redondea el `$num` (1º argumento) al entero más cercano. El 2º argumento (`$precision`) es opcional, e indica el número de decimales en el redondeo. Si es negativo se redondeará a entero convirtiendo las últimas cifras en ceros.
ceil()	`$num2 = ceil($num);` **Redondeo al alza:** Redondea el número pasado en el argumento al siguiente entero inmediatamente superior.
floor()	`$num2 = floor($num);` **Redondeo a la baja:** Redondea el número pasado en el argumento al número entero inmediatamente inferior.
number_format()	`$num2 = number_format($num,$decimales,"sep_decimal","sep_miles")` **Formato de números:** Define cómo se presentará el número en pantalla, podemos poner 1, 2, o 4 argumentos, (pero no 3). Con un argumento el número se mostrará tal cual. Con dos argumentos, el 2º indica el número de decimales que debe mostrarse. En el 3º y 4º argumento escribiremos los signos que deben mostrarse para separar los decimales (en 3º argumento) y para separar los grupos de miles (4º argumento).

5.3.4. Generar un número aleatorio.

Funciones para crear número aleatorio

Nombre	Código y explicación
rand()	`$num = rand($min,$max)` **Número aleatorio:** Genera un número entero aleatorio comprendido entre los números pasados en el 1º argumento `$min` (mínimo) y el 2º `$max` (máximo). Si no pasamos argumentos los límites estarán entre 0 y RAND_MAX (en Windows es 32768).
mt_rand()	`$num = mt_rand($min,$max);` **Número aleatorio:** Esta función funciona exactamente igual que la anterior, pero tiene la ventaja de que genera los números 4 veces más rápido.

5.3.5. Funciones trigonométricas

Las funciones trigonométricas son aquellas que relacionan la circunferencia con el radio, o los lados de un triángulo rectángulo con sus ángulos.

Los ángulos pueden medirse en grados o en radianes, sin embargo PHP sólo trabaja con radianes. Es por esto que ponemos también las funciones de conversión entre grados y radianes.

Funciones Trigonométricas

Nombre	Código y explicación
deg2rad()	`$rad = deg2rad($grados);` **Conversión grados a radianes:** Convierte el número pasado en el argumento a radianes.
rad2deg()	`$grados = rad2deg($rad);` **Conversión radianes a grados:** Convierte el número pasado en el argumento a grados.
sin()	`$num = sin($rad);` **Seno:** Calcula el seno del ángulo pasado en el argumento.
cos()	`$num = cos($rad);` **Coseno:** Calcula el coseno del ángulo pasado en el argumento.
tan()	`$num = tan($rad);` **Tangente:** Calcula la tangente del ángulo pasado en el argumento.
asin()	`$rad = asin($num);` **Arco-seno:** Calcula el arco-seno del número pasado en el argumento.
acos()	`$rad = acos($num);` **Arco-coseno:** Calcula el arco-coseno del número pasado en el argumento.
atan()	`$rad = atan($num);` **Arco-tangente:** Calcula el arco-tangente del número pasado en el argumento.

5.4. Fechas

5.4.1. Funciones básicas.

La marca de tiempo Unix

La marca de tiempo Unix o tiempo Unix es la manera que tienen los ordenadores de medir el tiempo. Esta consiste en un número que indica los segundos transcurridos desde el 1 de enero de 1970 a las 0h. 0m. 0s. hora GTM.

De esta manera a cada fecha le corresponde un número. Las fechas anteriores dan un número negativo.

time();

La función `time()` nos devuelve la marca de tiempo Unix de la fecha actual (en el momento de cargarse la página). Toma la hora local del ordenador del usuario. Esta función no tiene argumentos:

```
$fecha_actual = time();
```

mktime($hora,$min,$seg,$mes,$dia,$ano);

Esta función devuelve la marca de tiempo Unix para una fecha concreta. La fecha la pasamos en los argumentos de la función en el siguiente orden: hora, minuto, segundo, mes, día, año. Los argumentos se escribirán con números. La función tiene en cuenta la hora local.

gmmktime($hora,$min,$seg,$mes,$dia,$ano);

Esta función es igual que la anterior, pero nos devuelve la fecha en horario GTM (internacional o del meridiano de Greenwich).

5.4.2. Formato de fechas

Hasta ahora obtenemos las fechas en tiempo Unix, pero lo que nos interesa es poder mostrarla en distintos formatos más normales. Usamos para ello la función `date()`.

```
$fecha = date($formato,$tiempo_Unix);
```

En el segundo argumento pondremos el tiempo Unix de la fecha que queremos dar formato. Si no se pone este argumento se toma la fecha actual.

El primer argumento es el formato que le daremos a la fecha, este consiste en una serie de letras que indican los elementos a mostrar (día, mes, hora, etc.). Entre estas letras podemos poner otros elementos como guiones, comas, espacios en blanco, etc. que formarán parte de la fecha. Por ejemplo para la fecha actual pondremos:

```
echo date("d-m-Y // h:i:s");
```

Lo cual lo veremos así: 28-11-2013 // 04:41:23

Los distintos caracteres que podemos poner para definir los elementos de la fecha en el formato los vemos en la siguiente tabla:

Elementos del formato de fecha:

Carácter	Descripción	Valores devueltos
Día	**-------**	**-------**
d	Día del mes. 2 dígitos, con 0 inicial	01 a 31
j	Día del mes sin 0 inicial	1 a 31
Semana	**-------**	**-------**
D	Día de la semana abreviado (inglés, 3 letras)	Mon a Sun
l (ele minúscula)	Día de la semana completo (inglés)	Sunday a Saturday
N	Número del día de la semana	1 (lun) a 7 (dom)
w	Número del día de la semana	0 (dom) a 6 (sáb)
W	Semanas transcurridas del año	1 a 52 ó 53
Mes	**-------**	**-------**
m	Número de mes con 0 inicial	01 a 12
n	Número de mes sin 0 inicial	1 a 12
F	Mes (texto en inglés)	January a December
M	Mes abreviado. Tres letras (en inglés)	Jan a Dec
t	Días que tiene el mes.	28 a 31
año	**-------**	**-------**
Y	Año en 4 dígitos	Ej.: 1990 ó 2003
y	Año en 2 dígitos	Ej.: 90 ó 03
L	Indica si el año es bisiesto	1 (sí), 0 (no)
Hora, min, seg.	**-------**	**-------**
g	Hora de 1 a 12 con 0 inicial	00 a 12
h	Hora de 1 a 12 sin 0 inicial	1 a 12
G	Hora de 0 a 23 con 0 inicial	00 a 23
H	Hora de 0 a 23 sin 0 inicial	0 a 23
a	Ante meridiano o Post meridiano. Minúsculas.	am ó pm
A	Ante meridiano o Post meridiano. Mayúsculas.	AM ó PM
i	Minuto con 0 inicial	00 a 59
s	Segundo con 0 inicial	00 a 59
Fecha completa	**-------**	**-------**
c	Fecha en formato ISO 8601	Ej.: 2005-03-12T15:19:21+00:00
r	Fecha en formato RFC 2822	Ej: Thu, 21 Dec 2000 16:01:07 +0200
Otros	---	---
e	Identifica zona horaria	Ej.: Europe/Paris
I (i mayúscula)	Indica si estamos en horario de verano.	1 = sí, 0 = no
P	Diferencia con la hora GTM (Greenwich) indica horas y minutos.	Ej: +02:00
Z	Diferencia con la zona horaria en segundos.	-43200 a 43200
U	Tiempo Unix de la fecha dada	igual que en mktime()

5.4.3. Fecha en idioma local

En el formato anterior se nos muestran los días de la semana y los meses en inglés. Sin embargo nosotros queremos mostrar la fecha en nuestro idioma local. Para ello utilizamos en principio la siguiente función para indicar cuál es la información que debe tratarse en idioma local:

```
setlocale($categoría,$localización);
```

En el primer argumento se indica el tipo de información que debe tratarse de forma local. Para la fecha pondremos: `CL_Time`.

En el segundo argumento pondremos en una cadena el código del país o región. Si se pone una cadena vacía. `""` se utilizará el código del país donde esté el servidor:

```
setlocale(CL_Time,"");
```

Esta función afecta también a las demás páginas del sitio, por lo que las demás páginas se verán afectadas también por esta función.

El problema está cuando el servidor donde tenemos la página está alojado en otro país. En ese caso debemos definir el código del país o región donde estamos. Este código puede ser diferente en distintos servidores, por lo que lo mejor es poner una lista de los posibles códigos separados por comas. Para España la función se definirá así:

```
setlocale(LC_TIME,"esp,sp,spa,es,spanish");
```

Además es posible que nuestro servidor tenga diferente zona horaria que nosotros, por lo que utilizaremos esta función para ponerla en horario español:

```
date_default_timezone_set("Europe/Madrid");
```

En el argumento pondremos la zona horaria de donde estamos.

Si estamos en otro país que no sea España, podemos consultar la lista de códigos de países en la página *http://www.loc.gov/standards/iso639-2/php/code_list.php.*

También podemos consultar la lista de zonas horarias en *http://es.php.net/timezones*

Después utilizaremos la siguiente función para mostrar la fecha en formato local:

```
strftime($formato,$tiempo_Unix);
```

La función es similar a `date()` por lo que como 2º argumento pondremos el tiempo Unix de la fecha. Si no lo ponemos nos dará la fecha actual.

El primer argumento indica el formato, que se escribe de forma parecida a `date()`, sólo que los elementos son diferentes. Estos son los siguientes.

Elementos para formato de fecha local

Carácter	Descripción	Valores devueltos
Día	--------	--------
%d	Día del mes con 0 inicial	`01 a 31`
%e **%#d (en Windows)**	Día del mes sin 0 inicial.	`1 a 31`
Semana	--------	--------
%A	Día completo de la semana.	`Lunes a domingo`
%a	Dia abreviado de la semana	`lun a dom`
%w	Día de la semana en número.	`0 (Domingo) a 6 (Lunes)`
%U	Semana del año: la semana empieza en domingo.	`1 a 53`
%W	Semana del año: la semana empieza en lunes.	`1 a 53`

Carácter	Descripción	Valores devueltos
Mes	**--------**	**--------**
%B	Nombre completo del mes.	`Enero a Diciembre`
%b	Nombre abreviado del mes.	`ene a dic`
%m	Mes en número con 0 inicial.	`01 a 12`
Año	--------	`--------`
%C	Dos primeros dígitos del año	`Ej.: 20 (para 2010)`
%y	Dos últimos dígitos del año.	`13 (para 2013)`
%Y	Año completo con lo cuatro dígitos.	`2010`
Hora min. y seg.	**--------**	**--------**
%H	Hora en formato 00 a 23.	`00 a 23`
%k	Hora en formato 0 a 23. (un dígito para menor de 10)	`0 a 23`
%I	Hora en formato 01 a 12.	`01 a 12`
%l ("L" minúscula)	Hora en formato 1 a 12.	`1 a 12`
%M	Minuto en formato 00 a 59.	`00 a 59`
%S	Segundo en formato 00 a 59.	`00 a 59`
Otros	**--------**	**--------**
%x	Fecha preferida, sin la hora.	`ej: como "%d/%m/%Y"`
%X	Hora preferida, sin la fecha.	`ej: como "%H:%M:%S"`
%c	Fecha y hora completa, preferida.	`ej: "%d/%m/%Y H:%M:%S"`
%Z	Zona horaria local.	`Ej: Europe/Madrid.`

5.5. Otras funciones

5.5.1. Cambiar el tipo de variable

Las siguientes funciones fuerzan a la variable pasada en el argumento a cambiar de tipo de variable.

Funciones de cambio de variable

Nombre	Código y explicación
floatval()	`$var2 = floatval($var);` Devuelve la variable pasada como argumento transformada en número real.
intval()	`$var2 = intval($var);` Devuelve la variable pasada como argumento transformada en número entero.
strval()	`$var2 = strval($var);` Devuelve la variable pasada como argumento transformada en cadena de texto
settype()	`$vr2 = settype($var,"tipo")` Devuelve la variable pasada en el 1º argumento transformada al tipo indicado en el 2º argumento. Los valores admitidos en el 2º argumento son los siguientes: `"integer", "double", "string", "array", "object"`.

5.5.2. Comprobar el tipo de variable.

Las siguientes funciones comprueban si la variable es del tipo indicado:

Funciones de comprobación del tipo de variable

Nombre	Código y explicación
is_int()	`$respuesta = is_int($var);` Comprueba si la variable pasada en el argumento es un número entero. Devuelve verdadero (1) o falso ("");
is_float()	`$respuesta = is_float($var);` Comprueba si la variable pasada en el argumento es un número real. Devuelve verdadero (1) o falso ("");
is_string()	`$respuesta = is_string($var);` Comprueba si la variable pasada en el argumento es una cadena de texto. Devuelve verdadero (1) o falso ("");
is_array()	`$respuesta = is_array($var);` Comprueba si la variable pasada en el argumento es un array. Devuelve verdadero (1) o falso ("");
is_object()	`$respuesta = is_object($var);` Comprueba si la variable pasada en el argumento es un objeto. Devuelve verdadero (1) o falso ("");
gettype();	`$respuesta = gettype($var);` Dependiendo del tipo de variable que pasamos en el argumento devuelve una de las siguientes cadenas de texto: `"integer"` (entero), `"double"` (decimal), `"string"` (cadena), `"array"` (array), `"object"` (objeto), `"unknown type"` (tipo desconocido)

5.5.3. Más funciones de variables:

isset()

```
$respuesta = isset($var);
```

Compueba si la variable existe. Si es así devuelve verdadero (1), y si no devuelve falso ("");

unset()

```
unset($var);
```

Elimina la variable pasada en el argumento.

La función eval()

Esta función convierte el valor de una variable en código PHP. Normalmente son variables de cadena donde se guarda el código. Ejemplo:

```
$a="echo 'hola mundo';";eval($a);
```

Redireccionar páginas

Si queremos que cuando el usuario entre en una página, esta nos redireccione a otra utilizaremos la siguiente función:

```
header("Location:ruta_página");
```

En donde `ruta_pagina` será la dirección a donde se redirecciona. El script donde esté esta función debemos ponerlo al principio de la página, incluso antes que la etiqueta !DOCTYPE.

5.5.4. Obtener la URL actual

A veces puede interesarnos obtener los datos del servidor donde estamos alojados. Lo hacemos mediante el siguiente array interno de PHP:

`$_SERVER['HTTP_HOST'];` : devuelve el dominio actual

`$_SERVER['SERVER_PORT'];` : devuelve el puerto actual

`$_SERVER['REQUEST_URI'];` : devuelve la uri actual

6. Cookies y Sesiones

6.1. Introducción

Estas son dos formas de disponer de datos y variables en todo el sitio web. El ámbito de las variables es la página, por lo que al cambiar de página (dentro del mismo sitio), las variables que tenemos desaparecen. Tanto cookies como variables solucionan este problema, si bien de forma diferente.

6.2. Cookies

6.2.1. Concepto

Una cookie es un archivo que se crea automáticamente para poder guardar un dato o una variable. Una vez creado para utilizarlo en otra página no tenemos más que abrirlo en esa otra página.

Sin embargo las cookies tienen una serie de características que restringen bastante su uso:

- La cookie se guarda como un archivo en el navegador del usuario.
- El usuario no debe haber desactivado el uso de cookies en el navegador.
- Las cookies tienen fecha de caducidad. por lo que estarán disponibles hasta esa fecha.
- El usuario debe utilizar el mismo navegador para recuperarlas.
- Se puede acceder a una cookie desde cualquier página del sitio, y durante el tiempo que esté activa, siempre que se sepa su nombre.
- En una cookie sólo puede guardarse texto, y no puede ocupar más de un Kb.

6.2.2. Insertar cookies

Para crear nuevas cookies utilizamos la función `set_cookie()`. Esta debe ponerse al principio de la página, incluso antes de la etiqueta `!DOCTYPE`. Esta función tiene varios argumentos, sólo el primero es obligatorio, pero para que la cookie funcione debemos incluir los tres primeros:

```
set_cookie($nombre,$valor,$caducidad);
```

En cada uno de estos argumentos pondremos lo siguiente:

`$nombre`: Nombre que le pondremos a la cookie, necesario para poder recuperarla.

`$valor`: Es el dato que queremos guardar en la cookie.

`$caducidad`: Marca de tiempo Unix (número correspondiente a una fecha), que indica hasta cuando la cookie estará activa

Para indicar la fecha de caducidad lo normal es crearla a partir de la fecha actual (funcion `time()`), a la que se le suma el tiempo que queremos que esté activa. Ejemplo de cookie:

```
set_cookie("saludo","Hola mundo",time()+3600*24*30);
```

La cookie de este ejemplo tiene como nombre `"saludo"`, como valor `"Hola mundo"` y como caducidad 30 días (3600*24*30 = segundos que tiene un mes).

Podemos además añadir otros argumentos a la función que crea la cookie, aunque no suelen utilizarse mucho debemos conocerlos:

- **ruta** : Por defecto la cookie sólo está disponible en el directorio (carpeta) en que se ha creado. Si queremos ampliarlo a otros directorios pondremos aquí la ruta. Para todo el sitio pondremos `"/"`.
- **dominio** : Si el sitio tiene varios subdominios, la cookie sólo está disponible en el subdominio que fue creado, para ampliarlo a otros subdominios lo indicaremos aquí. Para que esté en todos ellos indicaremos el nombre del dominio.
- **seguro** : Este argumento indica si la cookie debe transmitirse únicamente como https (de forma segura), su valor para ello debe ser el booleano `true`.

6.2.3. Acceso a cookies

Para poder acceder a una cookie desde cualquier página en la que esté disponible lo haremos mediante el array interno `$_COOKIE`. Desde ahí con el nombre de la cookie accedemos a su valor.

Por ejemplo si hemos guardado esta cookie:

```
set_cookie("saludo","Hola mundo",time()+3600*24*30);
```

Para poder ver el dato en pantalla en otra página en la que esté disponible escribiremos:

```
echo $_COOKIE['saludo'];
```

6.2.4. Borrar cookies

Una cookie se borra automáticamente al alcanzar la fecha de caducidad, o al reescribirla de nuevo, en este último caso es remplazada por la nueva cookie.

También podemos borrarla explícitamente, para ello la volvemos a reescribir mediante set_cookie(), pero poniendo sólo el primer argumento, es decir, el nombre:

```
set_cookie('saludo');
```

Borramos aquí la cookie que hemos usado en ejemplos anteriores.

6.3. Sesiones

6.3.1. Concepto

Una sesión es el recorrido que hace un usuario por un sitio web, desde que entra hasta que sale, es decir todas las páginas que recorre en el sitio.

Las variables normalmente expiran al cerrar cada página, sin embargo en PHP se pueden crear variables de sesión, que estén disponibles en todas las páginas.

Para ello debemos primero crear una sesión, y después las variables de sesión.

Las sesiones se utilizan para crear cuentas o sesiones de usuario, páginas de compras, etc.

Las variables estarán disponibles en todas las páginas de la sesión hasta que ésta se cierre. Al cerrar la sesión éstas se pierden. Su utilidad está en poder disponer de ellas en distintas páginas mientras que el usuario navega por nuestro sitio. Para guardarlas de forma permanente utilizaremos otros métodos (cookies, archivos o bases de datos).

6.3.2. Iniciar sesión.

Para iniciar una sesión debemos utilizar la siguiente función:

```
session_start();
```

Esta función debe escribirse al principio de la página, antes de cualquier script, antes de la etiqueta `!DOCTYPE` y antes incluso de cualquier otra sentencia en PHP como pueden ser la función `header()` o `set_cookie()` vistas anteriormente.

La función `session_start()` no tiene argumentos. Todas las páginas en las que utilicemos variables de sesión deben llevar esta función al principio.

La función `session_start()` busca si el usuario tiene abierta una sesión, y entra en ella, si no la tiene abierta la crea y le da un identificador, que será distinto para cada usuario. Puede, por tanto haber varios usuarios viendo la página al mismo tiempo, y cada uno de ellos tendrá una sesión distinta.

6.3.3. Variables de sesión

En todas las páginas identificadas como de la sesión (con la función session_start()) hay disponible el array interno `$_SESSION`, con el que podemos guardar o mostrar las variables de sesión.

Para crear una variable de sesión creamos un nuevo elemento de este array:

```
$_SESSION['saludo']="Hola mundo.";
```

Si queremos pasar una variable de la página lo haremos:

```
$_SESSION['nombre']=$nombre;
```

Para poder ver o utilizar la variable en otra página de la sesión buscamos el elemento del array:

```
echo $_SESSION['saludo'];
$nombre = $_SESSION['nombre'];
```

6.3.4. Manejar sesiones y variables.

Las funciones `isset()` y `unset()` permiten respectivamente comprobar si una variable existe, y eliminar la variable. Podemos aplicarlas también a las variables de sesión.

```
$comprobar=isset($_SESSION['nombre']);
```

Devuelve un valor booleano (`true, false`), por lo que comprueba si la variable ha sido creada.

```
unset($_SESSION['saludo']);
```

Elimina esta variable de sesión.

Si queremos eliminar todas las variables de sesión utilizaremos la función:

```
session_unset();
```

Esta función no tiene argumentos, elimina y libera el espacio que ocupan todas las variables de sesión.

Si queremos eliminar todos los datos referentes a la sesión, utilizaremos la función:

```
session_destroy();
```

Elimina la sesión en sí con todos sus datos, pero no elimina las cookies asociadas.

6.3.5. Otras variables de sesión

Otras variables nos permiten ver información sobre la sesión o cambiar sus datos;

- **`session_id()`** : Lee el identificador de sesión para el usuario. este consiste en una serie de caracteres alfanuméricos que identifican a cada usuario. Podemos cambiarlo si le pasamos otro como primer argumento.
- **`session_name()`** : Identifica el nombre de la sesión. Por defecto suele ser `PHPSESSID` pero podemos cambiarlo por otro si se lo pasamos como primer argumento.
- **`session_save_path()`** : Lee la ruta en la que se guardan los archivos de los datos de la sesion actual. Podemos cambiarlos especificando una nueva ruta como argumento.

7. Manejar archivos

Con PHP podemos crear, modificar, guardar, y leer archivos en nuestra página. Esto nos permite guardar datos de forma permanente, de manera que podamos recuperarlos al abrir la página o cuando el usuario inicie su sesión.

7.1. El método fopen()

El método fopen nos permite manejar archivos. El método utiliza varias funciones. Se abre un archivo mediante la función `fopen()`, si el archivo no existe se crea. Después tenemos otras funciones para modificarlo, y una vez modificado se cierra mediante la función `fclose()`:

```
$manejador = fopen($ruta,$modo); //abrir o crear el archivo
//... código para manejar el archivo ...
fclose($manejador)
```

Debemos guardar la función `fopen()` en una variable (`$manejador`), esta variable será el manejador del archivo, y es la que se le pasa como argumento a la función `$fclose()` para cerrar el archivo.

La función `$fopen()` tiene dos argumentos. En el primero pondremos la ruta del archivo. La función abre el archivo, y si el archivo no existe, crea uno nuevo.

El segundo argumento indica el modo en que se abre el archivo. Con cada modo tendremos unas opciones de trabajo distinto. Los distintos valores que podemos poner en `$modo` son:

- **`'a'`** : Modo añadir escritura, el texto se añade al que tenía el archivo existente.
- **`'w'`** : Modo borrar y escribir, se borra el texto anterior y se escribe el texto nuevo en su lugar.
- **`'r'`** : Modo lectura, para leer el texto del archivo.
- **`'a+'`** : Modo añadir escritura + lectura, como el modo 'a' pero también permite la lectura.
- **`'w+'`** : Modo borrar y escribir + lectura, como el modo 'w' pero también permite la lectura.
- **`'r+'`** : Modo lectura + escritura, como el modo 'w', pero también acepta escritura como en el modo 'w'

7.1.1. Crear un archivo.

Para crear un archivo sólo hay que abrir el archivo con `fopen()` indicando como primer argumento la ruta con el nombre del nuevo archivo. Después utiizaremos el método `fwrite()` para escribir en el archivo. Veamos un ejemplo:

```
<?php
//datos para pasar al archivo:
$nombre="Santiago";$apellido=Gonzalez;$email="santigonzi@hotmail.com";
$tel="600145588";
/*utilizamos el nombre y apellido como nombre del archivo. Este se guarda en
la carpeta "archivos"*/
$manejador=fopen("archivos/".$nombre.$apellido.".txt","w");
$datos ="Nombre: ".$nombre."; Apellido: ".$apellido."; E-Mail: ".$email.";
Teléfono: ".$tel;
fwrite($manejador,$datos); //introducimos los datos;
fclose($manejador); //cerramos el archivo.
?>
```

Para introducir los datos utilizamos la función `fwrite()`. Como primer argumento pasamos el manejador, y como segundo los datos a introducir.

En la carpeta *archivos*, que tenemos que haber creado previamente, tras ejecutarse el código encontraremos el nuevo archivo *SantiagoGonzalez.txt*, y su contenido será el siguiente:

```
Nombre: Santiago; Apellido: Gonzalez; E-Mail: santigonzi@hotmail.com;
Teléfono: 600145588
```

Aquí hemos utilizado datos que hemos creado nosotros, pero también podemos guardar datos provenientes de las respuestas de un formulario. De esta forma podemos guardar datos de los usuarios.

También podemos utilizar la función `fputs()` que funciona exactamente igual que `fwrite()`.

7.1.2. Leer el archivo.

Para leer el archivo, utilizaremos la función fgets() una vez abierto el archivo. Esta función lee el archivo línea a línea. Como el archivo anterior sólo tiene una línea, basta con ponerla una vez.

Pero antes de eso comprobaremos que el archivo realmente existe, para ello utilizaremos la función:

```
file_exists($ruta) or die("El archivo buscado no existe");
```

La función `file_exists()` tiene como argumento la ruta del archivo y comprueba si este existe, Detrás ponemos la intrucción `or die()`, que nos da un texto alternativo en caso de que la función no exista. Esta función no necesita tener el archivo abierto para que funcione.

```
<?php
//Comprobar si el archivo existe
file_exists("archivos/SantiagoGonzalez.txt") or die ("Error: El archivo no
existe.");
$manejador=fopen("archivos/SantiagoGonzalez.txt",'r'); //abrir archivo
$ver = fgets($manejador); //leer archivo
fclose($manejador); //cerrar archivo
echo $ver;
?>
```

Observa que para crear o modificar el archivo lo hemos abierto en modo escritura (w), mientras que para leerlo lo hemos abierto en modo lectura (r).

El resultado del código anterior será el texto siguiente:

Nombre: Santiago; Apellido: Gonzalez; E-Mail: santigonzi@hotmail.com; Teléfono: 600145588

7.1.3. Borrar archivos

Para borrar archivos utilizaremos la funciòn `unlink()`. Como argumento pasaremos el archivo a borrar.

En este ejemplo comprobamos si el archivo existe antes de borrarlo.

```
<?php
$ruta="archivos/SantiagoGonzalez.txt"; //ruta del archivo
file_exists($ruta) or die ("no existe el archivo indicado"); /*comprobar si
existe*/
unlink($ruta); //borrar archivo
echo "tu archivo ha sido borrado.";
?>
```

7.1.4. Leer archivos de más de una línea

Al leer el archivo con la función `fgets()`, leemos sólo la primera línea del archivo. Para leer la segunda línea deberemos utilizar otra vez la función `fgets()`, y así sucesivamente hasta leer todas.

Esto se debe a que al abrir el archivo se crea un puntero interno. Este al abrir el archivo apunta al primer carácter del archivo. La función `fgets()` Localiza el puntero interno. Lee la línea en la que está, y lo coloca en la siguiente línea.

Sabiendo esto veamos cómo leer un archivo con varias líneas. Para ello pondremos un ejemplo:

Creamos primero un archivo nuevo:

```
<?php
$nombre="Joaquín";$apellido="Rodríguez";$email="jkinrodri@gmail.com";
$tel="688211454";
$datos="<p>Nombre: ".$nombre."</p>\n<p> Apellido: ".$apellido."</p>\n<p>
       E-Mail: ".$email."</p>\n<p> Teléfono: ".$tel."</p>";
$manejador=fopen("archivos/".$nombre.$apellido.".html","w");
fwrite($manejador,$datos);
fclose($manejador);
?>
```

Hasta aquí hemos seguido los mismos pasos que en el ejemplo anterior, las diferencias son que en lugar de archivo de texto hemos escrito un archivo HTML, y para ponerlo en distintas líneas hemos utilizado el carácter de escape `\n`.

si abrimos este nuevo archivo con el editor de textos comprobaremos que cada dato está dentro de una etiqueta de párrafo y ocupa una línea distinta.

Para poder leer este archivo necesitamos que la función `fgets()` lea todas las líneas del archivo, y no sólo la primera. Para ello utilizamos un bucle `while` que comprueba si la línea a la que apunta el puntero interno existe, y de ser así la lee.

```
<?php
$manejador=fopen("archivos/JoaquinRodríguez.html","r");
//abrimos el archivo en modo lectura
while ($ver=fgets($manejador)) { //si la línea existe la condición se cumple
      echo $ver; //ver en pantalla la línea
      }
fclose($manejador); //cerramos el archivo
?>
```

El resultado del ejemplo anterior nos dará el siguiente texto (aquí resaltado en azul):

Nombre: Joaquín

Apellido: Rodríguez

E-Mail: jkinrodri@gmail.com

Teléfono: 688211454

7.2. El puntero interno

Tenemos ahora el problema de manejar el puntero interno. Éste al crearse apunta hacia el primer carácter. Va avanzando al ir leyendo el archivo y llega al final. Una vez en el final, éste no vuelve al principio, a no ser que se le especifique mediante una función.

Las siguientes funciones permiten controlar el puntero interno. Todas ellas deben utilizarse con el archivo abierto y su primer argumento es el manejador.

Funciones que controlan el puntero interno.

Nombre	Código y explicación
fgetc()	`$caracter = fgetc($manejador);` Lee el carácter al que apunta el puntero interno. El puntero avanza después una posición hacia el siguiente carácter.
ftell()	`$num = ftell($manejador);` Devuelve un número que indica la posición del puntero interno.
fseek()	`fseek($manejador,$avance,$num_desde);` Permite mover el puntero interno. el 2º argumento `$avance` indica el número de posiciones que avanza el puntero. El tercer argumento es opcional, e indica la posición desde la que debe empezar a avanzar el puntero.
rewind()	`rewind($manejador);` Coloca el puntero interno al principio del archivo.
fgetcsv()	`$array = fgetcsv($manejador,$num,"separador");` Devuelve un array en el qe el 2º argumento indica hasta dónde avanzar el puntero interno para formar los elementos del array. El 3º argumento indica el carácter que separará los elementos del array. Si el número del 2º argumento es superior al número de caracteres del archivo, éste se tomará hasta el final.

Nombre	Código y explicación
feof()	`$comprobar = feof($manejador);` Indica si el puntero interno ha llegado al final del archivo. En ese caso devuelve true, si no es así devuelve false.

7.3. Funciones con el archivo cerrado

Las siguientes funciones sirven para manejar archivos pero no necesitan tener abierto el archivo mediante `fopen()` para poder utilizarlas.

Funciones con archivo cerrado

Nombre	Código y explicación
copy()	`copy($ruta_origen,$ruta_destino);` Copia un archivo ubicado en `$ruta_origen` en la `$ruta_destino`. El 1º argumento es la ruta para localizar al archivo, y el 2º la ruta del nuevo archivo que se crea.
rename()	`rename("ruta_origen","ruta_destino");` Mueve (cambia de sitio) un archivo ubicado en `$ruta_origen` a la `$ruta_destino`. El 1º argumento es la ruta para localizar al archivo, y el 2º la ruta de la nueva ubicación del archivo.
file_get_contents()	`$miarchivo = file_get_contents($ruta);` Lee el archivo completo. Como argumento pasamos la ruta el archivo.
file()	`$array = file($ruta);` Crea un array que lee el contenido del archivo. Cada uno de los elementos del array está delimitado por los saltos de línea en el archivo. Como argumento pasamos la ruta del archivo.
fileatime()	`$fecha = fileatime($ruta);` Devuelve la fecha (en tiempo Unix) de la última vez que se accedió al archivo.
filectime()	`$fecha = filectime("ruta_archivo");` Devuelve la fecha (en tiempo Unix) de la última vez que se modificó el archivo.
filesize()	$tamano = filesize($ruta); Devuelve el tamaño en bytes del archivo.
filetype()	`$tipo = filetype($ruta);` Devuelve el tipo MIME de archivo, por ejemplo "file", si es un archivo de texto.
fileperms()	`$num = fileperms($ruta);` Devuelve un número que indica los permisos del archivo.
fileowner()	`$propiedad = fileowner($ruta);` Devuelve el nombre del dueño del archivo, normalmente el mismo que hay en la carpeta donde fue creado.
is_executable()	`$comprobar = is_executable($ruta);` Devuelve un booleano. Verdadero si el archivo puede ejecutarse, o falso en caso contrario.
is_readable()	`$comprobar = is_readable($ruta);` Devuelve un booleano. Verdadero si el archivo puede leerse, o falso en caso contrario.
is_writable()	`$comprobar = is_writable($ruta);` Devuelve un booleano. Verdadero si el archivo puede escribirse, o falso en caso contrario.

7.4. Directorios

PHP permite también crear y borrar carpetas del sitio. Para ello tenemos las siguientes funciones. Todas ellas tienen como argumento la ruta donde está la carpeta.

Funciones con directorios

Nombre	Código y explicación
is_dir()	`$comprobar = is_dir($ruta);` Comprueba si la carpeta existe. Devuelve el booleano verdadero si existe, o falso si no existe.
mkdir()	`mkdir($ruta);` Crea una nueva carpeta en la ruta con el nombre especificado. Si la carpeta ya existe no hace nada y devuelve el booleano falso.
mrdir()	`mrdir($ruta);` Borra la carpeta especificada siempre que ésta exista y esté vacía. En caso contrario devuelve `false` y da un mensaje de error.

7.4.1. Abrir directorios.

Al igual que con los archivos, hay una serie de funciones que necesitan de un método que deje abierto el directorio, para poder aplicarlas. Esto lo hacemos de la siguiente manera:

```
$ruta="archivos"; //ruta de la carpeta
$manejador=opendir($ruta); //abrir archivo
// incluir funciones con el archivo abierto
closedir($manejador); //cerrar el archivo
```

Con la función `opendir()` abrimos el directorio. Una vez abierto escribimos el codigo para leer o modificar el directorio. Después lo cerramos con la función `closedir()`. Fíjate que aquí también nos hace falta un "manejador".

Con el directorio abierto podemos utilizar las siguientes funciones:

`readdir($manejador);` Permite desplazarse por los distintos elementos dentro del directorio. Devuelve el nombre del archivo o carpeta a la que apunta el puntero interno. Una vez leído el puntero avanza una posición hacia el siguiente elemento.

El orden de los elementos no está definido, aunque suele ser el mismo que tienen dentro del directorio.

Para ver todos los elementos de un directorio utilizaremos un código parecido a este:

```
$manejador=opendir("documentos/archivos"); //abrir directorio
while ($archivo=readdir($manejador)) { /*Si hay archivo el bucle sigue, si no
se para.*/
      $archivos.=$archivo." ,<br/>"; //añadir a archivos encontrados
      }
closedir($manejador); //cerrar directorio.
echo $archivos; //leer resultados.
```

`rewinddir($manejador);` : Esta función coloca el puntero interno al principio del directorio, de manera que podamos volver a trabajar con todos los elementos.

7.4.2. Otras funciones para directorios

Las siguientes funciones se usan también con directorios y no requieren tener el archivo abierto.

Otras funciones con directorios

Nombre	Código y explicación
getcwd()	`$ruta = getcwd();` Devuelve la ruta del directorio actual, esta función no tiene argumentos.
scandir()	`$array = scandir($ruta);` Devuelve en un array todos los archivos y subdirectorios que están en el directorio pasado en el argumento.
disk_free_space()	`$espacio = disk_free_space($ruta);` Devuelve el espacio máximo disponible en bytes, que puede usarse en ese directorio. Puede usarse también con ficheros si pasamos en el argumento la ruta del fichero.

8. Bases de datos

8.1. Definición

Una base de datos es el lugar donde se guardan una serie de datos de manera organizada. No sólo consiste en guardar los datos, sino también en poder recuperarlos para trabajar con ellos: recuperarlos, cambiarlos, crear datos nuevos, etc.

Las bases de datos organizan los datos en tablas, Las tablas suelen estar relacionadas entre sí, de manera que los datos de una tabla tengan relación con los de otra.

Hay distintos programas de gestión de base de datos, tales como Oracle, Microsoft Access, etc. Sin embargo MySQL es el que mejor se adapta al lenguaje PHP.

8.2. Bases de datos en MySQL

MySQL es un lenguaje en el que podemos crear y administrar las bases de datos en formato ".sql". Este lenguaje es compatible con PHP, de modo que podemos utilizarlo dentro de una página web para trabajar con la base de datos.

Con el programa "phpMyAdmin" podemos crear bases de datos y trabajar con ellas de una manera gráfica; estas bases de datos podemos usarlas luego con PHP. Sin embargo con PHP podemos también crear y modificar bases de datos, y trabajar directamente con ellas, sin usar "phpMyAdmin".

8.3. Conceptos básicos

Para los que no estén muy familiarizados con las bases de datos incluimos aquí unos cuantos conceptos básicos:

- **Tabla:** Es la estructura donde se guardan los datos. La tabla consiste en una serie de filas y de columnas. Una base de datos puede tener una o varias tablas, y éstas pueden estar o no relacionadas entre sí.
- **Cabecera:** Es la primera fila de la tabla. En esta se indican los tipos de datos que van a guardarse (ej.: nombre / apellidos / teléfono / email / ...).
- **Campo:** Cada columna de la tabla es un campo, y guarda un tipo de dato distinto.
- **Registro** Cada fila de la tabla (excepto la cabecera) es un registro. En ella se guardan los datos correspondientes a una única entidad (persona, producto, etc.).
- **Celda:** Las celdas son la confluencia de cada fila (entidad) con cada columna (campo).

- **Dato:** El dato es la información que se guarda en la celda. En cada celda hay un sólo dato, y cada dato no puede ocupar más de una celda.

Por último tenemos también las relaciones entre tablas. Las tablas no son independientes unas de otras y los registros de unas se relacionan con los de otras. Por ejemplo en una base de datos de obras literarias, tendremos una tabla con los autores y otra con las obras. Al relacionar estas tablas los registros de la tabla "autores" debemos relacionarlos con los de la tabla "obras", de manera que a cada autor le correspondan sus obras.

Las formas de relacionar una tabla con otra pueden ser:

- **Relación uno a varios**: a cada registro de la tabla A le corresponden varios registros de la tabla B.
- **Relación uno a uno**: a cada registro de la tabla A le corresponden un sólo registro la tabla B.
- **Relación varios a varios**: a cada registro de la tabla A le corresponden varios registros de la tabla B y a cada registro de la tabla B le corresponden también varios registros de la tabla A

8.4. phpMyAdmin

El programa phpMyAdmin viene integrado en el paquete XAMPP con el que hemos instalado PHP.

Este programa consiste en una serie de páginas con las que podemos manejar las bases de datos: crearlas, modificarlas, cambiar su contenido, y todo tipo de operaciones dentro de la base de datos.

Para acceder a él desde la página principal de XAMPP, escribimos *localhost* en la barra de navegación, y en el menú de la izquierda, en la sección *Herramientas* pulsamos en *phpMyAdmin*. También podemos acceder desde la dirección: *localhost/phpmyadmin/*.

A partir de aquí podemos crear nuevas bases de datos o abrir alguna de las existentes, para trabajar con ellas.

Dentro de cada base de datos podemos crear tablas. Podemos añadir o quitar campos (columnas) a una tabla. y añadir o quitar o modificar registros.

En cada campo se debe indicar el tipo de datos que queremos guardar (texto, números, fechas, etc.), y además los campos pueden tener una serie de propiedades (clave primaria, clave única, autoincremeto, etc.). Como también podemos manejar las bases de datos desde la página web con PHP, iremos viendo todo esto a lo largo de este tema.

No vamos a extendernos más sobre phpMyAdmin en este resumen. El tema 13 de este manual explica la forma de trabajar con phpMyAdmin, nos remitimos a él en la página *http://aprende-web.net/php/php13_1.php* y siguientes.

Las bases de datos que creamos con phpMyAdmin son del tipo MySQL. Podemos una vez creadas trabajar con ellas también desde una página web.

8.5. Conectar con MySQL

8.5.1. Datos necesarios

Para trabajar con una base de datos en MySQL desde una página en PHP debemos saber los datos que nos permiten localizar y conectar con MySQL. Estos son:

- **Servidor** : Nombre de la máquina o servidor donde está MySQL
- **Usuario**: Dentro de MySQL puede haber varios usuarios, cada uno de ellos tendrá un nombre diferente.
- **Contraseña**: Cada usuario tiene una contraseña como medida de seguridad.

- **Base**: Como cada usuario puede tener más de una base de datos, cada una de ellas debe tener un nombre diferente.

Si trabajamos en local (con XAMPP), y no hemos modificado los datos en phpMyAdmin los datos serán los siguientes:

Servidor = "localhost"; usuario = "root"; contraseña = "". El nombre de la base será el que nosotros le hayamos dado al crearla. Esta base de datos sólo la podremos utilizar con el servidor local.

Si trabajamos con otro servidor en la web, éste debe proporcionarnos estos datos, los cuales, serán distintos de los del servidor local.

La mayoría de servidores web con PHP nos permiten crear una base de datos en el mismo servidor en que tenemos alojado el sitio. Normalmente tenemos que ir al panel de control que nos proporciona el servidor y buscar la opción de crear una base de datos. Al crearla el servidor nos proporcionará las claves. También suelen tener la opción de utilizar phpMyAdmin con la base de datos.

Si hemos creado la base de datos en el servidor local y queremos pasarlo al servidor web podemos utilizar las opciones *exportar* e *importar* de phpMyAdmin. Primero exportamos la base a nuestro ordenador en un archivo, y después la importamos al servidor desde ese archivo.

Como los datos son distintos en local y en web, debemos poner distintos datos en un sitio y en otro. Al subir la página debemos tener esto en cuenta. Podemos solucionar esto mediante una estructura condicional que detecte el nombre del servidor. Según el servidor que usemos tendremos unos datos u otros. El código será como éste:

```
<?php
$modo=$_SERVER['HTTP_HOST']; //localizamos el servidor
if ($modo=="localhost") { //datos en servidor local
    $servidor="localhost";
    $usuario="root";
    $contrasena="";
    $base="nombre_base";
    }
else { //datos en servidor web
    $servidor="nombre_servidor_web";
    $usuario="nombre_usuario_web";
    $contrasena="contraseña_em_web";
    $base="nombre_base_web";
    }
?>
```

Para mayor comodidad podemos guardar este código en un archivo (por ej.:*conexion.php*) y incluirlo en la página antes de realizar cualquier operación con bases de datos mediante:

```
include("conexion.php");.
```

8.5.2. Realizar la conexión

Para conectar con la base de datos utilizamos la siguiente función.

```
$db=mysql_connect($servidor,$usuario,$contrasena);
```

La conexión con MySQL está abierta. La variable donde guardamos la función es un "manejador", y deberá usarse en algunas funciones que utilicemos con la conexión abierta, así como para cerrar la conexión.

El siguiente paso es localizar la base de datos. Utilizamos la función:

```
mysql_select_db("nombre_base",$db);
```

Con esto la conexión a la base de datos indicada en el primer argumento queda abierta. El segundo argumento es el manejador.

Una vez conectados a la base de datos podemos trabajar con ella, para ello MySQL tiene su propio lenguaje, en el cual le diremos lo que queremos hacer en la base de datos, éste lo iremos viendo a lo largo de este tema. De momento una vez creado este código lo guardamos en una variable:

```
$sql="codigo_MySQL";
```

Donde pone `codigo_MySQL` insertaremos el código en lenguaje MySQL necesario para realizar las operaciones con la base de datos.

Mandamos el código MySQL a la base de datos, es lo que se llama enviar una petición. Utilizamos la siguiente función:

```
$datos=mysql_query($sql,$db);
```

La función `mysql_query()` envía el código en MySQL a la base de datos y devuelve una respuesta en caso de que la hayamos pedido (por ejemplo podemos pedir que devuelva algunos datos de la base). La variable `$datos` recoge esa respuesta.

Una vez realizadas las operaciones anteriores cerramos la conexión a MySQL, para ello utilizamos la función:

```
mysql_close($db);
```

El código completo para realizar una conexión es el siguiente:

```
$db = mysql_connect($servidor,$usuario,$contrasena); //conectar con MySQL.
   mysql_select_db("nombre_base",$db); //Conectar con la base de datos.
   $sql = "codigo_MySQL"; //Código MySQL con órdenes para la base de dados.
   $datos = mysql_query($sql,$db); //Enviar petición y recibir respuesta.
mysql_close($db); //cerrar la conexión.
```

Este código PHP es el que nos permite trabajar con una base de datos, y es casi siempre el mismo, por lo que podemos guardarlo en una función a la que le pasamos como argumento el código MySQL y nuestros datos. La función nos devolverá los datos pedidos a la base.

En nuestro caso podría ser así:

```
function conecta_db($conexion,$codigo) {
   include($conexion);
   $db = mysql_connect($servidor,$usuario,$contrasena);
   mysql_select_db($base,$db);
   $sql = $codigo;
   $datos = mysql_query($sql,$db);
   return $datos;
   }
```

Se supone que hemos hecho previamente un archivo con los datos de conexión, parecido al que hemos visto anteriormente. La ruta a este archivo será el primer argumento, la cual recogemos en la variable `$conexion`.

Como segundo argumento pasaremos el código MySQL que queremos enviar. Al llamar a la función ésta realizará las acciones que indica el código MySQL y nos devolverá los datos que le hayamos pedido.

```
$sql = "codigo en MySQL";
$datos = conecta_db("conexion.php",$sql);
```

8.6. Crear una base de datos

Podemos crear una nueva base de datos desde PHP. En este caso el código difiere un poco del anterior, ya que no llamamos a ninguna base de datos, y en su lugar creamos una con el siguiente código MySQL:

```
$sql="create database `nombreBase`";
```

El código MySQL ponemos las palabras `create database` seguido del nombre que le queremos dar a la nueva base. Aquí lo hemos puesto entre los signos ` ` . Esto no es obligatorio y podemos escribirla directamente sin los signos de acento grave. Sin embargo es necesario si el nombre tiene caracteres comprometidos (espacios en blanco, signos de puntuación, etc.).

El código PHP completo para crear una base de datos será:

```
<?php
$db=mysql_connect($servidor,$usuario,$contrasena);
$sql="create database `miagenda`";
mysql_query($sql,$db);
mysql_close($db);
?>
```

Como no recogemos ningún dato de la base no necesitamos que la función `mysql_query()` esté igualada a una variable.

Para comprobar que la base de datos se ha creado podemos abrir phpMyAdmin y comprobar que la base de datos está ahí.

Este código funciona siempre con el servidor local, sin embargo con los servidores web muchas veces no nos permiten crear las bases de datos de este modo, y tenemos que crearlas directamente desde su panel de control. En el servidor local (XAMPP) también tenemos la opción de crearla con phpMyAdmin.

8.7. Incluir tablas

Una vez creada la base de datos, debemos incluir tablas. Esto lo podemos hacer también desde PHP. Aquí utilizaremos la función `conecta_db()` que hemos creado antes. Siguiendo el ejemplo anterior El código MySQL para crear una nueva tabla es :

```
$sql = "create table `agenda` (`IDagenda` int(6) not null
        auto_increment primary key)";
```

En primer lugar ponemos las palabras clave `create table` seguido del nombre de la tabla. Ésta, al igual que al crear la base, puede ir entre signos de acentos o no.

El lenguaje MySQL no diferencia entre mayúsculas y minúsculas; pero en los nombres de los elementos (tablas, bases, campos, etc) sí que se diferencia.

Después y entre paréntesis ponemos el nombre del primer campo de la tabla seguido de sus propiedades. Debemos poner al menos un campo. Si ponemos más de uno los separaremos con comas.

En el código incluimos primero la función `conecta_db()` creada anteriormente, y después ponemos:

```
$ruta = "conexion.php";
$sql = "create table `agenda` (`IDagenda` int(6) not null au-
to_increment primary key)";
conecta_db($ruta,$sql);
```

Debemos también haber guardado el archivo *conexion.php* en la misma carpeta.

8.8. Propiedades de los campos

En el ejemplo anterior hemos puesto un campo llamado `idAgenda`, con varias propiedades. Éstas son:

- **`int(6)`**: La primera propiedad es la única que es obligatoria, e indica el tipo de elemento que debe incluirse en el campo. Aquí indica que es un número entero igual o menor de 6 cifras.
- **`not null`**: Indica que el campo no puede quedarse vacío. Siempre tiene que tener un dato.
- **`auto_increment`**: Auto incremento. Cada vez que creemos un nuevo registro se crea automáticamente el dato, que será el número del anterior incrementado en una unidad.
- **`primary key`**: Clave primaria. Cada tabla sólo puede tener una clave primaria, y es el campo de referencia para relacionarse con otras tablas. Los datos, dentro de este campo no pueden estar repetidos.

Distinguimos entre el tipo de datos (única propiedad obligatoria y que debemos poner la primera) y las demás propiedades. Vemos aquí los **tipos de campo** más comunes, aunque no son los únicos:

- **`varchar(n)`**: Cadena de un máximo de 255 caracteres. `(n)` indica el número máximo de caracteres que le asignamos.
- **`text`**: Cadena larga con un máximo de 65.536 caracteres.
- **`longtext`**: Cadena muy larga con un máximo de 4.294.967.295 caracteres.
- **`int(n)`**: Número entero. entre paréntesis indicamos el número máximo de dígitos.
- **`float(n,d)`**: Número decimal. en "n" indicamos el número máximo de dígitos, y en "d" el número máximo de decimales.
- **`double(n,d)`**: Número decimal. "n" y "d" tienen la misma función que en el anterior. La diferencia es que aquí soporta un número mucho mayor que el anterior.
- **`date()`**: Fechas en formato AAAA-MM-DD.
- **`datatime()`**: Fecha y hora en formato AAAA-MM-DD HH: MM: SS.
- **`datatime()`**: Fecha y hora en formato AAAA-MM-DD HH: MM: SS.
- **`timestamp()`**: Marca de tiempo Unix
- **`time()`**: Hora en formato HH: MM: SS.

Después del tipo de dato podemos poner otras **propiedades no obligatorias**. Además de las que hemos visto en el ejemplo anterior podemos poner las siguientes:

- **`default 'dato'`**: Es el 'dato' por defecto que tomará la celda si al crear el registro no se le asigna ninguno.
- **`unique`**: clave única. Indica que ningún dato puede estar repetido en este campo.
- **`foreign key`**: Clave foránea. Es el campo hacia el que apunta una relación desde otra tabla.

8.9. Anadir campos a una tabla

Podemos añadir más campos a una tabla mediante la siguiente instrucción en MySQL:

```
$sql="alter table `nombre_tabla` add `nuevo_campo` propiedades, add
      `nuevo_campo` propiedades";
```

El resto del código PHP es el visto anteriormente. Nosotros utilizaremos la función `conecta_db()` para insertar más campos. Ejemplo:

```
$ruta = "conexion.php";
$sql = "alter table `agenda`
        add `nombre` varchar(50) not null default '',
        add `telefono` int(9) not null default '000000000',
        add `email` varchar(100) not null default '---@---',
        add `descripcion` text not null default '' ";
conecta_db($ruta,$sql);
```

Podemos comprobar que los nuevos campos se han creado con phpMyAdmin, donde podemos ver en la base de datos la tabla con los nuevos campos.

8.10. Insertar registros.

Para insertar registros seguimos usando el mismo código php, que nosotros hemos guardado en la función `conecta_db()`. Lo único que varía es el código MySQL que le pasamos. Este tiene la siguiente sintaxis:

```
$sql = "insert into `nombre_tabla`
       (`campo1`, `campo2`, `campo3`)
       value ('dato1', 'dato2', 'dato3');";
```

En primer lugar ponemos la instrucción `insert into` seguido del nombre de la tabla. Después entre paréntesis ponemos los nombres de los campos en los que insertaremos los datos. Estos van separados por comas. A continuación ponemos la instrucción `value` seguida de otro paréntesis. En este paréntesis pondremos los datos a insertar en los campos. Éstos irán entre comillas simples y separados por comas.

Los datos se corresponden correlativamente con los campos, el primer campo indicado recibe el primer dato, el segundo campo recibe el segundo, y así sucesivamente.

Hay que poner el mismo número de campos que de datos. De otra manera el registro no se crea.

Si hay algún campo con clave única que esta duplicado, o algún campo con `not null` no relleno, el registro no se creará.

No es necesario poner todos los campos de la tabla. Los que no pongamos cogerán su valor por defecto o se quedarán sin valor.

Podemos pasar los nombres de tablas, campos y los datos en variables de php. En este caso las comillas y los acentos (si los hubiera) debemos seguir manteniéndolos.

Siguiendo con el ejemplo anterior crearemos un registro para la tabla `agenda`:

```
$ruta="conexion.php";
$sql = "insert into `agenda`
  (`nombre`, `telefono`, `email`, `descripcion`)
  value ('Vicente Gracia','685138554','vicentegracia@msn.com','amigo')";
conecta_db($ruta,$sql);
```

Puedes comprobar si el registro se ha creado en phpMyAdmin.

8.11. Cambiar datos de un registro.

Una vez creado un registro podemos cambiar los datos de un campo sin tener que borrar el registro y crear uno nuevo. La sintaxis es la siguiente:

```
$sql = "update `tabla` set `campo2` = 'dato2', `campo3`='dato3'
        where`campo1` = 'dato1'";
```

Los datos que variamos aquí son los del campo2 y campo3. El campo1 es el campo de referencia para buscar el registro. Esto lo veremos mejor con un ejemplo. En el registro creado anteriormente queremos cambiar el teléfono y el email. Lo haremos así:

```
$ruta="conexion.php";
$sql = "update `agenda` set `telefono` = '657889201',
        `email` ='vincengra@gmail.com'
        where `nombre` = 'Vicente Gracia'";
conecta_db($ruta,$sql);
```

8.12. Borrar registros

Para borrar un registro utilizaremos el siguiente código:

```
$sql= "delete from `tabla` where `campo1` = 'dato1'";
```

Se borrarán aquí todos los registros de `tabla` en los que se cumpla la condición de que `campo1` tenga como dato `dato1`.

Para asegurarnos de que no se borre más de un registro debemos elegir un campo con clave única o el de auto incremento.

8.13. Borrar tablas y bases.

Para borrar una tabla utilizaremos el siguiente código MySQL:

```
$sql = "drop table `nombre_tabla`";
```

Al borrar una tabla se borran también todos los datos que hay en ella, sin posibilidad de recuperarlos.

Para borrar una base de datos entera utilizaremos el siguiente código:

```
$sql = "drop database `nombre_base`";
```

Aquí el código PHP que utilizamos es el mismo que hemos utilizado para crear una base, en el cual cambiamos el código MySQL.

Al borrar una base de datos se borran todas las tablas y datos que hemos guardado en ella sin posibilidad de recuperarlos.

8.14. Leer datos de una tabla

Para leer los datos que hay en una tabla utilizaremos el siguiente código en MySQL:

```
$sql="select * from `nombre_tabla`";
```

Escribimos el código `select * from` seguido del nombre de la tabla.

Utilizamos después el mismo código PHP usado en los apartados anteriores, y que hemos incluido en la función `conecta_db()`. Para leer la tabla `agenda` creada anteriormente escribiremos:

```
$ruta = "conexion.php";
$sql = "select * from `agenda`";
```

```
$datos = conecta_db($ruta,$sql);
```

Los datos que pedimos están dentro de la variable `$datos`, pero esta variable no podemos leerla directamente ya que si intentamos sacarla en pantalla con la instrucción `echo`, nos dará un resultado parecido a esto:

Resource id #17

Para poder leerla PHP dispone de varias funciones. La más usada es `mysql_fetch_array()` a la que le pasamos como único argumento la variable obtenida `$datos`.

El resultado es un array a la vez indexado y asociativo en el que obtenemos el primer registro de la tabla.

Esto significa que en el array obtenemos el primer registro dos veces, una con array indexado, en el que los índices son el orden en que aparecen los campos, y otra como array asociativo, en el que la clave es el nombre del campo.

La función `mysql_fetch_array()` tiene un puntero interno que hace que una vez llamada salte hacia el siguiente registro, por lo que si volvemos a llamarla otra vez veremos el segundo registro. Cada vez que llamamos a la función el puntero interno avanza un registro, hasta que se leen todos.

8.14.1. Mostrar los nombres de los campos.

Para mostrar los nombres de los campos no necesitamos más que un registro, del cual extraemos las claves del array asociativo que se crea con `mysql_fetch_array()`

El código será parecido al siguiente.

```
$ruta = "conexion.php"; //archivo con datos de conexión a la base
$sql = "select * from `agenda`"; //código MySQL
$datos = conecta_db($ruta,$sql); //función de conexión con la base.
$campos=array(); //array donde guardaremos los datos.
$row=mysql_fetch_array($datos); //extraer datos de un registro
foreach ($row as $clave=>$valor) { //recorrer array de primer registro.
         if (is_string($clave)) { //sólo para array asociativo ...
            echo "$clave, ";  //extraemos la clave y la vemos.
            array_push($campos,$clave); //guardamos la clave en un array.
            }
         }
```

8.14.2. Leer los registros

La función `mysql_fetch_array()`sólo lee un registro, y mueve después el puntero interno hacia el siguiente registro. Necesitamos por lo tanto un bucle que repita esta función mientras que siga habiendo registros. Utilizaremos para ello el bucle `while`.

```
$ruta = "conexion.php"; //archivo con datos de conexión a la base
$sql = "select * from `agenda`"; //código MySQL
$datos = conecta_db($ruta,$sql); //función de conexión con la base.
while ($row=mysql_fetch_array($datos)) { //Bucle para ver todos los registros
      $nombre=$row['nombre']; //datos del campo nombre
      $telefono=$row['telefono']; //datos del campo teléfono
      $email=$row['email']; //datos del campo email
      echo "<p>$nombre, $telefono, $email. </p>"; //visualizar datos
       }
```

Dentro del bucle utilizamos los nombres de los campos para buscar los datos. Éstos son las claves del array `$row` que se crea en cada vuelta.

8.14.3. Ordenar registros.

Tenemos la opción de mostrar los registros ordenados, para ello variamos el código MySQL:

```
$sql="select * from `agenda` order by `nombre`";
```

Añadimos al final la instruccion `order by` seguido del nombre del campo que utilizamos para ordenar los registros.

Los registros aparecerán ordenados de forma ascendente, es decir si son números de menor a mayor, y si son textos, por orden alfabético.

También podemos ordenar los registros por orden descendente, incluyendo al final del código anterior la palabra `desc`.

```
$sql="select * from `agenda` order by `nombre` desc";
```

8.15. Buscar registros

Otra de las tareas frecuentes en una base de datos es buscar los datos pertenecientes a un mismo registro. Para buscar un registro debemos al menos saber un dato del mismo. Por ejemplo para buscar un teléfono debemos saber el nombre. El dato que sabemos es el dato de referencia, y los otros son los datos buscados.

Podemos hacer una búsqueda exacta o aproximada, dependiendo de si nos sabemos el dato de referencia de forma exacta o aproximada.

8.15.1. Búsqueda exacta.

En la búsqueda exacta debemos escribir el dato que conocemos de forma completa.

Para realizar una búsqueda seleccionamos primero la tabla entera, y después restringimos los registros mediante una condición. Este es el código en MySQL:

```
$sql="select * from `nombre_tabla` where `campo` = 'dato'"
```

Seleccionamos primero la tabla mediante `select * from` seguido del nombre de la tabla. Después marcamos la condición mediante `where` seguido del nombre del campo que conocemos, el signo igual, y el dato que conocemos.

El resto de código PHP es igual que para mostrar la tabla completa.

```
$ruta = "conexion.php"; //archivo con datos de conexión a la base
$sql = "select * from `agenda` where `nombre` = 'Vicente Gracia'";
       //código MySQL
$datos = conecta_db($ruta,$sql); //función de conexión con la base.
while ($row=mysql_fetch_array($datos)) { //Bucle para ver todos los registros
      $nombre=$row['nombre']; //datos del campo nombre
      $telefono=$row['telefono']; //datos del campo teléfono
      $email=$row['email']; //datos del campo email
      echo "<p>$nombre, $telefono, $email. </p>"; //visualizar datos
      }
```

El hecho de utilizar un bucle para una búsqueda es porque puede haber en la tabla más de un registro con el dato buscado. En este caso se mostrarían todos los registros que coinciden.

8.15.2. Búsqueda aproximada.

Es posible que no sepamos el dato de referencia de manera exacta. En este caso podemos hacer una búsqueda aproximada, lo único que cambia respecto de la anterior es el código MySQL:

```
$sql="select * from `tabla` where `campo` like 'dato%'";
```

Respecto al código anterior cambiamos el signo igual = por la palabra `like`, y en el dato podemos poner tanto delante como detrás el signo `%` que es un comodín que sustituye a cualquier cadena que pueda haber tanto por delante (si lo ponemos delante), como por detrás (si lo ponemos detrás).

De esta manera podemos buscar en los campos datos que tengan una cierta similitud con el que buscamos.

El resto del código PHP es exactamente igual que para la búsqueda exacta.

8.16. Más sobre MySQL

- El lenguaje SQL (o MySQL) tiene más posibilidades para trabajar con bases de datos. Para ver más consulta el manual de SQL en *http://aprende-web.net/progra/sql/sql_1.php*

8.17. Funciones para bases de datos

Además de las funciones vistas hasta ahora hay otra serie de funciones que pueden facilitarnos el trabajo con las bases de datos.

8.17.1. Funciones para consulta de datos

Las siguientes funciones se usan para consultar los datos extraídos.

- **`mysql_fetch_row($datos)`** : Se utiliza igual que `mysql_fetch_array()` con la única diferencia de que devuelve solamente el array indexado.
- **`mysql_fetch_assoc($datos)`** : Igual que la anterior pero esta vez devuelve solamente el array asociativo.
- **`mysql_data_seek($datos,num)`** : Cambia la posición del puntero interno. El segundo parámetro indica el lugar del puntero en la próxima consulta (se empieza a contar por el 0). La instrucción `mysql_data_seek($datos,0)` pone el puntero en la primera fila.
- **`mysql_fetch_length($datos)`** : devuelve un array indexado con la longitud de los datos de la última consulta con `mysql_fetch_array()` o alguna de las dos primeras funciones vistas en este apartado. Esta función no mueve el puntero interno.
- **`mysql_fetch_field($datos)`** : Devuelve un array con la información del campo (nombre, tipo, longitud, claves, etc.). Tiene un puntero interno que se mueve al siguiente campo si se le consulta de nuevo.
- **`mysql_num_rows($datos)`** : Devuelve el número de filas de la tabla extraídas en la consulta.
- **`mysql_num_fields($datos)`** : Devuelve el número de campos (columnas de la tabla) extraidos en la consulta.

8.17.2. Funciones de campos.

Con la conexión abierta:

- **`$datos=mysql_list_dbs($db)`** : Devuelve la lista de todas las bases de datos que hay en el servidor. Como parámetro pasamos el manejador. La lista viene en una serie de arrays que recorreremos con una función del tipo `mysql_fetch_array($datos)`.
- **`$datos=mysql_list_tables($base_datos,$db)`** : devuelve la lista de todas las tablas que hay en la base de datos especificada en el primer parámetro. El segundo parámetro es el manejador. La lista viene en una serie de arrays que recorreremos con una función del tipo `mysql_fetch_array($datos)`.

Información de conexión

Las siguientes funciones nos dan información sobre el programa que alberga las bases de datos y el tipo de alojamiento:

- **`$programa=mysql_get_client_info();`** : indica el programa usado para manejar las bases de datos (MySQL, u otro).
- **`$host=mysql_get_host_info();`** : indica el alojamiento del servidor que tiene las bases de datos (Si se trabaja en local : localhost via TCP/IP).
- **`$proto=mysql_get_proto_info();`** : indica la versión del protocolo usada en las bases de datos (Por ejemplo: 10).
- **`$server=mysql_get_server_info();`** : indica la versión de MySQL o el programa que usemos para manejar las bases de datos (Por ejemplo: 5.5.16).

8.17.3. Más funciones

Hemos visto aquí las principales funciones que se pueden usar con las bases de datos. Existen más funciones aunque no suelen ser muy habituales.

Puedes ver la lista completa de funciones de PHP para bases de datos con MySQL en *http://www.php.net/manual/es/book.mysql.php*

9. Objetos en PHP

9.1. La Programación Orientada a Objetos

La programación orientada a objetos es usada en muchos lenguajes de programación. Consiste en crear clases de objetos.

En cada clase de objetos se agrupan objetos de características similares. A cada clase se le asignan unas propiedades y unos métodos.

En programación, crearemos primero la clase de objetos, con sus métodos y propiedades, para crear después los objetos concretos que pertenecen a esa clase.

Las propiedades son cualidades o características que posee el objeto, y se definen en variables, mientras que los métodos son acciones que pueden realizarse con el objeto. Se definen mediante funciones.

9.2. Creación de clases

Para definir una clase de objetos, utilizamos la palabra reservada `class` seguida del nombre de la clase. Después, entre llaves escribimos las propiedades y los métodos de la clase.

```
class Productos { /*...*/ }
```

Aunque no es obligatorio, es conveniente poner el nombre de la clase con mayúscula, para distinguirla del resto de elementos.

9.3. Propiedades

Para definir las propiedades, dentro de las llaves escribiremos una serie de variables:

```
class Productos {
   var $nombre ="no_definido";
   var $precio = 0;
   var $descripcion;
   var $imagen = "objetos/producto.gif";
   }
```

Al definir las variables se crean las propiedades de esta clase de objetos.

Las propiedades pueden llevar un valor por defecto, que es el que les damos, o no llevar ninguno.

9.4. Instanciar un objeto

Instanciar un objeto es crear un objeto concreto de la clase que hemos creado, para ello escribimos:

```
$naranja = new Productos;
```

Hemos creado el objeto `$naranja` perteneciente a la clase `Productos`.

Este objeto tiene las propiedades y métodos que se hayan definido al crear la clase, con sus valores por defecto.

Para acceder a sus propiedades lo haremos de la siguiente manera:

```
echo $naranja->nombre;
```

Aquí hemos accedido a la propiedad en modo lectura, pero si queremos cambiarla, utilizaremos el modo escritura, en el cual le damos otro valor:

```
$naranja->nombre = "Naranja";
```

El código completo de este ejemplo, en el que instanciamos el objeto y le damos sus propiedades, es el siguiente:

```
$naranja=new Productos;
$naranja->nombre = "Naranja";
$naranja->precio = 1.25;
$naranja->descripcion = "Naranja. Origen Valencia. clase Navelina.";
$naranja->imagen = "objetos/naranjas.gif";
```

9.5. Métodos

Un método es una función en la que pedimos al objeto que haga algo. Por ejemplo podemos hacer una función que muestre en pantalla el producto del ejemplo anterior.

La función se escribe dentro de las llaves que ponemos al crear el objeto. Dentro de la función podemos utilizar las propiedades del objeto, y para referirnos al objeto utilizaremos la variable `$this`. En este ejemplo modificamos la creación del objeto anterior:

```
class Productos {
   var $nombre ="no_definido";
   var $precio = 0;
   var $descripcion;
   var $imagen = "producto.jpg";
   function mostrar(){
     echo "<img src='$this->imagen' style='float: left; height: 80px; margin-
right: 10px;'>";
     echo "Nombre = $this->nombre<br/>";
     echo "Precio = $this->precio<br/>";
     echo "Descripción = $this->descripcion.<br/>";
  }
}
```

En color rojo oscuro hemos destacado el código nuevo que hemos añadido al que ya teníamos, el cual consiste en la función que crea el método.

Para utilizar el método en un objeto ya instanciado de la clase lo haremos de la siguiente manera.

```
$naranja->mostrar();
```

El método tendrá el mismo nombre que la función. Si la función necesita argumentos deberemos ponerlos también en el método.

9.6. Función constructora

Una función constructora permite que al instanciar un objeto podamos indicar directamente sus propiedades, sin tener que definirlas luego.

Para ello al crear la clase debemos poner dentro de las llaves la función constructora. Veamos una función constructora en el ejemplo anterior.

```
class Productos {
   function Productos($n="no definido",$p=0,$d="",$i="producto.jpg") {
    $this->nombre=$n;
    $this->precio=$p;
    $this->descripcion=$d;
    $this->imagen=$i;
  }
   function mostrar(){
     echo "<img src='$this->imagen' style='float: left; height: 80px; margin-
right: 10px;'>";
     echo "Nombre = $this->nombre<br/>";
     echo "Precio = $this->precio<br/>";
     echo "Descripción = $this->descripcion.<br/>";
  }
}
```

Hemos quitado el código en el que hemos creado las propiedades y lo hemos sustituido por la función constructora (destacada en color rojo oscuro).

En la función los argumentos que pongamos serán las propiedades, a las cuales les damos un valor por defecto. Después mediante la fórmula `$this->propiedad = $argumento` creamos las propiedades de la clase.

La ventaja está en que ahora al instanciar un nuevo objeto podemos darle directamente las propiedades. Por ejemplo:

```
$nombre="Pimiento verde";
$precio=1.40;
$descripcion="Pimiento verde tipo Italiano. Origen: La Rioja.";
$imagen="objetos/pimiento.jpg";
$pimiento=new Productos($nombre,$precio,$descripcion,$imagen);
```

Definimos primero las propiedades en variables, y luego las pasamos al instanciar el objeto en forma de argumentos. También podríamos haber introducido los datos directamente en los argumentos:

```
$pimiento = new Productos("Pimiento verde",1.4,"Pimiento verde tipo
            Italiano. Origen: La Rioja.", "objetos/pimiento.jpg");
```

El orden en que se pasan los argumentos debe ser siempre el mismo, ya que a cada argumento le corresponde una propiedad.

9.7. Subclases

Dentro de una clase de objetos podemos crear una o varias subclases. Éstas normalmente especifican el tipo de objeto dentro de la clase.

Cada subclase puede tener propiedades y métodos específicos, además de las propiedades y métodos de la clase general.

Para definir una subclase lo haremos mediante:

```
class Subclase extends Clase_general { .. }
```

Creamos la subclase con la palabra `class` seguida del nombre de la clase, seguimos con la palabra `extends` y el nombre de la clase a la que pertenece. Después entre llaves pondremos las nuevas propiedades y métodos específicos de la subclase.

Siguiendo el ejemplo anterior crearemos una subclase de la clase `Productos`

```
class Frutas extends Productos {
    var $clase="no definida";
    var $origen="indefinido";
    }
```

9.7.1. Herencia y polimorfismo

La **herencia** consiste en que todas las propiedades y métodos de la clase general se conservan en las subclases secundarias. De esta manera los objetos instanciados desde las clases secundarias tendrán los métodos y propiedades generales y también los específicos de su clase.

El **polimorfismo** consiste en que podemos modificar una propiedad o un método de la clase general en una secundaria, de manera que sólo afectará a los objetos instanciados desde la clase secundaria, mientras que los instanciados desde la clase general u otras subclases no les afecta el cambio.

9.7.2. Cambiar propiedades o métodos por defecto en subclase

Para cambiar una propiedad o un método dentro de una subclase volveremos a definir la propiedad o método dentro de la subclase. Supongamos que en el ejemplo anterior queremos cambiar el método `mostrar()` de manera que se muestren también las nuevas propiedades específicas. También cambiaremos el valor por defecto de la propiedad `nombre`

Para no tener que repetir el código de la función de la clase general haremos un referencia a ella dentro de la función de la sublcase mediante: `parent::nombre_funcion()`. Este es el código:

```
class Frutas extends Productos {
  var $clase="no definida";
  var $origen="indefinido";
  var $nombre="Frutas";
  function mostrar(){
     parent::mostrar();
     echo "Clase : $this->clase<br/>";
     echo "Origen : $this->origen<br/>";
     }
  }
```

El método `mostrar()` amplia el código del que teníamos en la clase general, y ahora muestra también las propiedades específicas

Hemos cambiado también el valor por defecto de la propiedad `nombre` dentro de esta subclase. Sin embargo esto sólo funciona si hemos definido las propiedades en la clase general directamente, y no con la función constructora.

De igual manera podemos definir nuevos métodos que se utilicen sólo con una subclase. Para ello no tenemos más que crear las funciones que realicen las acciones de los métodos.

9.7.3. Función constructora en subclases.

Volvamos al ejemplo en el cual queremos instanciar los objetos ya con propiedades, porque hemos puesto una función constructora.

Al añadir más métodos en una subclase, la función constructora de la clase general no permite definir las propiedades específicas al instanciar el objeto.

Debemos hacer una función constructora dentro de la subclase que contenga la función constructora de la clase general más las clases específicas de la subclase.

Para ello utilizaremos también la instrucción `parent::funcion()` para incluir la función constructora de la clase general.

En el ejemplo se muestra cómo queda el código de la creación de la subclase tras añadir la función constructora y el método visto anteriormente:

```
class Frutas extends Productos {
function Frutas($n="Productos",$p=0,$d="",$i="producto.jpg",
         $c="no definido",$o="indefinido") {
   parent::Productos($n,$p,$d,$i);
   $this->clase=$c;
   $this->origen=$o;
   }
function mostrar(){
        parent::mostrar();
        echo "Clase : $this->clase.<br/>";
        echo "Origen : $this->origen.<br/>";
        }
}
```

En color rojo oscuro está destacado el código de la función constructora, en ella ponemos como argumentos las propiedades de la clase general más las propiedades que añadimos, con sus valores por defecto.

Incluimos la función general mediante la instrucción `parent::Productos()`. Aquí debemos poner como argumentos las variables que hemos utilizado anteriormente como argumentos para las propiedades generales.

Después incluimos las propiedades específicas de la misma manera que lo hemos hecho con las generales en la clase general.

Tabla de contenidos