Ciclo de Grado Superior: Desarrollo de Aplicaciones WEB

MÓDULO 2ºDAW:
Desarrollo Web en entorno Servidor.

Autor: Jorge López.

Tema 01: Sintaxis básica en PHP - Parte 3ª

10 Funciones

- Funciones de usuario
- Paso de valores por referencia a una función
- Funciones con número de argumentos variables
- Documentación sobre funciones predefinidas:
  - Funciones de conversión de tipo de datos
  - Funciones de cadena
  - Funciones de arrays
  - Funciones para comprobar el estado de una variable
  - Funciones para uso con la Base de Datos MySql
  - Funciones de ficheros

04 Funciones.

Funciones de usuario

De igual forma que ocurre con el navegador en el caso del HTML, PHP lee e interpreta las instrucciones contenidas en los scripts de **forma secuencial**. Es decir, las instrucciones se van ejecutando en el mismo orden en el que aparecen en el documento original, con la excepción de las **funciones**.

En este caso, en una **función**, su bloque de instrucciones son puestos a disposición de PHP, pero no se ejecutarán hasta el momento en que sean requeridas de forma expresa.

¿Dónde deben insertarse?:

Aunque en versiones antiguas de PHP era necesario definir la función antes de invocarla, a partir de la **versión 4 de PHP** no es necesaria esa organización secuencial. La función puede estar escrita dentro de cualquier script y en cualquier parte del documento, sin que tenga importancia alguna el lugar en el que se incluya la llamada a la misma.

También es posible (y bastante habitual) incluir funciones de uso frecuente en documentos externos de modo que pueden ser compartidas por varios ficheros **.php** independientes, en este caso, además de invocarla es necesario indicar a PHP el lugar donde debe buscarlas. Hablaremos de ello cuando estudiemos lo relativo a **include**.

Definición de la función:

Las funciones de usuario requieren la siguiente sintaxis:

```php
<?php
  function nombre()
    {
    Instrucción1;
    instrucción2;
    ....
    // el return es opcional
    instrucciónN;
    return algo;
    }
?>
```

Crea un fichero **prueba.php** y prueba este código:

```php
<?php
    function suma($a,$b)
    {
      return $a+$b;
    }

    $resultado=suma(4,7);
    echo "el resultado es: ".$resultado;
?>
```

Es **imprescindible respetar estrictamente la sintaxis** que requiere de forma obligatoria los siguientes elementos:

- La palabra **function** debe estar escrita en minúsculas.
- El **nombre de la función**, que debe seguir criterios similares a los de los nombres de variables, aunque en este caso no se antepone el símbolo $ ni ningún otro.
- Los **paréntesis** (), incluso cuando no contenga ningún parámetro.
- Las **llaves de apertura y cierre** {}, dentro de las cuales se escribirán las instrucciones correspondientes a la función.

Las funciones pueden ser llamadas con varios parámetros o con ninguno, dependiendo de su definición. Cuando PHP encuentra en el código la llamada a una función, primero, evalúa cada argumento y los utiliza como parámetro de entrada. Después, ejecuta la función y devuelve el valor solicitado o realiza alguna acción sin enviar ningún valor de salida.

El siguiente ejemplo contiene un conjunto de llamadas a funciones con 0,1 ó 2 parámetros de entrada:

```
<?php
    echo sqrt(9);        // Visualiza la raíz cuadrada de 9 que es 3
    echo rand(10,20);  // Visualiza un número aleatorio entre 10 y 20
    echo pi();             // Visualiza el número pi
?>
```

Ejecución de la función:

Las funciones PHP no se ejecutan **en tanto no sean invocadas**. Para invocar una función la sintaxis es la siguiente:

nombre()

Al ser llamada con esta sintaxis (desde cualquier script) se ejecutarán las instrucciones contenidas en dicha función.

Ámbito de las variables:

Resumamos lo ya comentado cuando tratamos el tema de las variables en la **parte 01**:

- Las funciones no leen valores de variables definidas fuera de su ámbito salvo que dentro de la propia función se definan de forma expresa como **globales**.
- Si una función modifica el valor de una variable global, el nuevo valor persiste después de abandonar la función.
- Si dentro de una función se define y se utiliza una variable, la nueva variable se inicia con valor nulo y los eventuales valores que pudiera ir conteniendo se pierden en el momento en que se acaba la ejecución de la función.

Asignación de valores a variables de la función:

A las variables **no globales** se les pueden asignar sus **valores iniciales** de dos formas:

- Incluyéndolas en una línea de instrucciones contenida en la propia función:

```
$a=100; $b=200;
```

- Insertando los valores directamente dentro del paréntesis que (de forma obligatoria) debe seguir al nombre de la función. En este caso la sintaxis sería:

```
// se supone que tenemos esta función
function nombre ($a,$b)
// la llamada a la función sería así
nombre (100,200);
```

Si el número de valores contenidos en la llamada fuera mayor que el número de variables definidas en la función, los excedentes serían ignorados y, si fuera inferior, aparecerá el siguiente error por defecto: "**Warning: Missing argument 1 for…**"

- También es posible incluir en la llamada a la función los nombres de algunas variables definidas en el ámbito externo a la función. Si la función hiciera algún cambio sobre esta variable, externa a la función, no le afectaría.

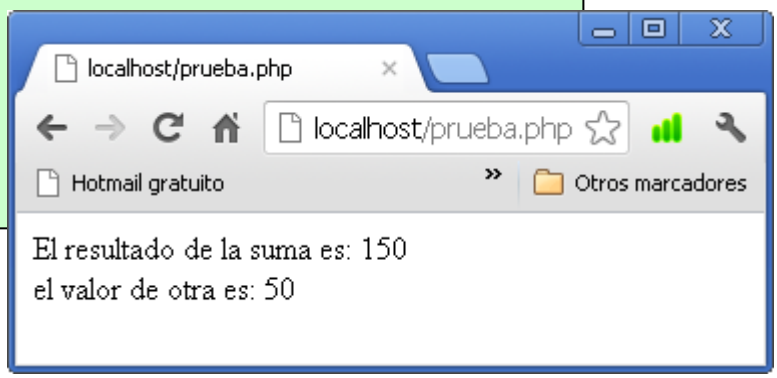
Vamos a ver un **ejemplo** de esto para que quede claro:

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$otra=50;

function ejemplo($a,$b)
{
echo "El resultado de la suma es: ",$a+$b;
$otra=123;
// también podemos poner esto, el resultado es el mismo, no afecta
$b=123;
}

ejemplo(100,$otra);
echo "<br> el valor de otra es: ".$otra;
?>
```



Paso de valores por referencia a una función

Tal como hemos visto, las funciones PHP pueden recibir valores de variables externas y utilizar esos valores sin que el valor original de las mismas (salvo que se les asigne la condición de globales dentro de la función) sufra modificación.

Una manera de lograr que los valores de una variable externa puedan ser modificados por una función, es lo que se llama **pasar variables por referencia**.

Por defecto, **los parámetros de una función se pasan por valor**, de manera que, al cambiar el valor de un parámetro dentro de la función, no se ve modificado fuera de ella. Para permitir que dichos cambios se vean reflejados fuera de la función, hay que pasar los parámetros por referencia.

Para conseguir que un parámetro de una función **siempre se pase por referencia**, hay que anteponer el símbolo **&** al nombre del parámetro en la definición de la función, y **también a la función**, podremos pasar parámetros por **referencia** y también parámetros por **valor**.

Veamos un **ejemplo**:

Crea un fichero llamado **prueba.php** y prueba este código:

```php
<?php
    function concatena($string)
    {
      $string.=' y algo más';
    }
    function concatena2(&$string,$numero)
    {
      $string.=' y algo más';
      $numero=500;
    }

    $cadena1 = 'Esto es una cadena1';
    $cadena2 = 'Esto es una cadena2';
    $numero=123;
    concatena($cadena1);                    // paso por valor (no cambia valor)
    concatena2($cadena2,$numero);     // paso por referencia para $cadena2 (cambia valor)
    echo "el valor de cadena1 es : ".$cadena1."<br>";
    echo "el valor de cadena2 es : ".$cadena2."<br>";
?>
```

Funciones con número de argumentos variables

Es habitual que el número de parámetros que se le pase a una función dependa de la situación en la cual es llamada. Hay tres formas de hacer esto (**llamar a una función con más o menos parámetros según el caso**) en PHP:

- **Definiendo la función con argumentos por defecto**. Este método permite hacer llamadas con menos parámetros sin que aparezca un error.
- **Usando un array** para pasar las variables.
- **Usando las funciones de argumento variable**:
  func_num_args () , func_get_arg() y func_get_args () , ya utilizadas en **PHP 4**.

Argumentos por defecto:

Para definir parámetros de este tipo, se sustituyen las variables por expresiones de asignación. El formato es el siguiente:

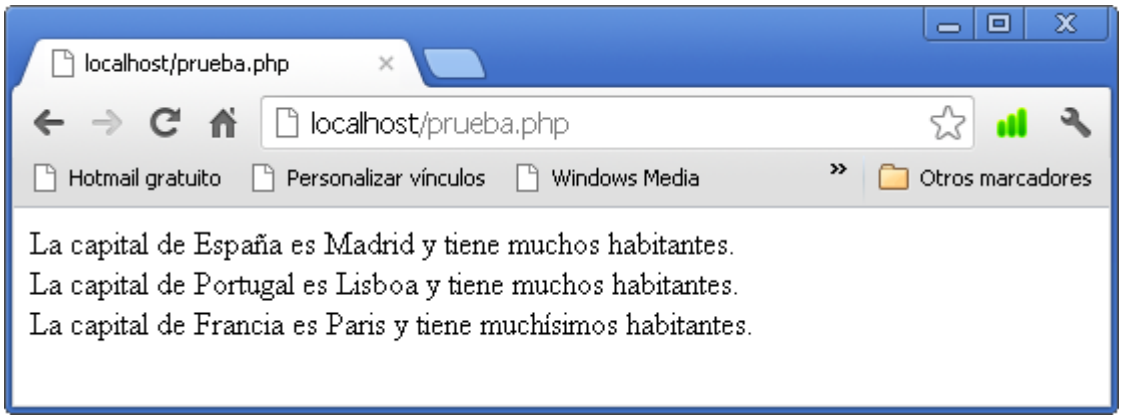
function nombre ($argumento1=valor1, $argumento2=valor2, ...)

De esta forma, al llamar a la función, podemos hacerlo con varios parámetros. Si alguno de los parámetros es obviado, la variable tendrá como valor, el valor por defecto de la definición.

Vamos a ver un **ejemplo**:

```php
<?php
    function capitales($A,$B="Madrid",$C="muchos")
     {
     return ("La capital de $A es $B y tiene $C habitantes.<br>") ;
     }
    echo capitales("España");
    echo capitales("Portugal","Lisboa");
    echo capitales("Francia","Paris","muchísimos");
?>
```

La salida por pantalla es la siguiente:

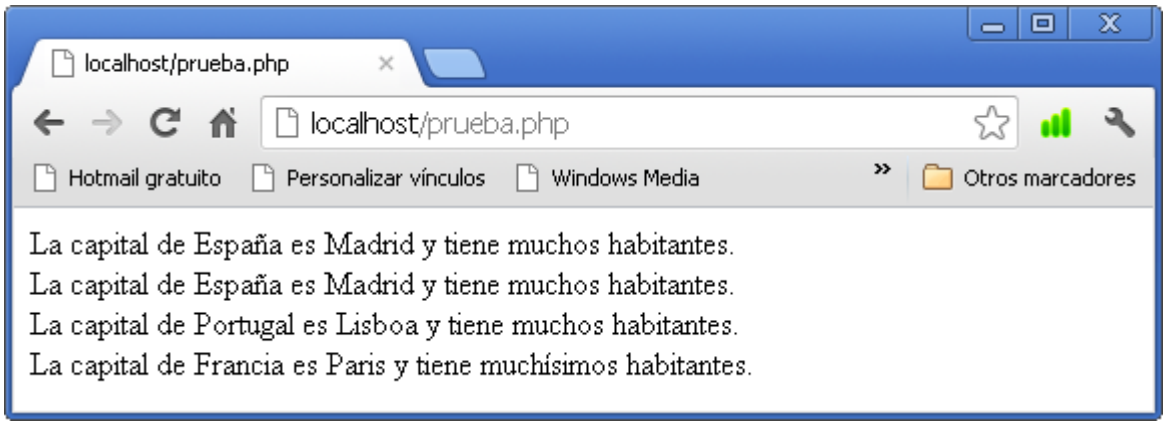


Argumentos mediante un array:

Vamos a ver cual es el procedimiento para pasar valores a una función **utilizando un array**.

El **ejemplo** siguiente usa el operador ternario y los arrays asociativos:
(página siguiente)

```php
<?php
    function capitales($datos)
    {
    // isset determina si una variable está definida y no es NULL.
    $A = isset ($datos['Pais'] ) ? $datos ['Pais'] : "España";
    $B = isset ($datos['Capital'] ) ? $datos ['Capital'] : "Madrid";
    $C = isset ($datos['Habitantes'] ) ? $datos['Habitantes'] : "muchos";
    return ("La capital de $A es $B y tiene $C habitantes.<br>");
    }
    // Introducimos en el array los datos uno por uno para que sea más fácil de entender
    $datos=array();
    echo capitales($datos);
    $datos ['Pais'] = "España2";
    echo capitales($datos);
    $datos ['Pais'] = "Portugal";
    $datos ['Capital'] = "Lisboa";
    echo capitales($datos);
    $datos ['Pais'] = "Francia";
    $datos ['Capital'] = "Paris";
    $datos ['Habitantes'] = "muchísimos";
    echo capitales($datos);
?>
```



Como se puede ver, la función tiene un único argumento, que contiene todos los datos que nuestro programa necesita. Esta vez, los valores por defecto los introducimos mediante el **operador ternario**, que chequea si existe el dato referido al **Pais**, **Capital** o **Habitantes** y, si no existe, se añaden los datos "España", "Madrid" y "muchos".

El ejemplo utiliza la función con 0,1,2 y 3 parámetros. La ventaja de este método es que podrá utilizar los argumentos que quiera y en el orden que necesite.

Múltiples argumentos con func_num_args():

Desde la versión 4, PHP ofrece algunas funciones que pueden ser utilizadas para recuperar los argumentos. Son muy similares al lenguaje C:

func_num_args(): devuelve el número de argumentos que recibe la función desde la que es llamada.

func_get_arg(): devuelve uno a uno los argumentos pasados de la siguiente forma:
func_get_arg(0), func_get_arg(1), func_get_arg(5).

func_get_args(): devuelve un array con todos los argumentos pasados a la función, con los índices del array empezando desde 0.

Cualquiera de estas funciones dará un error si son llamadas fuera del entorno de una función, y **func_get_arg()**, producirá un fallo si es llamada con un número más alto que los argumentos que se reciben.

Las funciones anteriores dan una ventaja a largo plazo, ya que si, durante el período de vida de una función necesita añadir algún argumento más, se puede realizar sin necesidad de cambiar el **código de las llamadas** o **el código de definición de la función**.

En el **ejemplo** siguiente podemos comprobar cómo se utiliza este método:

```php
<?php
    function capitales()
    {
    $na = func_num_args();

    $A = $na > 0 ? func_get_arg(0) : "España";
    $B = $na > 1 ? func_get_arg(1) : "Madrid";
    $C = $na > 2 ? func_get_arg(2) : "muchos";

    return ("Número argumentos es: $na. La capital de $A es $B y tiene $C habitantes.<br>");
    }

    echo capitales();
    echo capitales("Portugal", "Lisboa");
    echo capitales("Francia", "Paris", "muchísimos");
?>
```

Esta forma **de utilizar los argumentos sigue teniendo una limitación**. Los argumentos deben ser pasados en un lugar determinado, sino la función no hará bien su trabajo.

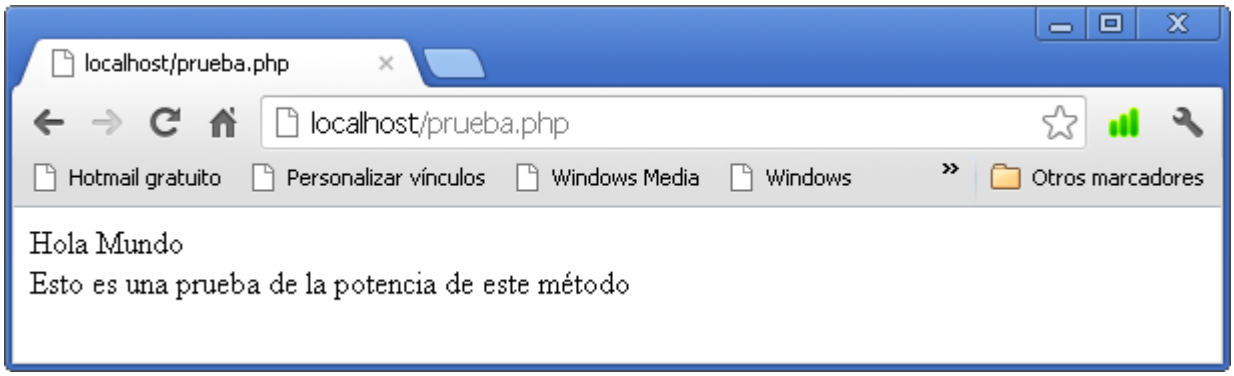
Utilizar los parámetros como array, es el método más flexible y el más utilizado en los programas PHP. Aún así, es muy útil cuando no sepa cuántos datos necesita manejar una función. Podemos utilizarlo para funciones que sumen todos los parámetros que introduzca o concatenen todas las palabras, como el **ejemplo** siguiente:

```php
<?php
    function concatenar()
    {
    $resultado="";
    $na = func_num_args();

    $array_parametros = func_get_args();

    for ($x=0; $x< $na; $x++)
      {
      $resultado = $resultado . $array_parametros[$x] ;
      }
    return $resultado."<br>" ;
    }

    echo concatenar("Hola ","Mundo");
    echo concatenar("Esto ","es ","una ","prueba ","de"," la potencia"," de este"," método");
?>
```



Documentación sobre funciones predefinidas

Desde los inicios, PHP se diseñó para que fuera **sencillo de utilizar** y **de ir haciéndose cada más grande** en lo que se refiere a **funcionalidad**, debido a su naturaleza de código abierto toda la comunidad de desarrolladores puede contribuir aportando instrucciones y funciones nuevas.

La potencia reside en **la gran cantidad de funciones escritas de todo tipo**.

Por muchas funciones que aquí describiéramos sería imposible cubrir todas las funciones existentes con detalle. Por esta razón es imprescindible tener como punto de referencia **la guía on-line** de **PHP** en **www.php.net**, donde podemos encontrar clasificadas por temática todas las funciones existentes hasta ahora. Hay muchos caminos **distintos** para llegar a consultar el funcionamiento y uso de una función, aquí te dejo una dirección donde poder consultar cualquier función:

http://es.php.net/manual/es/index.php

A continuación veremos un **resumen** con las **funciones más importantes**:

Funciones de conversión de tipo de datos:

Sentencia	Resultado
`(int) $var` `(integer) $var`	Conversión a tipo `integer`.
`(bool) $var` `(boolean) $var`	Conversión a tipo `boolean`.

`(float) $var`
`(double) $var`
`(real) $var`
	Conversión a tipo `float`.

`(string) $var`	Conversión a tipo `string`.
`(array) $var`	Conversión a tipo `array`.
`(object) $var`	Conversión a tipo `object`.

Valor de $var	(int) $var	(bool) $var	(string) $var	(float) $var
null	0	false	""	0
true	1	true	"1"	1
false	0	false	""	0
0	0	false	"0"	0
3.8	3	true	"3.8"	3.8
"0"	0	false	"0"	0
"10"	10	true	"10"	10
"6 metros"	6	true	"6 metros"	6
"hola"	0	true	"hola"	0

Funciones de cadena:

Método	Descripción	Sintaxis
echo()	Imprime una cadena.	echo(string) echo "hola"; ➜ "hola"
explode()	Rompe un *string* en trozos, utilizando el delimitador y lo guarda en un *array*.	explode(delimitador,string[,limite]) explode(",","Hola mundo"); ➜Array([0]=>Hola, [1]=> mundo)
implode()	Convierte un *array* en un *string*.	implode(delimitador,nombre_array) $a = array('lunes', 'martes'); $b=implode(" ",$a); ➜ b= " lunes martes"
ltrim()	Elimina los espacios u otros caracteres predefinidos en el lado izquierdo de una cadena.	ltrim(string[,caracteres predefinidos]) ltrim(" hola "); ➜ "hola "
rtrim()	Elimina los espacios u otros caracteres predefinidos en el lado derecho de una cadena.	rtrim(string[,caracteres predefinidos]) rtrim(" hola "); ➜ " hola"
str_repeat()	Repite un *string* un número específico de veces.	str_repeat(string, nºrepeticiones) str_repeat("#",2);➜ "##"
str_replace()	Reemplaza una parte de un *string* por otra cadena.	str_replace(cadena_vieja,cadena_nueva,string,[count]) str_replace("hola","adios", "hola a todos");➜ "adiós a todos"
strcmp()	Compara dos *string*. Devuelve 0 si son iguales y distinto de 0 si son distintos.	strcmp(string1,string2) strcmp("hola", "hola"); ➜0
strlen()	Devuelve la longitud de un *string*.	strlen(string) strlen("hola");➜4
strrev()	Devuelve un *string* invertido.	strrev(string) strrev("hola");➜"aloh"
strstr()	Busca la primera ocurrencia de un *string* en otro.	strstr(string1,string2) strstr("hola a todos", "todos"); ➜ "todos"
strtolower()	Convierte a minúsculas un *string*.	strtolower(string) strtolower("HOLA");➜ "hola"
strtoupper()	Convierte a mayúsculas un *string*.	strtoupper(string) strtoupper("hola");➜ "HOLA"
trim()	Elimina los espacios en blanco y otros caracteres predefinidos a un lado y a otro del *string*.	trim(string[,caracteres predefinidos]) trim(" hola "); ➜"hola"

Funciones de arrays:

Método	Descripción	Sintaxis
array()	Crea un *array*.	array(clave => valor) $a=array("a"=>"lunes","b"=>"martes");
array_merge()	Combina uno o más *arrays* en otro *array*.	array_merge(array1[,array2]…) $a1=array("a"=>"lunes","b"=>"martes"); $a2=array("c"=>"miercoles"); array_merge($a1,$a2);➔array("a"=>"lunes","b"=>"martes","c"=>"miercoles")
array_pop()	Elimina el último elemento del *array*.	array_pop(nombre_array) $a=array("a","b","c"); array_pop($a);➔ array("a","b")
array_push()	Añade uno o más elementos a un *array* por el final.	array_push(nombre_array, valor1, valor2…) $a=array("a","b","c"); array_push($a, "d"); ➔array("a","b","c","d")
array_values()	Devuelve el valor de todos los elementos.	array_values(nombre_array) $a1=array("a"=>"lunes","b"=>"martes"); array_values($a1);➔ array([0]=>"lunes", [1]=>"martes")
count()	Devuelve el nº de elementos que hay en un *array*.	count(nombre_array) $a=array("a","b","c"); count($a);➔3
in_array()	Informa de si un elemento está dentro del *array*. Devuelve un valor booleano	in_array(valor,nombre_array[,tipo_búsqueda]) $a=array("a","b","c"); in_array("a",$a); ➔ true

Funciones para comprobar el estado de una variable:

Contenido de $var	isset($var)	empty($var)	(bool) $var
$var = null;	false	true	false
$var = 0;	true	true	false
$var = true	true	false	true
$var = false	true	true	false
$var = "0";	true	true	false
$var = "";	true	true	false
$var = "foo";	true	false	true
$var = array( );	true	true	false
unset ($var);	false	true	false

Funciones para uso con la Base de Datos MySql:

Método	Descripción
`mysql_num_rows($result)/ mysqli_num_rows($result)`	Devuelve el número de filas que tiene un resultado.
`mysql_num_fields($result)/ mysqli_num_fields($result)`	Devuelve el número de campos que tiene un resultado.
`mysql_affected_rows([$link]) /mysqli_affected_rows($link)`	Devuelve el número de filas afectadas por una operación SQL.
`mysql_free_result($result)/ mysqli_free_result()`	Libera la memoria reservada para almacenar un resultado.
`mysql_fetch_array($result)/ mysqli_fetch_array($result)`	Devuelve un *array* con las filas obtenidas o un valor booleano que índica que no tiene más filas.
`mysql_fetch_row($result)/ mysqli_fetch_row($result)`	Devuelve una fila de resultados como un *array* enumerado.
`mysql_fetch_field($result, [$pos_campo])/ mysqli_fetch_field($result)`	Devuelve el siguiente campo del conjunto de resultados.
`mysql_fetch_assoc($result)/ mysqli_fetch_assoc($result)`	Devuelve una fila de resultados como un *array* asociativo.
`mysql_fetch_object($result)/ mysqli_fetch_object($result)`	Devuelve la fila actual del resultado en forma de un objeto.
`mysql_result($result)`	Devuelve el resultado.
`mysql_field_len($result, $pos_campo)`	Devuelve el tamaño de un campo específico.
`mysql_fetch_lengths($result, $num)/ mysqli_fetch_lengths($result)`	Devuelve la longitud de las columnas de la fila actual en el conjunto de resultados.

Funciones de ficheros:

Funciones PHP	Descripción
`$descriptor=fopen($nombre, $tipo_acceso)`	Crea y abre un fichero (válido para todo tipo de ficheros). Los tipos de acceso más utilizado son: r (lectura), w (escritura), a (escritura añadiendo al final del fichero).
`fclose($descriptor)`	Cierra un fichero (válido para todo tipo de ficheros).
`fread($descriptor, $tamaño)`	Lee de un fichero de texto un tamaño determinado.
`fwrite($descriptor, $texto, [$tamaño])`	Escribe en un fichero de texto la cadena que le pasamos como parámetro.
`fgetcsv($descriptor,[$tamaño], [$delimitador],[$cierre_campo], [$carácter_escape])`	Lee un fichero CSV. Devuelve en una matriz los valores de un campo.
`fputcsv($descriptor, $matriz_valores, [$delimitador],[$cierre_campo])`	Escribe en un fichero CSV. Devuelve la longitud de la cadena escrita.
`feof($descriptor)`	Devuelve un booleano indicando si se ha llegado al final del fichero (válido para todo tipo de ficheros).