



# PHP & MySQL

*Anexo 2.- Sintaxis del lenguaje*



**DISTRIBUIDO POR:**

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

# El lenguaje PHP

## Sintaxis básica

### Etiquetas y sentencias.

Las etiquetas `<?php` y `?>` permite inscribir código PHP.

```
<?php echo 'hola'; ?>
```

Cada sentencia de código PHP debe terminar en punto y coma:

```
<?php
    for( $i = 0; $i < 10; $i++ ) {
        echo "$i<br />";
    }
?>
```

Pueden declararse varias sentencias en una misma línea, pero se recomienda indicar una sentencia por línea para favorecer la claridad del código.

### Comentarios

Los comentarios indican regiones de código que no son procesadas por el intérprete de PHP y que únicamente contienen indicaciones del programador.

Los comentarios pueden ser de una línea:

```
// Comentario de una línea
```

También puede ser multilínea.

```
/*
    Comentario de múltiples líneas
    línea 0
    línea 1
    línea 2
*/
```

### Llamadas a funciones

PHP define multitud de funciones que pueden emplearse para diferentes tareas. Estas funciones conforman lo que se denomina la API de PHP, y pueden consultarse en la página oficial de PHP:

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

Estas funciones están caracterizadas por tres elementos:

- **Identificador** → Es el nombre mismo de la función por la que la llama.
- **argumento** → Es el valor o valores que requiere la función para hacer su tarea.
- **Retorno** → Es el valor retornado por la función como resultado

A modo de ejemplo veamos la función **date()**.

```
string date ( string $format )
```

La función **Date()** devuelve una cadena (*string*) con la fecha actual del servidor expresada en el formato indicado por el argumento ( *\$format* ).

El siguiente código:

```
date('H:i, jS F Y')
```

Devolverá una cadena de caracteres con el siguiente contenido:

```
15:08, 8th November 2016
```

El valor devuelto puede asignarse a una variable o emplearlo como argumento para otra función:

```
<?php
// Asigna en una variable el valor devuelto por la función date()
$fecha = date('H:i, jS F Y');
// Muestra el valor de $fecha
var_dump($fecha);
// Muestra el valor devuelto por la función date()
var_dump(date('H:i, jS F Y'));
?>
```

La función **var\_dump()** es una función que muestra por pantalla el valor y tipo de la variable indicada como argumento. Esta función es similar al comando **echo** pero se emplea únicamente para labores de comprobación y desarrollo.

Referencia de la función **date()**:

<http://php.net/manual/es/function.date.php>

Referencia de la función **var\_dump()**:

<http://php.net/manual/es/function.var-dump.php>

## Generando contenido dinámico

La principal funcionalidad de PHP es la generación de código como parte del código HTML de una página web cuando ésta es solicitada al servidor.

El comando más básico para generar contenido en PHP es ***echo***. Este permite incrustar en el lugar que ocupa el código PHP la cadena indicada.

Sea la siguiente página:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
  <body>
    <div>
      <?php
        echo "<h1>HOY ES ";
        echo date('H:i, jS F Y');
        echo "</h1>";
      ?>
    </div>
  </body>
</html>
```

La página devuelta al usuario contendrá el siguiente código HTML:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title></title>
  <meta charset="utf-8" />
</head>
<body>
  <div>
    <h1>HOY ES 15:08, 8th November 2016</h1>
  </div>
</body>
</html>
```

## Operador de concatenación (.)

El mismo resultado que el del código PHP anterior podría haberse conseguido con una sola línea:

```
<?php
  echo "<h1>HOY ES echo ".date('H:i, jS F Y')."</h1>";
?>
```

En este caso se ha empleado el operador de concatenación “.” para añadir diferentes valores a la función *echo* en la misma línea.

## Variables

Una variable se define como un área de memoria a la que se asigna un identificador que la representa y en la que se almacena un valor de un determinado tipo.

Los identificadores de variables en PHP cumplen las siguientes normas:

- Comenzar por el signo \$
- El segundo carácter sólo puede ser un letra o un guión bajo.
- A partir del tercer carácter pueden emplearse números.

Los identificadores pueden tener cualquier longitud pero se recomienda que no sean excesivamente largos y que identifiquen el valor que almacenan. Debe tenerse en cuenta también que PHP diferencia entre mayúsculas y minúsculas en por lo que las variables \$a y \$A no son consideradas la misma variable.

## Tipos de variables

El tipo de una variable indica el tipo de valor que contiene, es decir; si almacena un número entero, decimal, un valor lógico ( TRUE / FALSE ), una cadena de texto... etc.

Los principales tipos de datos son los siguientes:

- **Enteros:**

```
$a = 20;
```

- **Decimales:** Cualquier valor fraccionario. El separador decimal es el punto.

```
$d = 245.94; // Notación convencional.  
$d = 2395e-2; // Notación científica -> equivale a 23,95
```

- **Cadenas de caracteres:** Almacenan palabras y caracteres. Los valores deben expresarse entre comillas simples o dobles.

```
$nombre = "angel"; // Cadena  
// valor cadena multilinea.  
$mensaje = <<<HERE  
Esto es una mensaje multilinea<br>  
para cadenas de texto largas.<br>  
HERE;
```

## Programación Web en PHP

Si se emplean comillas dobles para definir una cadena de texto, se resuelve y concatena el valor de cualquier variable que aparezca. Si se emplean comillas simples se incluye el nombre de la variable sin más.

```
$nombre = "Roger Petrov";  
$saludo1 = "Saludos $nombre";  
$saludo2 = 'Saludos $nombre';  
  
echo $saludo1;           // Muestra 'Saludos Roger Petrov'  
echo $saludo2;           // Muestra 'Saludos $nombre'
```

El código mostrará:

```
string 'Saludos Roger Petrov' (length=20)  
  
string 'Saludos $nombre' (length=15)
```

- **Lógico:** Admiten únicamente valores cierto y falso. Para asignarles valor pueden expresarse las constantes predefinidas **true** y **false**.

```
$validado = true;
```

- **Matrices:** Almacenan múltiples valores con el mismo identificador y acceder a ellos empleando un índice. ( *se ve más adelante* ).
- **Objetos.** Almacenan datos complejos junto con métodos para su manipulación definidos previamente mediante clases. ( *se ve más adelante* ).

El tipo de una variable en PHP cambia según el valor que se le asigna. Esto es importante tenerlo presente a la hora de hacer operaciones con las variables. Si es necesario conocer el tipo de una variable puede emplearse la función `gettype()` que devuelve una cadena con el tipo de la variable:

```
// $id es de tipo entero  
$id = 0;  
echo gettype($id). "<br/>";  
// $id es de tipo decimal  
$id = 0.0;  
echo gettype($id). "<br/>";  
// $id es de tipo cadena  
$id = "petrov";  
echo gettype($id). "<br/>";  
// $id es de tipo logico  
$id = true;  
echo gettype($id). "<br/>";
```

El resultado sería:

```
integer  
double  
string  
boolean
```

PHP también dispone de funciones específicas para comprobar el tipo de una variable:

- `is_array()` → Indica si la variable es una matriz
- `is_bool()` → Indica si la variable contiene un valor lógico.
- `is_float()` → Indica si la variable contiene un valor decimal.
- `is_integer()` → Indica si la variable contiene un valor entero.
- `is_double()` → indica si la variable contiene un valor decimal ( = `is_float` )
- `is_long()` → indica si la variable contiene un valor entero de 64 bits.
- `is_numeric()` → indica si la variable contiene un valor numérico.
- `is_object()` → indica si la variable contiene un objeto
- `is_real()` → indica si la variable contiene un valor decimal. ( = `is_float` )
- `is_scalar()` → indica si la variable contiene un valor entero.
- `is_string()` → indica si la variable contiene una cadena de texto

Todas estas funciones reciben como parámetro la variable a evaluar y retornan un valor lógico *True* si la variable es del tipo indicado, o *False* en caso contrario:

```
<?php
    $dato1 = "cada";
    $dato2 = "2304";
    $esnumero1 = is_numeric($dato1);
    $esnumero2 = is_numeric($dato2);
    var_dump($esnumero1);           // Muestra FALSE
    var_dump($esnumero2);           // Muestra TRUE
?>
```

*(\*) La función **var\_dump** muestra información del tipo y valor de la variable indicada como parámetro. Esta función es especialmente útil durante el desarrollo de una aplicación para mostrar el tipo y valor de una variable.*

### Conversiones de tipos

El tipo de las variables resulta importante para realizar ciertas operaciones. Por ejemplo; tiene sentido multiplicar dos variables de tipo entero y/o decimal, pero no dos variables de tipo cadena.

Para convertir el tipo de una variable pueden emplearse las conversiones. Para ello puede emplearse la función **settype()**. La función recibe como primer argumento la variable a convertir y el segundo parámetro es el tipo al que se quiere convertir:

- *boolean* → Convertir a valor lógico 'cierto' / 'falso'
- *integer* → Convertir a valor entero.
- *String* → Convertir a cadena de caracteres.
- *Float* → Convertir a valor decimal.

```
<?php
$a = "20";
$b = "40";
echo gettype($a). "<br />"; // Muestra "string"
echo gettype($b). "<br />"; // Muestra "string"
settype($a, "integer");
settype($b, "integer");
echo gettype($a). "<br />"; // Muestra "integer"
echo gettype($b). "<br />"; // Muestra "integer"
?>
```



## Programación Web en PHP

PHP dispone también de funciones que permiten transformar el tipo del valor de una variable a otro tipo concreto.

- **intval()** → Devuelve el valor entero equivalente al valor de la variable indicada como parámetro.
- **floatval()** → Devuelve el valor decimal equivalente al valor de la variable indicada como parámetro.
- **strval()** → Devuelve el valor de tipo cadena de caracteres equivalente al valor de la variable indicada como parámetro.

```
<?php
$cadena = "12345";
$numero = intval( $cadena );
var_dump( $numero );           // numero es entero y contiene 12345
?>
```

### Comprobación y liberación de variables.

Para crear una variable en PHP basta con declararla asignándole un valor:

**`$variable = <valor>;`**

PHP dispone de funciones para comprobar si una variable ya ha sido declarado, y destruirla liberando la memoria que ocupa una vez que ya no es necesaria:

- **isset()** → Esta función devuelve un valor lógico 'cierto' si existe una variable con el identificador indicado como argumento.
- **unset()** → Esta función elimina de la memoria la variable con el identificador dado

Si asignamos un entero a una variable, ésta pasa a ser de tipo entero. Si le asignamos una cadena de texto, pasa a ser de tipo cadena de texto. El tipo de la variable cambia en función de los valores que se le van asignando:

```
<?php
var_dump(isset($a));           // Muestra FALSE
$a=10;                         // Declaracion y asignación de valor.
var_dump(isset($a));           // Muestra TRUE
unset($a);                     // Elimino la variable.
var_dump(isset($a));           // Muestra FALSE
?>
```

## Constantes

Una constante es un identificador al que se le asigna un valor que no puede cambiar.

El identificador de una constante no va precedido del carácter '\$', y puede empezar con una letra o un guión bajo. Las constantes pueden almacenar datos de tipo entero, decimal, cadena o lógico.

Para definir una constante se emplea la función **define()**. Esta función requiere como argumentos el identificador de la constante y su valor.

**define("identificador", valor);**

Para saber si una constante está ya declarada puede emplearse la función **defined()** que recibe como argumento el identificador de la constante y devuelve un valor lógico 'cierto' si existe.

```
<?php
// Definición de una constante con el identificador CTE
define("CTE", 1);
// Indica si la constante CTE existe.
var_dump(defined("CTE"));
// Visualizar el valor de la constante CTE
echo "El valor de la constante es". CTE. "<br />";
?>
```

El resultado mostrado es:

```
boolean true

El valor de la constante es1
```

## Constantes mágicas

Las constantes mágicas son un tipo de constantes definidas por PHP cuyo valor depende del momento en el que se invocan. Estas se caracterizan por tener dos guiones bajos como prefijo y sufijo.

__LINE__	→ Indica el nº de línea del archivo/página PHP en ejecución en el que se invoca.
__FILE__	→ indica el nombre del archivo/página de PHP en el que es invocada.
__FUNCTION__	→ Indica el nombre de la función en la que es invocada.
__CLASS__	→ Indica el nombre de la clase a la que pertenece el método desde donde es invocada.
__METHOD__	→ Indica el nombre del método desde el que es invocada.

**(\*) Más adelante se verán que son las funciones, las clases y los métodos.**

Estas constantes pueden emplearse para mostrar durante el desarrollo de la posición en el código de un determinado error.

## Operadores

Una expresión es una sentencia de código que genera un valor a partir de una operación entre dos valores y un operador. Los valores pueden ser literales, constantes o variables y el operador determinan la operación a realizar.

$\$a + 2;$  → Devuelve la suma del valor de  $\$a$  y el valor 2.  
 $"DNI" . \$a$  → Devuelve la concatenación de la cadena "DNI" y el valor de  $\$a$ .

Igualmente una expresión puede estar compuesta por otras expresiones cuyos resultados se combinan empleando operadores.

$(10 * 5) + (\$a * 2);$

### Operadores aritméticos

Un operador combina dos valores y genera un resultado. Los valores pueden ser literales, variables, o el resultado de otras expresiones.

Los operadores pueden catalogarse según su funcionalidad en los siguientes tipos:

- **Aritméticos:** Realizan operaciones entre valores numéricos.

Ejemplo	Resultado
$\$a + \$b$	Adición Suma de $\$a$ y $\$b$ .
$\$a - \$b$	Sustracción Diferencia de $\$a$ y $\$b$ .
$\$a * \$b$	Multipliación Producto de $\$a$ y $\$b$ .
$\$a / \$b$	División Cociente de $\$a$ y $\$b$ .
$\$a \% \$b$	Módulo Resto de $\$a$ dividido por $\$b$ .

- **Concatenación:** Permite concatenar dos valores de tipo cadena en una única cadena resultante:

```
$a1 = "HOLA ";
$a2 = "MUNDO";
$r = $a1 . $a2;
var_dump( $r ); // Muestra "HOLA MUNDO"
```

- **Pre/PostIncremento / Pre/PostDecremento:** Permite el incremento o decremento del valor de una variable.

Ejemplo	Nombre	Efecto
$++\$a$	Pre-incremento	Incrementa $\$a$ en 1, y retorna valor.
$\$a++$	Post-incremento	Retorna valor, y e incrementa $\$a$ en 1.
$--\$a$	Pre-decremento	Decrementa $\$a$ en 1 y retorna $\$a$ .
$\$a--$	Post-decremento	Retorna valor, y decrementa $\$a$ en 1

**Asignación:** Asigna a una variable un literal, una variable, el resultado de una función o de otra expresión.

```
$a = 10;           //Asignación de valor
$a = $b;           // Asignación de otra variable
$a = sumar( 1, 1 ); // Asignación del retorno de una función
$a = $a + 10;      // Asignación del resultado de otra expresión
```

El operador de asignación puede combinarse con otros operadores cuando el resultado se aplica sobre la misma variable que se opera:

<code>\$a = \$a + \$b</code>	$\rightarrow$ <code>\$a += \$b;</code>
<code>\$a = \$a - \$b</code>	$\rightarrow$ <code>\$a -= \$b;</code>
<code>\$a = \$a * \$b</code>	$\rightarrow$ <code>\$a *= \$b</code>
<code>\$a = \$a / \$b</code>	$\rightarrow$ <code>\$a /= \$b</code>
<code>\$a = \$a &amp; \$b</code>	$\rightarrow$ <code>\$a &amp;= \$b</code>
<code>\$a = \$a   \$b</code>	$\rightarrow$ <code>\$a  = \$b</code>
<code>\$a = \$a ^ \$b</code>	$\rightarrow$ <code>\$a ^= \$b</code>
<code>\$a = \$a . \$b</code>	$\rightarrow$ <code>\$a .= \$b</code>

## Operadores de comparación

Permiten crear expresiones condicionales que comparan dos valores retornando como resultado lógico *cierto* o *falso*:

Ejemplo	Nombre	Resultado
<code>\$a == \$b</code>	<i>Igual</i>	TRUE si \$a es igual a \$b.
<code>\$a === \$b</code>	<i>Idéntico</i>	TRUE si \$a es igual a \$b, y son del mismo tipo
<code>\$a != \$b</code>	<i>Diferente</i>	TRUE si \$a no es igual a \$b.
<code>\$a &lt;&gt; \$b</code>	<i>Diferente</i>	TRUE si \$a no es igual a \$b.
<code>\$a !== \$b</code>	<i>No idéntico</i>	TRUE si \$a no es igual a \$b, o son diferente tipo.
<code>\$a &lt; \$b</code>	<i>Menor que</i>	TRUE si \$a es estrictamente menor que \$b.
<code>\$a &gt; \$b</code>	<i>Mayor que</i>	TRUE si \$a es estrictamente mayor que \$b.
<code>\$a &lt;= \$b</code>	<i>Menor o igual</i>	TRUE si \$a es menor o igual que \$b.
<code>\$a &gt;= \$b</code>	<i>Mayor o igual</i>	TRUE si \$a es mayor o igual que \$b.

En PHP hay que distinguir entre los operadores de comparación (==) y (===). El operador (==) compara el valor únicamente al margen del tipo. El operador (===) compara tanto el valor como el tipo.

```
$a = "1";
$b = 1;
var_dump( $a == $b );    // Muestra "cierto"
var_dump( $a === $b );  // Muestra "falso"

$r = 1;
var_dump($r == true);    // Muestra "cierto". ( 1 equivale 'true' )
var_dump($r === true);  // Muestra "falso"
```

## Operadores de lógico

Permiten combinar expresiones condicionales para crear expresiones condicionales más complejas:

Ejemplo	Nombre	Resultado
<code>!\$a</code>	<i>Negación (no)</i>	TRUE si \$a no es TRUE.
<code>\$a &amp;&amp; \$b</code>	<i>Conjunción (y)</i>	TRUE si tanto \$a como \$b son TRUE.
<code>\$a    \$b</code>	<i>Disyunción (o inclusivo)</i>	TRUE si cualquiera de \$a o \$b es TRUE.

## Programación Web en PHP

- Una **conjunción** ( && ) sólo es cierta cuando todas las condiciones son ciertas. Sea:

`($a >= 10) && ($a <= 20)`

La expresión sólo es cierta cuando ambas condiciones son ciertas, es decir; \$a tiene un valor comprendido entre 10 y 20 ambos incluidos.

- Una **disyunción** ( || ) sólo es falsa cuando todas las condiciones son falsas. Sea:

`($a <= 0) || ($a >= 10)`

La expresión sólo es falsa cuando todas las condiciones son falsas. Si \$a es inferior o igual a 0, o es superior o igual a 10; el resultado es cierto.

- La **negación** ( ! ) invierte el resultado lógico de una condición. Si la condición es cierta la vuelve falsa, si es falsa la vuelve cierta.

`!($a < 0);`

La expresión sólo es cierta cuando ( $a < 0$ ) es cierta. Por tanto, la expresión sólo es cierta cuando \$a es igual o superior a 0.

### Operador condicional ternario.

Este operador da lugar a una expresión que devuelve uno de dos valores en función de la evaluación de una expresión condicional. Este operador sigue la siguiente estructura:

`<variable> = <condicion> ? <asignación si cierto> : <asignación si falso>`


`$mensaje = ( $check == true ) ? "TODO BIEN" : "FALLO";`

Sea la variable \$check de tipo booleano, si su valor es cierto; se ejecuta la asignación \$mensaje="TODO BIEN". Si el valor de \$check es falso, se ejecuta la asignación \$mensaje="FALLO".

### Operador de supresión de errores ( @ )

Este operador evita la visualización de mensajes de error. Se emplea anteponiéndolo a la sentencia que podría generar el error. Sea:

```
<?php
    $a = 7 / 0;
?>
```

 Warning: Division by zero in C:\xampp\htdocs\prueba\inicio.php on line 9				
Call Stack				
#	Time	Memory	Function	Location
1	0.0040	130200	{main}()	..\inicio.php:0

## Programación Web en PHP

Para evitar que el intérprete de PHP muestre un mensaje propio si se produce un error se antepone el '@'.

```
<?php
    $a = @(7 / 0) or die("ERROR DIVISION POR CERO");
?>
```

Si se desea mostrar un mensaje informativo propio al usuario puede emplearse la construcción **or die()**. Esta muestra el mensaje indicado como argumento evitando que se ejecuten las siguientes líneas de código.

## Estructuras de Control.

En los capítulos anteriores se ha visto el modo de definir variables, constantes y expresiones para realizar operaciones de diferentes tipos entre ellas. Estas expresiones permiten transformar y manipular datos para obtener unos resultados.

Por defecto, el código PHP se ejecuta secuencialmente línea a línea. Las **estructuras de control** modifican el orden en que se ejecuta el código de una página o archivo PHP en función de expresiones condicionales.

Se diferencian dos tipos de estructuras de control:

- Estructuras condicionales
- Estructuras repetitivas. (*bucles*)

### Secciones de código

Una sección de código ( o *bloque* ) es un conjunto de líneas de código que aparecen delimitadas entre llaves. { } y cuya ejecución depende de una estructura de control:

```
$a = 0;
if ( $a > 0 )
{
    echo "Por debajo o igual a 0";
    echo "<br />";
    echo "HELADO";
} else {
    echo "Por encima de 0";
    echo "<br />";
    echo "LIQUIDO";
}
```

Una sección de código puede contener, a su vez, otras sentencias de control, lo que permite establecer estructuras de control anidadas unas dentro de otras.

```
$a = 0;
if ( $a == 0 )
{
    if ( $a > 0 ) {
        echo "Mas de 0 grados";
    } else {
        echo "Menos de 0 grados.";
    }
} else {
    echo "0 grados.";
}
```

## Estructuras Condicionales

Las estructuras condicionales permiten controlar la ejecución de una sección de código si se evalúa como cierta una determinada expresión condicional, o una variable posee un determinado valor. Pueden diferenciarse tres tipos:

### Estructuras condicional de una ramas

La estructura condicional de una rama permite establecer una sección de código que se ejecuta si y solo si su expresión condicional se evalúa como cierta.

```
if ( <expresión_condicional> ) {  
    // sección a ejecutar si condición es cierta  
}
```

### Estructura condicional de dos ramas

La estructura condicional de dos ramas establece la ejecución de dos secciones de código en función del resultado de una expresión condicional, tal que una se ejecuta si la condición es cierta, y la otra si es falsa.

```
if ( <expresión_condicional> ) {  
    // sección a ejecutar si condición es cierta ( RAMA IF )  
} else {  
    // sección a ejecutar si condición es falsa ( RAMA ELSE )  
}
```

### Estructura condicional de múltiples ramas

Esta estructura gestiona la ejecución de múltiples secciones de código en función del valor de una variable o una expresión de modo que ejecuta una determinada sección de código para cada valor indicado en la estructura:

```
switch( <expresión> ) {  
    case <valor_1>:  
    {  
        // sección de código para valor1  
    }; break;  
    case <valor_2>:  
    {  
        // sección de código para valor1  
    }; break;  
    default:  
    {  
        // sección de código por defecto  
    }; break;  
}
```

Cada sentencia **case** lleva asociado un valor que debe tener <expresión> para que se ejecute su código. La sección de código de cada rama debe terminar con la sentencia **break**. Una vez que se ejecuta una sección de código de una rama la ejecución salta al final de la estructura y continúa. Puede incluirse una rama opcional default cuyo código se ejecuta si ninguna otra rama se ejecuta.



**Ejemplo:** El siguiente código muestra el día de la semana correspondiente al valor de la variable `$dia`. Si el valor no está contenido entre 1 y 7 ambos incluidos se muestra el mensaje “Día desconocido.”

```
<?php
    $dia = 3;
    switch( $dia ) {
        case 1: {
            echo "Lunes";
        }; break;
        case 2: {
            echo "Martes";
        }; break;
        case 3: {
            echo "Miercoles";
        }; break;
        case 4: {
            echo "Jueves";
        }; break;
        case 5: {
            echo "Viernes";
        }; break;
        case 6: {
            echo "Sabado";
        }; break;
        case 7: {
            echo "Domingo";
        }; break;
        default: {
            echo "Dia desconocido";
        }
    }
}
```

### Anidamiento de estructuras

Una estructura de control *anidada* es aquella que reside en una sección de código comprendida dentro de otra estructura de control que la contiene. Es importante evitar que el anidamiento de estructuras se extienda muchos niveles puesto que genera códigos más difíciles de mantener y corregir en caso de fallos.

**Ejemplo:** La sección de código que aparece se ejecuta únicamente si las dos estructuras condicionales se evalúan como ciertas.

```
if ( $dato >= 25 )
{
    if ( $dato <= 50 )
    {
        // Sección de código
    }
}
```

Puede simplificarse de la siguiente manera con el mismo resultado:

```
if ( $dato >= 25 && $edad <= 50 )
{
    // Sección de código
}
```

**Ejemplo:** El código siguiente gestiona la ejecución de las diferentes secciones de código en función del valor de una determinada variable:

```
$edad = 25;
if ( $edad > 0 && $edad < 17 ) // Primera Condicional
{
    echo "Es menor de edad";
}
else
{
    if ( $edad > 18 && $edad < 65 ) // Condicional embebida nivel 1.
    {
        echo "Es adulto";
    }
    else
    {
        if ( $edad >= 65 ) // Condicional embebida nivel 2.
        {
            echo "Es mayor de edad";
        }
    }
}
```

El código puede simplificarse empleando la partícula **elseif**. Esta partícula permite incluir una expresión condicional anidada dentro de la rama **else** de otra expresión condicional. Esto permite escribir un código más sencillo que el anterior manteniendo la misma funcionalidad.

```
if ( $edad > 0 && $edad < 17 ) // Primera condicional
{
    echo "Es menor de edad";
}
else if ( $edad > 18 && $edad < 65 ) // Condicional embebida nivel 1.
{
    echo "Es adulto";
}
else if ( $edad >= 65 ) // Condicional embebida nivel 2.
{
    echo "Es mayor de edad";
}
```

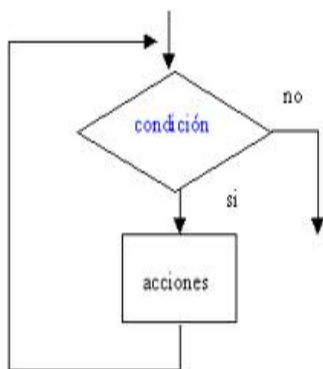
## Estructuras Repetitivas

Las estructuras repetitivas permiten establecer la ejecución reiterada de una sección del código para conseguir un determinado resultado. Pueden diferenciarse dos tipos de bucles: determinados e indeterminados.

Los bucles indeterminados son aquellos que no se conoce cuantas veces tendrán que ejecutarse y se ejecutan mientras se cumple una determinada condición.

### Estructura repetitiva indeterminada 0-n

```
while ( <expresion_condicional> ) {  
    // sección de código a repetirse.  
}
```



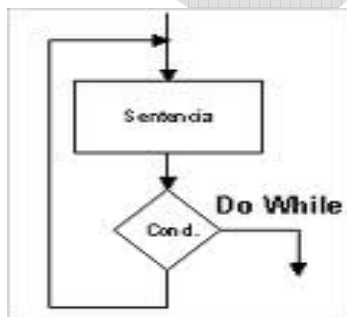
La estructura repetitiva **while** permite que una sección de código se ejecute mientras la expresión condicional indicada se evalúa como cierta. En el momento en que la condición se evalúa como falsa la ejecución salta fuera de la estructura y continúa con el resto del código.

Esta estructura repetitiva evalúa la condición antes de ejecutar la sección de código, por lo que si se evalúa como falsa la primera vez la ejecución salta la sección de código y no llega a ejecutarse ni una sola vez.

Esta estructura repetitiva se utiliza para programar secciones de código que pueden ejecutarse de 0 a n veces.

### Estructura repetitiva indeterminada 1-n

```
do {  
    // sección de código a repetirse.  
} while ( <expresion_condicional> );
```



Esta estructura repetitiva es equivalente a la anterior pero con la diferencia de que la expresión condicional se evalúa tras ejecutar la sección de código. Si la condición se evalúa como cierta se repite la sección de código, en caso contrario la ejecución continúa más allá.

En cualquier caso; la sección de código se ejecuta siempre como mínimo una vez.

Esta estructura repetitiva se utilizar para programar secciones de código que pueden ejecutarse al menos una vez.

### Estructura repetitiva determinada

Los *bucles determinados* son los se ejecutan un número de veces fijo y conocido desde el principio.

```
for( <expresión_asignación >; <expresión_condición>; <expresión_aritmética> )
{
    // sección de código a repetirse.
}
```

Esta estructura repetitiva emplea una *variable de control* que contabiliza el número de veces que se va ejecutando el bucle hasta llegar al valor deseado. La estructura consta de tres expresiones:

- *<expresión\_asignación>* → Expresión que asigna valor inicial a la variable de control.
- *<expresión\_condición>* → Expresión condicional que indica la condición para que siga ejecutándose el bucle.
- *<expresión\_aritmética>* → Expresión aritmética que modifica el valor de la variable de control por cada ejecución del bucle.

**Ejemplo:** En este bucle la variable de control es *\$i* que empieza valiendo 0 y va incrementándose de 1 en 1 mientras su valor sea inferior a 10. El bucle se ejecuta un total de 10 veces en las cuales la variable *\$i* recorre los valores entre 0 y 9 inclusive.

```
<?php
echo "<table border='1'>";
echo "<tr>";
for( $i = 0; $i < 10; $i++) {
    echo "<td>".$i."</td>";
}
echo "</tr>";
echo "</table>";
?>
```

El código generado resultante es:

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

## Sentencias de salto en bucles

Las sentencias de salto permiten “romper” la ejecución normal de un bucle provocando que termine antes de tiempo (*break*) o salte una iteración (*continue*)

### Sentencia *break*

Esta sentencia fuerza el abandono de la ejecución de una estructura de control antes de lo previsto. El valor *n* indica los niveles a saltar en el caso de que se estén ejecutando estructuras de control anidadas. El valor por defecto es 1.

**Ejemplo:** El siguiente código muestra un bucle que se debería ejecutar 10 veces. Sin embargo, cuando el valor de *\$i* vale 5 se ejecuta una sentencia *break*. Esto provoca el salto de la ejecución fuera de la estructura repetitiva sin haber completado el bucle:

```
for ( $i = 0; $i < 10; $i++)  
{  
    echo $i;  
    if ( $i == 5 ) break;  
}  
echo "fin";
```

El resultado mostrado por pantalla será:

012345Fin

### Sentencia *continue*

Esta sentencia *continue* se estructura en estructuras de control repetitivas para forzar el salto a la siguiente iteración sin haberse completado la ejecución de la sección de código correspondiente.

**Ejemplo:** El siguiente código muestra el mismo un bucle que debe ejecutarse un total de 10 veces mostrando en cada iteración el valor de la variable *\$i*. Sin embargo; cuando la variable alcanza el valor 5, se ejecuta una sentencia *continue*. Esto provoca el salto de la ejecución a la siguiente iteración sin completarse la ejecución de toda la sección de código; por lo que el valor queda sin mostrar al no ejecutarse la sentencia *echo* correspondiente.

```
for ( $i = 0; $i < 10; $i++)  
{  
    if ( $i == 5 ) continue;  
    echo $i."<br>";  
}  
echo "fin";
```

El resultado mostrado por pantalla será:

012346789fin

## Ejercicios

1.- Crear una página que muestre el mensaje “HOLA MUNDO” repetidos diez veces. Hazlo empleando código en línea.

2.- Crea una página que muestre la suma de los 100 primeros números naturales y muestre el mensaje: “La suma de los primeros 100 números naturales es: XXX”. Implementar el cálculo empleando las tres estructuras repetitivas conocidas.

3.- Crea una página que muestre una tabla bidimensional con la tabla de multiplicar del 1 al 10.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

4.- Crea una página que muestre mediante código PHP una tabla con un rango de temperaturas de 0°C a 100°C ordenador en intervalos de 10°C. Los valores deben aparecer con un fondo de un determinado color en función de las siguientes condiciones:

- Los valores comprendidos entre 0 y 20°C aparecen en azul.
- Los valores comprendidos entre 20 y 60°C aparecen en verde.
- Los valores comprendidos entre 60 y 80°C aparecen en amarillo.
- Los valores comprendidos entre 80 y 90°C aparecen en naranja
- El valor 100°C aparece en rojo.

5.- Crea una página que muestre el siguiente dibujo al ser solicitada.

```

*
***
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

## Obtención de datos de un formulario

### Métodos de envío de datos.

Los formularios son elementos existentes en HTML que permiten recoger y enviar información al servidor Web.

```
<form action="control.php" method="post">  
</form>
```

- El atributo **action** determina la página o script de PHP al que van dirigidos los datos.
- El atributo **method** determina el tipo de petición enviada al servidor y el modo en que se envían los datos del formulario. Los posibles valores son "get" y "post".

Los datos son enviados al servidor a través de una petición HTTP (*HTTP Request*). Esta petición puede ser de dos tipos en función del valor del atributo "method".

### Método GET

En el método GET los datos se envían concatenándolos al final de la dirección de la página a la que han de enviarse dando lugar a una cadena de consulta o *querystring*.

**<direccion>?<clave>=<valor>&<clave>=<valor>&<clave>=<valor>....**

**Ejemplo:** El siguiente formulario:

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="ISO-8859-1">  
<title>Insert title here</title>  
</head>  
<body>  
<form action="control.php" method="get">  
  NOMBRE: <input type="text" name="userID"><br />  
  CLAVE: <input type="password" name="password"><br />  
  <input type="submit" value="enviar">  
</form>  
</body>  
</html>
```

Los datos se enviarían al servidor mostrando la siguiente URL en la barra de direcciones del navegador:

<http://localhost/prueba/control.php?userID=roger&password=petrov>

Cada clave se corresponde con el valor del atributo **name** del control seguido de su correspondiente valor.

El envío de datos empleando el método **GET** se considera inseguro:

- Los datos son visibles en la barra del navegador y pueden ser modificados.
- Impone un límite en la cantidad de datos que pueden enviarse.

Para el envío de datos delicados tales como claves, contraseñas, o datos de gran tamaño se requiere el uso del método **POST**

### Método POST

En el método POST los datos son enviados dentro de la petición HTTP. Este modo es mucho más seguro puesto que los datos no son visibles ni fácilmente modificables:

```
Request URL: http://localhost:8080/AjaxTest/Login
Request Method: POST
Status Code: 200 OK
▼ Request Headers view source
Accept: */*
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Connection: keep-alive
Content-Length: 23
Content-type: application/x-www-form-urlencoded
Host: localhost:8080
Origin: http://localhost:8080
Referer: http://localhost:8080/AjaxTest/ajaxpost.html
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7
▼ Form Data view URL encoded
userId: abc
password: abc
```

### Las variables superglobales \$\_GET / \$\_POST

Se denominan variables superglobales a un tipo especial de variables definidas por PHP que son accesibles desde cualquier código. Se caracterizan por tener identificadores en mayúsculas con guión bajo como prefijo: `$_GET`, `$_PUT`, `$_SERVER`, `$_COOKIES`, `$_SESSION`, `$_FILES...`, etc.

La mayoría de estas variables se corresponden con matrices asociativas que contienen datos diversos referentes al servidor, el navegador del cliente, datos enviados desde el navegador, errores sucedidos durante la ejecución de PHP..., etc.

Para la obtención de datos enviados desde un formulario se emplean las siguientes variables superglobales en función del método empleado:

- **\$\_GET** — Contiene valores pasados desde formulario por GET
- **\$\_POST** — Contiene valores pasados desde formulario por POST



Para obtener el valor asociado a cualquier elemento de un formulario debe emplearse la siguiente sintaxis indicando el valor del atributo 'name' del elemento del formulario cuyo valor desea recuperarse:

`$_GET["atributo_name"]` → Si los datos son enviados mediante `method='get'`

`$_POST["atributo_name"]` → Si los datos son enviados mediante `method='post'`

Ambas variables son matrices asociativas en las que se almacena una clave y un valor por cada control del formulario.

**(\*) Las matrices asociativas se verán más adelante.**

### Recuperación de datos de formulario.

#### Cajas de texto

Sea un formulario con una caja de texto:

Edad:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <form action="control.php" method="post">
    Edad:
    <input id="txtEdad" name="campoEdad" type="text" /><br />
    <input id="submit" title="Enviar" type="submit" value="Enviar Datos" />
  </form>
</body>
</html>
```

Para recuperar el valor de la caja de texto llamamos a la variable superglobal indicando el valor de su atributo "name" en la página: "control.php":

```
$edad = $_POST["campoEdad"];
```

El valor devuelto siempre será una cadena. Si se ha dejado vacío la caja de texto se obtiene una cadena vacía. Si se esperan valores numéricos es preciso **validar** el valor obtenido antes de convertirlo y utilizarlo:

```
<?php
$edad = $_POST["campoEdad"];
// Comprobacion -> Es un número?
if ( is_numeric($edad) ) {
    // OK -> Es un número. Conversión de cadena a número.
    $edad = intval($edad);
    echo "El año que viene cumplirás " . ($edad+1) . " años.";
    // ERROR -> No es un número.
} else {
    echo "El valor indicado no es un número";
}
?>
```

## Botones de opción

Los botones de opción suelen ir en grupos compartiendo el mismo valor del atributo “name” para que sólo uno pueda estar marcado a la vez. El siguiente formulario muestra dos botones de opción para permitir marcar el género de una persona.

Genero: ☐ Hombre ☐ Mujer

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <form action="control.php" method="post">
    Genero:
    <input id="rdoHombre" value="hombre" name="genero" type="radio" />Hombre
    <input id="rdoMujer" value="mujer" name="genero" type="radio" />Mujer
  </form>
</body>
</html>
```

El valor devuelto se corresponde con el del atributo “value” correspondiente al botón de opción marcado en el momento del envío del formulario:

```
$sexo = $_POST["genero"];
```

Si ninguno de los botones de opción está marcado, no se devuelve ningún valor. Por tanto, debe comprobarse siempre primero si existe el valor mediante la función **isset()**:

```
<?php
// Comprobacion -> Se ha seleccionado algún género
if (isset($_POST["genero"])) {
  // Se ha marcado un género
  $genero = $_POST["genero"];
  if ( $genero == "hombre" ) { // botón de opción "HOMBRE" marcado
    echo "Hola señor!";
  }
  if ( $genero == "mujer" ) { // botón de opción "MUJER" marcado
    echo "Hola señora!";
  }
} else {
  // No se ha marcado género alguno
  echo "No se ha seleccionado ningún género";
}
?>
```

## Listas de valores

En el caso de la lista de elementos, el valor devuelto se corresponde con el valor del atributo “value” del elemento `<option>` seleccionado:

Idioma

Castellano
Frances
Aleman
Ingles

```
Idioma
<select id="Select1" name="idioma" size="5">
  <option value="1">Castellano</option>
  <option value="2">Frances</option>
  <option value="3">Aleman</option>
  <option value="4">Ingles</option>
</select>
```

Si no se ha seleccionado ninguna opción, el valor no existirá por lo que debe comprobarse siempre primero con la función `isset()`.

```
<?php
// Comprobacion -> Se ha seleccionado una opción?
if ( isset($_POST["idioma"])) {
    // Idioma seleccionado -> Obtencion del valor de la opcion
    $idioma = $_POST["idioma"];
    // Visualizacion del idioma seleccionado en función del valor devuelto.
    switch($idioma) {
        case 1: {
            echo "Castellano";
        };break;
        case 2: {
            echo "Francés";
        };break;
        case 3: {
            echo "Alemán";
        };break;
        case 4: {
            echo "Inglés";
        };break;
    }
} else {
    // No se ha seleccionado ninguna opción de la lista.
    echo "No se ha seleccionado ningún idioma.";
}
?>
```

En el caso de permitirse la selección de múltiple elementos en la lista; el valor devuelto es una matriz con los valores de todas las opciones seleccionadas:

```
Idioma
<select id="Select1" name="idioma[]" multiple="true" size="5">
  <option value="1">Castellano</option>
  <option value="2">Frances</option>
  <option value="3">Aleman</option>
  <option value="4">Ingles</option>
</select>
```

(\*) En el caso de las listas de selección múltiple, debe indicarse corchetes ‘[]’ al final del valor del atributo “name” para permitir el envío de los valores de varias opciones a la vez.

## Programación Web en PHP

El siguiente código PHP muestra todos los idiomas seleccionados, o un mensaje de error en caso de no haberse seleccionado ninguno:

```
<?php
// Comprobacion -> Se ha seleccionado una opción?
if ( isset($_POST["idioma"])) {
    // Idioma seleccionado -> Obtención de matriz de valores seleccionados.
    $idiomas = $_POST["idioma"];
    // Obtención de la cantidad de valores
    echo "Se han seleccionado ".count($idiomas). " idiomas<br />";
    echo "<ul>";
    // Bucle de obtención y visualización de cada idioma seleccionado.
    foreach ($idiomas as $idioma){
        // Visualización del idioma seleccionados
        switch($idioma) {
            case 1: {
                echo "<li>Castellano</li>";
            };break;
            case 2: {
                echo "<li>Francés</li>";
            };break;
            case 3: {
                echo "<li>Alemán</li>";
            };break;
            case 4: {
                echo "<li>Inglés</li>";
            };break;
        }
    }
    echo "</ul>";
} else {
    // No se ha seleccionado ninguna opción de la lista.
    echo "No se ha seleccionado ningún idioma.";
}

?>
```

El resultado podría ser el siguiente:

Se han seleccionado 2 idiomas

- Castellano
- Alemán

**(\*) El recorrido de matrices lo veremos en el siguiente anexo**

### Casillas de verificación

En el caso de las casillas de verificación existen dos posibilidades.

Una casilla con su propio valor en su atributo *name*:

Estado civil: Casado ☐

```
Estado civil: Casado
<input type="checkbox" name="estado" value="casado"/>
```

## Programación Web en PHP

En este caso, el valor devuelto coincide con el del atributo 'value' si la casilla es marcada. En caso contrario no se retorna ningún valor:

```
<?php
// Comprobación
if ( isset($_POST["estado"])) {
    // Casilla marcada
    echo "Esta casado";
} else {
    // Casilla no marcada
    echo "No está casado.";
}
?>
```

Varias casillas de verificación con el mismo valor en su atributo "name" terminado en corchetes "[]":

Intereses:

☒ Videojuegos

☒ Cine

☐ Teatro

```
<br/> Intereses: <br/>
<input type="checkbox" name="intereses[]" value="videojuegos"/>Videojuegos<br />
<input type="checkbox" name="intereses[]" value="cine"/>Cine<br />
<input type="checkbox" name="intereses[]" value="teatro"/>Teatro<br />
```

En este caso, el valor devuelto es una matriz con los valores del atributo "values" de todas las casillas marcadas. Si no se marca ninguna no se retorna valor, por lo que debe comprobarse primero la función **isset()**:

```
<?php
// Comprobacion -> Se ha seleccionado una opción?
if ( isset($_POST["intereses"])) {
    // Idioma seleccionado -> Obtención de matriz de valores seleccionados.
    $intereses = $_POST["intereses"];
    // Obtención de la cantidad de valores
    echo "Se han seleccionado ".count($intereses). " intereses<br />";
    echo "<ul>";
    // Bucle de obtención y visualización de cada idioma seleccionado.
    foreach ($intereses as $interes){
        // Visualización del idioma seleccionados
        echo "<li>".$interes."</li>";
    }
    echo "</ul>";
} else {
    // No se ha seleccionado ningún interes.
    echo "No se ha seleccionado ningún interes.";
}
?>
```

### Recuperación de datos de una *QueryString*

La recuperación de datos concatenados en una URL se realiza empleando la variable superglobal `$_GET` indicando el identificador del valor:

```
$_GET["identificador"]
```

Sea por ejemplo:

<http://localhost/prueba/control.php?nombre=pepe&edad=34>

Los valores “*nombre*” y “*edad*” se recuperarían del siguiente modo:

```
<?php
// Recuperación del valor 'nombre'
echo $_GET["nombre"]."<br />";
// Recuperación del valor 'edad'
echo $_GET["edad"]."<br />";
?>
```

## Seguridad en recuperación de datos

Uno de los principales puntos de ataque a una aplicación Web son los valores introducidos a través de formularios y querystrings.

Una vez obtenidos los datos deben realizarse una serie de operaciones para evitar amenazas contra la aplicación web

### Desinfección y filtrado de valores

Consiste en impedir que un usuario mal-intencionado introduzca en los valores secuencias de código HTML o *Javascript* que corrompan las páginas web al ser incluidos para su mostrado.

**Ejemplo:** Supóngase el siguiente formulario que solicita su nombre al usuario:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="control.php" method="post">
  NOMBRE: <input type="text" name="nombre"><br />
  <input type="submit" value="enviar">
</form>
</body>
</html>
```

El formulario envía los datos a la página “control.php” que muestra un mensaje de bienvenida al usuario generado por el siguiente código PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      $nombre = $_POST["nombre"];
      echo "Hola ", $nombre;
    ?>
  </body>
</html>
```

Si introducimos el nombre “pepe” en el formulario, la página “control.php” nos mostrará la siguiente página:

Hola Pepe

in embargo, si ponemos por nombre un código HTML como “<h1>PEPE</h1>”, el resultado sería el siguiente:

Hola

### Encabezado

Esto como poco rompería el diseño de la página web de saludo.

Sin embargo, si ponemos por nombre un script de Javascript maligno como: “<script>window.location.href=window.location.href</script>”, provocaríamos que el navegador del usuario entrase en un bucle recargando la página web sin fin.

Esta amenaza puede neutralizarse empleando la función **htmlspecialchars()**. Esta función transforma ciertos caracteres para que una cadena no pueda interpretarse como código por los navegadores:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Recuperacion del valor 'nombre' filtrado.
      $nombre = htmlspecialchars($_POST["nombre"]);
      echo "Hola ", $nombre;
    </body>
  </html>
```

Si intentamos insertar ahora una sección de código HTML o Javascript, el resultado es el siguiente sin más efectos.

Hola <script>window.location.href=window.location.href</script>

### La función **filter\_input()**

Versiones modernas de PHP incluyen una extensión de PHP dedicada al filtrado de datos con sus propias funciones especializadas. Entre ellas destaca la función **filter\_input()** que permite filtrar aplicando diferentes filtros.

**<valor> = filter\_input( <tipo>, <clave>, <filtro> )**

El parámetro **<tipo>** indica la procedencia del valor a filtrar y admite predefinidas como valores. Si se trata de un valor procedente de un formulario enviado mediante POST, debemos indicar **INPUT\_POST**. Si se trata de un valor procedente de una *querystring* enviado desde un formulario mediante GET debemos indicar **INPUT\_GET**.

El campo **<clave>** se corresponde con la clave del valor que desea filtrarse. El parámetro **<filtro>** es opcional e indica el tipo de filtrado a aplicar.



**Ejemplo:** Aplicando la función `filter_input()` a la recogida de valores enviados desde formulario, el código de recuperación de datos de los ejemplos anteriores debería reescribirse de la siguiente manera:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Recuperacion del valor 'nombre'
      $nombre = filter_input(INPUT_POST, 'nombre', FILTER_SANITIZE_SPECIAL_CHARS);
      echo "Hola ", $nombre;
    ?>
  </body>
</html>
```

Para más información sobre los la extensión de filtrado y la función `filter_input()` puede consultarse la página web oficial de PHP:

<http://php.net/book.filter>

<http://php.net/manual/es/function.filter-input.php>

## Validación

Consiste en comprobar que el usuario ha dado valores y que son lo que se esperan.

- Comprobar que se han introducido los valores obligatorios.
- Comprobar que se ha seleccionado una opción, casilla o botón de opción cuando es obligatorio hacerlo.
- Comprobar que los valores son del tipo que han de ser: *números, decimales, fechas, correos electrónicos*, que están dentro de un rango indicado, y que tienen una longitud determinada.

Para esto pueden emplearse los nuevos controles de formulario de HTML5 y sus características de validación, o scripts de *Javascript* para validar los datos antes de enviarlos al servidor.

***En cualquier caso, las validaciones deben realizarse siempre igualmente en el lado del servidor asumiendo que en el navegador podrían NO ejecutarse.***

## Recarga de formularios

Un formulario con recarga es un formulario generado desde una página PHP que se envía los datos así mismo para validarlos hasta que éstos son correctos, momento en el se redirige al usuario a otra página de la aplicación web.

El código PHP debe diferenciar cuando el formulario es solicitado por primera vez, y cuando es una solicitud del propio formulario enviándose los datos ya introducidos. Para ello, debe comprobarse la cantidad de valores que llegan en las variable superglobales `$_GET` / `$_POST` mediante la función `count()`:

### Ejemplo:

Supóngase que tenemos una página `control.php` que muestra un formulario solicitando un nombre de usuario y una contraseña. Si el usuario introduce el usuario “`cipsa`” y la clave “`profesor`”, es redireccionado a la página `ok.htm`; en caso contrario vuelve a mostrarse el formulario con los valores introducidos anteriormente y un mensaje de error “usuario no reconocido”.

```
<?php
$usuario = "";
$clave = "";
$error = false;
// Comprobacion: ¿Hay valores en $_POST ?
if ( count($_POST) > 0 ) {
    // Hay valores --> Obtencion de los valores.
    $error = true;
    $usuario = filter_input(INPUT_POST, "nombre", FILTER_SANITIZE_SPECIAL_CHARS);
    $clave = filter_input(INPUT_POST, "clave", FILTER_SANITIZE_SPECIAL_CHARS);
    // Comprobacion de nombre de usuario y clave.
    if ( $usuario == "alumno" && $clave == "cipsa" ) {
        // Correctos --> Redireccionamiento del usuario a página: ok.html
        header('Location: ok.html');
    }
}

?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            USUARIO:
            <input id="txtUsuario" name="nombre" type="text"
                value="<?php echo $usuario; ?>" /><br />

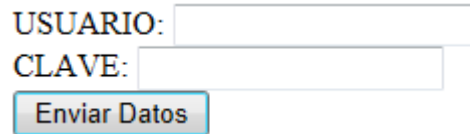
            CLAVE:
            <input id="txtClave" name="clave" type="text"
                value="<?php echo $clave; ?>" /><br />

            <input id="Submit1" title="Enviar datos" type="submit" value="Enviar Datos" />
        </form>

        <?php
            if ($error) {
                // Si $error es TRUE -> Se muestra mensaje.
                echo "<h1>Usuario incorrecto</h1>";
            }

        ?>
    </body>
</html>
```

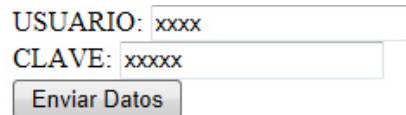
La página comienza con una sección de código PHP que determina si la página es solicitada por el usuario, o si se trata de una recarga de la misma. Si es la primera vez, el formulario se muestra vacío:



USUARIO:   
CLAVE:

Al pulsarse el botón “Enviar Datos”, se envían los datos a la propia página provocando su *recarga*. En tal caso la variable global **`$_POST`** contiene más de cero valores y se comprueba el valor de los campos `$_usuario` y `$_clave`.

Si los valores insertados no se corresponden con “*cipsa*” y “*profesor*” se vuelve a mostrar el formulario pero incluyendo los valores introducidos previamente.



USUARIO:   
CLAVE:

### Usuario incorrecto

Si el nombre y contraseña se corresponden con los correctos, se envía al navegador del usuario una cabecera de redireccionamiento dirigida la página *ok.htm*.

## Obtención de información del usuario

El usuario se comunica con el servidor mediante el envío de peticiones HTTP. Estas peticiones contienen datos tales como el navegador del usuario, su dirección IP,.. etc. Estos valores pueden obtenerse mediante la variable superglobal **`$_SERVER`**.

**`$_SERVER`** es una matriz asociativa que almacena datos sobre el servidor Web, el navegador del usuario, y el motor de ejecución de PHP. Estos valores se obtienen mediante unas claves ya definidas:

- **`SERVER_ADDR`**: Contienen la dirección IP del servidor Web.
- **`REMOTE_ADDR`**: Contiene dirección IP del equipo del usuario.
- **`HTTP_USER_AGENT`**: Contiene la identificación del navegador web empleado por el usuario.
- **`DOCUMENT_ROOT`**: Contiene la ruta absoluta del directorio de publicación de Apache en el servidor web.
- **`SCRIPT_FILENAME`**: Contiene la ruta absoluta de la página o script PHP en ejecución.

La totalidad de los valores y claves que almacena **`$_SERVER`** puede examinarse en la web oficial de PHP:

<http://php.net/manual/es/reserved.variables.server.php>

**Ejemplo:** El siguiente código PHP genera una tabla HTML en la que se muestra la dirección IP del usuario e información sobre su navegador:

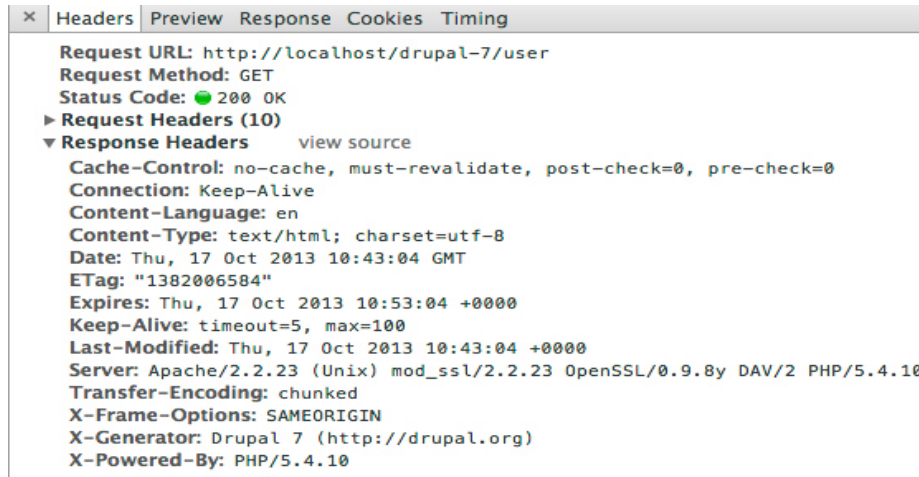
```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table border="1">
  <tr>
    <td>IP:</td><td><?php echo $_SERVER["REMOTE_ADDR"] ?></td>
  </tr><tr>
    <td>NAVEGADOR:</td><td><?php echo $_SERVER["HTTP_USER_AGENT"] ?></td>
  </tr>
</table>
</body>
</html>
```

El resultado mostrado por pantalla será:

IP:	::1
NAVEGADOR:	Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like Gecko

## Generación de cabeceras HTTP

El servidor web se comunica con el navegador del usuario enviando una respuesta HTTP. Esta contiene múltiples datos además del propio código HTML de la página generada.



Extracto del contenido de un paquete de respuesta HTTP

La primera línea contiene el código de respuesta mediante un valores de tres cifras. En función de la primera cifra los significados son los siguientes:

- 1XX: → Mensaje informativo:
- 2XX: → Mensaje respuesta correcta.
- 3XX: → Mensaje de redireccionamiento para el navegador del cliente.
- 4XX: → Mensajes de error por parte del navegador.
- 5XX: → Mensajes de error por parte del servidor.

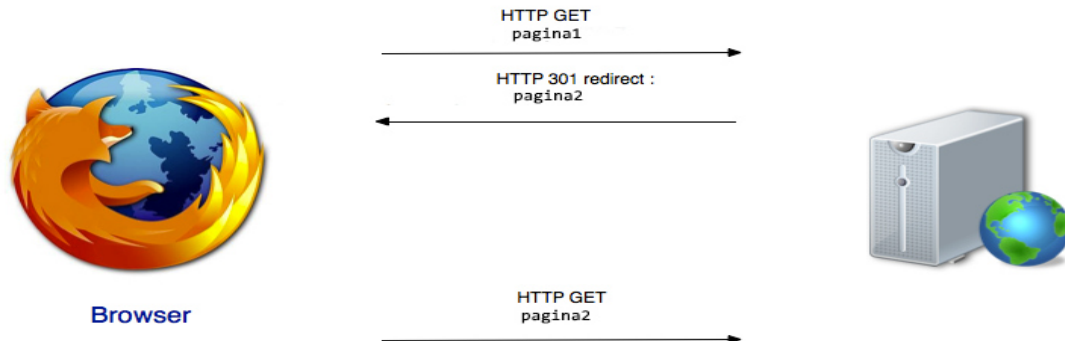
Cada uno de estos mensajes informa al navegador del resultado de la petición enviada previamente. A continuación van las cabecera con información acerca del servidor y la página o recurso enviado. La parte final del mensaje es el cuerpo en el que se envía el código HTML de la página solicitada, o el recurso solicitado.

Algunos códigos de estado concretos comúnmente empleados son los siguientes:

- HTTP 404** → Página no encontrada. La página solicitada por el navegador no existe.
- HTTP 403** → No permitido. La página solicitada no es accesible para el usuario.
- HTTP 500** → Fallo del servidor. El servidor no ha podido generar la página solicitada.
- HTTP 302** → Redireccionamiento. El navegador debe solicitar la página indicada.
- HTTP 200** → Correcto.

## Redireccionamiento

El redireccionamiento es la operación por la que el servidor web envía al una respuesta HTTP con código de estado 302 que le obliga a solicitar otra página indicada:



Para enviar un determinado código de estado desde servidor puede emplearse la función **header()** de PHP. Es importante destacar que esta función debe ejecutarse antes que se envíe ningún código al navegador del usuario, por lo que debe estar antes que cualquier etiqueta HTML o comando PHP que genere código. De lo contrario, se producirá un error.

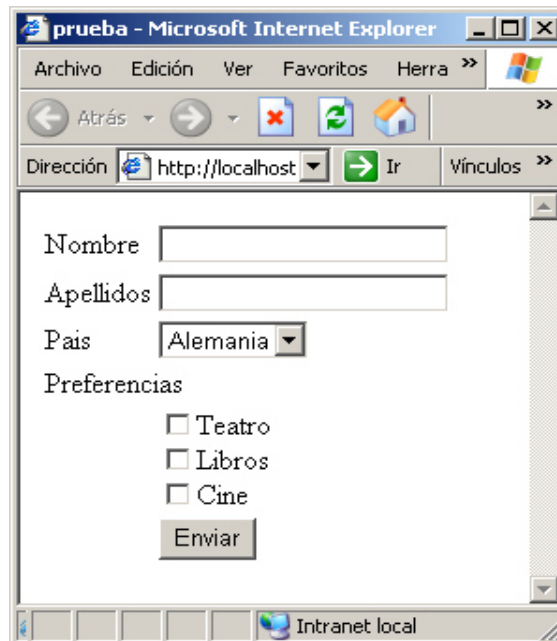
**Ejemplo:** El siguiente código muestra una página provista de un formulario en la que se solicita al usuario su nombre y contraseña. Los datos son enviados al propio formulario el cual redirecciona al usuario a la página “ok.html” o “error.html” en función de los valores indicados.

```
<?php
// Comprobacion: ¿Hay valores en $_POST ?
if ( count($_POST) > 0 ) {
    $usuario = filter_input(INPUT_POST, "nombre", FILTER_SANITIZE_SPECIAL_CHARS);
    $clave = filter_input(INPUT_POST, "clave", FILTER_SANITIZE_SPECIAL_CHARS);
    // Comprobacion de nombre de usuario y clave.
    if ( $usuario == "alumno" && $clave == "cipsa" ) {
        // Correctos --> Redireccionamiento página ok.html.
        header('Location: ok.html');
    } else {
        // Incorrecto --> Redireccionamiento página error.html
        header('Location: error.html');
    }
}

?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            USUARIO:
            <input id="txtUsuario" name="nombre" type="text"/><br />
            CLAVE:
            <input id="txtClave" name="clave" type="text"/><br />
            <input id="Submit1" title="Enviar datos" type="submit" value="Enviar Datos" />
        </form>
    </body>
</html>
```

## Prácticas

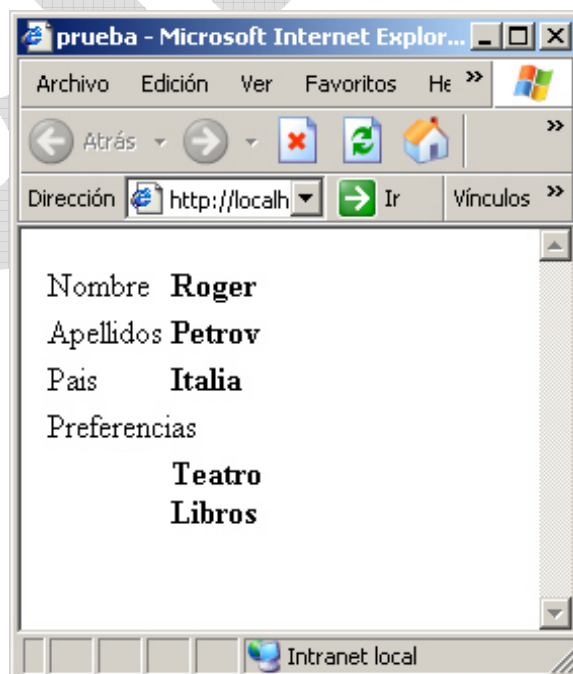
1.- Crea una página HTML de nombre “form.htm” con el siguiente diseño empleando controles simples HTML:



A screenshot of a Microsoft Internet Explorer window titled "prueba - Microsoft Internet Explorer". The address bar shows "http://localhost". The form contains the following elements:

- Nombre:
- Apellidos:
- Pais:
- Preferencias:
  - ☐ Teatro
  - ☐ Libros
  - ☐ Cine
- Enviar:

Al pulsar el botón “Enviar” el formulario deberá enviar los datos a la página “datos.php” empleando el método POST. Diseña a continuación dicha página para que muestre los valores introducidos por el usuario en cada uno de los campos del formulario:



A screenshot of a Microsoft Internet Explorer window titled "prueba - Microsoft Internet Explor...". The address bar shows "http://localh...". The form displays the submitted data:

- Nombre: **Roger**
- Apellidos: **Petrov**
- Pais: **Italia**
- Preferencias:
  - Teatro**
  - Libros**

## Programación Web en PHP

2.- Crear una página HTML con el siguiente formulario para tramitar las compras de una frutería virtual.

Producto			
<input type="radio"/> Judías <input type="radio"/> Garbanzos <input type="radio"/> Lentejas	Cantidad <input type="text"/> Kgs	Tarjeta Visa <input type="checkbox"/>	<input type="button" value="ENVIAR DATOS"/>

El botón “ENVIAR DATOS” debe invocar la página “factura.php” que debe mostrar los siguientes datos:

### ***FACTURA***

Producto: Garbanzos  
Unidades: 23  
VISA: SI

**Importe:** 12.50€

El cálculo del importe se hará en base a las siguientes reglas: El Kilo de Judías vale 2€, el de garbanzos vale 2'5€, y el de lentejas vale 1'25€. Si la cantidad es mayor de 10 kilos se aplica un descuento del 2%, y si es mayor de 50 Kilos se aplica un descuento del 10%. Adicionalmente, si el pago se realiza con tarjeta VISA, se aplica un descuento adicional del 5%.

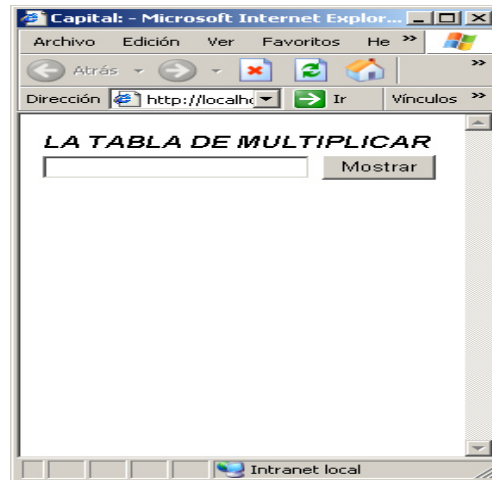
3.- Crea una página mediante PHP que permita introducir una cantidad de euros, y devuelva el desglose en las distintas monedas. Las cantidades indicadas de monedas y billetes debe incrementarse según se van introduciendo nuevas cantidades.

The screenshot shows a Mozilla Firefox browser window displaying a web page titled "DESGLOSE DE MONEDAS". The page has a form with a label "Cantidad a desglosar:" followed by a text input field. Below this is a list of currency denominations: 10000, 5000, 2000, 1000, 500, 200, 100, 50, 25, 10, 5, 2, and 1. Each denomination is followed by a text input field for the count, with the number '0' visible in each. The denominations 5000, 1000, 200, 50, 10, and 2 are highlighted in yellow. At the bottom of the list is a button labeled "Ejecutar Desglose". The browser's status bar at the bottom indicates "Terminado".



## Programación Web en PHP

4.- Crea una página Web que muestre la tabla de multiplicar de un número indicado por el usuario. Cuando el usuario solicite por primera vez la página debe mostrarse un formulario semejante el siguiente:



The screenshot shows a Microsoft Internet Explorer window titled 'Capital: - Microsoft Internet Explor...'. The address bar shows 'http://localh...'. The page content is titled 'LA TABLA DE MULTIPLICAR' and contains a text input field and a button labeled 'Mostrar'.

Al pulsar el botón “mostrar” deberá comprobarse que el valor introducido es un número comprendido entre 1 y 10 ambos incluidos. En tal caso se muestra debajo la tabla de multiplicar correspondiente. En caso contrario, deberá mostrarse un mensaje de error explicativo.



The screenshot shows the same web browser window, but now the input field contains the number '5' and the 'Mostrar' button is disabled. Below the input field, a multiplication table for the number 5 is displayed.

5 x 0	0
5 x 1	5
5 x 2	10
5 x 3	15
5 x 4	20
5 x 5	25
5 x 6	30
5 x 7	35
5 x 8	40
5 x 9	45

5.- Crea una página web llamada operaciones.php que presente un formulario con dos cajas de texto y cuatro botones llamados “Suma”, “Resta”, “Producto”, “División”.

Al pulsarse los botones deberá mostrarse en la misma página el resultado de la operación indicada entre los valores introducidos. En caso de que alguno de los valores introducidos sea nulo, o vacío; deberá mostrarse un mensaje de error.

6.- Crea una página Web que muestre el siguiente formulario. La página deberá llamarse datos.php.

**TABLA DE MUESTRAS**

muestra 1	<input type="text"/>
muestra 2	<input type="text"/>
muestra 3	<input type="text"/>
muestra 4	<input type="text"/>
muestra 5	<input type="text"/>
muestra 6	<input type="text"/>
muestra 7	<input type="text"/>
muestra 8	<input type="text"/>
muestra 9	<input type="text"/>
muestra 10	<input type="text"/>
<input type="button" value="Calcular"/>	

Al pulsar el botón “Calcular”, la página debe comprobar que se ha introducido un valor en todos los campos de texto y que dichos valores son en todos los casos números. Si algún campo no cumple los requisitos deberá mostrarse con fondo rojo y un mensaje al lado que indique “Campo Vacio” o “Valor no válido” según sea el caso.

Si todos los campos cumplen la validación deberá redireccionarse al usuario a la página resultados.php donde se han de mostrar los siguientes valores en función de los datos introducidos por el usuario:

- El valor más grande.
- El valor más pequeño.
- El sumatorio de todos los valores introducidos.
- La media.

### **Funciones**

Cuando se programa en cualquier lenguaje siempre es conveniente separar de alguna forma aquellas secciones de código que se emplean repetidamente o que se ocupan de tareas muy concretas. Con ello se busca tener un código más ordenado y corto reutilizando código en vez de repetirlo. Para conseguirlo se emplean las **funciones**.

Dentro de PHP pueden diferenciarse tres tipos distintos de funciones.

- **Funciones nativas del lenguaje:** Estas son funciones ya definidas como parte de PHP. Entre ellas pueden encontrarse funciones que permiten manipular matrices, cadenas, conocer el tipo de una variable, o si está definida... etc.
- **Funciones de extensión del lenguaje:** Estas son funciones ya definidas en librerías externas que permiten operaciones especiales como enviar un correo electrónico, conectarse a un servidor FTP, generar un documento PDF...etc.
- **Funciones definidas por el usuario:** Estas son funciones desarrolladas por el programador que desempeñan tareas específicas dentro de la aplicación a la que pertenecen. Estas funciones pueden agruparse en ficheros dando lugar a *librerías* que pueden emplearse en otras aplicaciones web.

El uso de funciones es adecuado por las siguientes ventajas:

- **Modularidad:** El uso de funciones permite dividir las tareas de un programa en diferentes funciones cada una de las cuales se ocupa de una parte.
- **Reusabilidad:** Las funciones pueden invocarse para que realicen su tarea tantas veces como sean necesarias evitando tener que duplicar código.
- **Legibilidad:** El uso de funciones estructura el código haciendo que sea más fácil de entender y mantener.

## Declaración de funciones

Las funciones se definen mediante la palabra clave **function**:

```
function <nombre>(<parámetros>)  
{  
    // código de la función  
    return <variable_valor>;  
}
```

El valor **<nombre>** se corresponde con el identificador de la función. Este debe cumplir las mismas normas que los de las variables ( debe identificar la tarea implementada por la función ), salvo que no va precedido por el signo \$.

### Parámetros de una función

Los **<parámetros>** son el conjunto de valores que requiere la función para realizar su labor. Pueden ser uno, varios, o ninguno. Cada parámetro se declara con un identificador lo mismo que una variable:

```
// Función sin parámetros.  
function DevolverFecha() { ... }  
// Función que recibe un único parámetro  
function Incrementar( $valor ) { ... }  
// Función que recibe dos parámetros  
function Sumar( $valor1, $valor2 ) { ... }
```

Al llamar a una función deben indicarse un valor (también llamado *argumento*) para cada parámetro. Los argumentos pueden ser un literal, una variable, una expresión, o el *retorno* de otra función. Los argumentos se asignan a los parámetros en orden de declaración. El primer argumento va al primer parámetro, el segundo al segundo, y así sucesivamente.

```
// Invocación de la función suma pasando dos valores literales.  
sumar( 2, 3 );  
// Invocación pasando una variable y un literal.  
sumar( $a, 2 );  
// Invocación pasando el resultado de una expresión aritmética.  
mostrar( $a + $b - 5 );
```

Si se llama a una función pasando un número de parámetros inferior al requerido por la función se produce un error.

Si se llama a la función pasando un número de parámetros superior al requerido ésta se ejecuta tomando los parámetros necesarios e ignorando el resto.

### Parámetros predeterminados

Es posible indicar un valor predeterminado para un parámetro, el cual se emplea al llamar a la función sin indicar ningún argumento. Para indicar un parámetro predeterminado basta con asignarle un valor en la propia declaración de la función:

**Ejemplo:** Esta función recibe tres parámetros, donde el último tiene como valor predeterminado el 100.

```
function Sumar( $a, $b, $c = 100 ) {  
    return $a + $b + $c;  
}
```

La siguiente llamada es correcta. Los parámetros \$a, \$b y \$c reciben respectivamente los valores 10,20 y 30:

```
$resultado = sumar( 10,20,30 );
```

En este caso, los parámetros \$a, \$b reciben los valores 10 y 20. El parámetro \$c recibe el valor predeterminado 100. No se produce ningún error:

```
$resultado = sumar( 10,20 );
```

Esta última invocación provocaría un error ya que el segundo parámetro no es predeterminado y no se ha indicado ningún argumento:

```
$resultado = sumar(10 );
```

Una función puede tener tanto parámetros normales como predeterminados, pero no pueden declararse parámetros normales tras uno predeterminado.

**Ejemplo:** La siguiente declaración no es válida; los parámetros \$a y \$b tienen valores predeterminados, pero \$c no.

```
function Sumar( $a = 100, $b= 100, $c ) {  
    return $a + $b + $c;  
}
```

## Funciones con listas de parámetros

Las funciones convencionales definen una serie limitada de parámetros algunos. Para llamarlas debe indicarse un argumento por parámetro salvo en el caso de los parámetros con valores predeterminados. Las funciones con lista de parámetros son aquellas que no definen una serie de parámetros concretos y admiten una cantidad indefinida de argumentos.

Estas funciones se definen sin ningún parámetro definido, y pueden invocarse pasando cualquier número de argumentos. Para recuperar el valor de los argumentos dentro de la función pueden emplearse las siguientes funciones de PHP:

- `int func_num_args ( void )`  
→ Devuelve la cantidad de argumentos dados en la llamada.
- `array func_get_args ( void )`  
→ Devuelve una matriz con todos los argumentos recibidos
- `mixed func_get_arg ( int $arg_num )`  
→ Devuelve el argumento recibido en la posición indicada por `$arg_num`, siendo 0 el primero.

**Ejemplo:** El siguiente código muestra la función `sumatorio()` que recibe una lista de parámetros y retorna la suma de los argumentos indicados en la llamada.

```
<?php
function sumatorio() {
    $r = 0;
    // Bucle de obtención de argumentos
    for( $i = 0; $i < func_num_args(); $i++ ) {
        // Obtención y sumatorio de argumento.
        $r = $r + func_get_arg($i);
    }
    return $r;
}
// Llamada con 4 valores
echo sumatorio(2,3,4,5)."<br/>";           // Muestra 14
// Llamada con 2 valores
echo sumatorio(2,10,4,5)."<br/>";          // Muestra 21
// Llamada sin valores
echo sumatorio()."<br/>";                  // Muestra 0
?>
```

La función emplea un bucle **for** para obtener e ir sumando los argumentos dados. Para ello se emplea la función **func\_get\_arg()** indicando como argumento la variable de control **\$i**. El bucle es limitado por el valor devuelto por la función **func\_num\_args()**.

El uso de listas de parámetros son útiles cuando se desean crear funciones que procesan un número variables de argumentos.

### Retorno de una función

Una función puede retornar un valor como resultado. En tal caso; el valor retornado se indica mediante la palabra clave **return** dentro su código:

**Ejemplo:** La siguiente función recibe dos parámetros y retorna la suma de ambos.

```
function sumar( $valor1, $valor2 ) {  
    return $valor1 + valor2;  
}
```

El valor retornado por una función puede ser de cualquier tipo y venir dado por un literal, una variable, una expresión o el resultado de una función.

Las funciones que retornan valores deben llamarse desde la parte derecha de una asignación indicando en el lado izquierdo una variable que tome el valor devuelto.

**Ejemplo:** La siguiente llamada para a la función “*sumar*” pasa los valores 3 y 5 a los parámetros *\$valor1* y *\$valor2* respectivamente, y recoge en la variable *\$r* el resultado.

```
$r = sumar( 3, 5 );
```

## Paso de parámetros.

El paso de parámetros hace referencia al modo en que los valores dados como argumentos pasan a los parámetros de la función. Esto es relevante cuando se emplean variables como argumentos:

### Paso por Valor:

Cada parámetro recibe una copia del valor de la variable indicada como argumento. Si se modifica el valor del parámetro dentro de la función, el cambio no afecta al valor de la variable dada como argumento:

**Ejemplo:** En el siguiente código la función *intercambio()* cruza los valores de los parámetros \$a y \$b. Sin embargo, el cambio no afecta al valor de las variables \$x e \$y utilizadas como argumento.

```
<?php
// Función de intercambio de valores
function intercambio( $a, $b ){
    $tmp = $a;
    $a = $b;
    $b = $tmp;
    // Se intercambian los
    // valores de $a y $b.
}

$x = 1;
$y = 2;
echo $x." - ".$y."<br />"; // Muestra 1 - 2
intercambio( $x, $y);     // Llamada a la función
echo $x." - ".$y."<br />"; // Muestra 1 - 2
?>
```

### Paso por Referencia

Cada parámetro recibe una referencia a la variable indicada como argumento. La referencia contiene la dirección de memoria en la que se aloja la variable, por lo que, si se modifica el parámetro dentro de la función, se modifica la variable dada como argumento.

El paso por referencia se indica anteponiendo el signo **&** delante de aquellos parámetros que se desean reciban su valor por referencia:

```
<?php
// Función de intercambio de valores
function intercambio( &$a, &$b ){
    $tmp = $a;
    $a = $b;
    $b = $tmp;
    // Se intercambian los
    // valores de $a y $b.
}

// Código principal
$x = 1;
$y = 2;
echo $x." - ".$y."<br />"; // Muestra 1 - 2
intercambio( $x, $y);    // Llamada a la función
echo $x." - ".$y."<br />"; // Muestra 2 - 1
?>
```



En este caso, los parámetros de la función \$a y \$b reciben la referencia de las variables \$x e \$y indicadas como argumentos. Al modificar \$a y \$b se modifica también \$x e \$y; por lo que tras la ejecución de la función sus valores aparecen intercambiados.

El paso por referencia puede emplearse para hacer que una función devuelva más de un valor modificando el de los argumento.

**Ejemplo:** La función **check\_numero()** recibe un parámetro \$cadena con un valor de tipo cadena, y un parámetro por referencia &\$numero donde almacena el valor convertido a número. La función retorna además un valor lógico indicando si la conversión fue posible.

```
<?php
function check_numero( $cadena, &$numero ) {
    // Comprobacion -> El valor es numérico.
    if ( is_numeric($cadena) ) {
        // Es numérico -> Conversion y modificacion de $numero
        $numero = intval($cadena);
        return true;
    } else return false;
}

// Código principal
// Valor a convertir
$entrada = "dsfds";
if ( check_numero($entrada, $valor) == true ) {
    // Conversion OK -> Muestra valor convertido
    echo $valor;
} else {
    // Conversion ERROR.
    echo "Error. valor no válido";
}

?>
```

## Prototipado de funciones

El prototipo de una función es la declaración formal ( no de código ) que determina:

- El nombre de la función.
- Los parámetros que recibe y los tipos de datos esperados.
- El tipo de información que devuelve como resultado.

La web oficial de PHP contiene información de todas las funciones de PHP incluyendo sus prototipos e información de cada parámetro y valor devuelto.

Ejemplo: El siguiente código muestra el prototipo de la función ***filter\_input()***, tal como viene descrita en la web de PHP:

<http://php.net/manual/es/function.filter-input.php>

```
mixed filter_input ( int $type ,  
                    string $variable_name  
                    [, int $filter = FILTER_DEFAULT  
                    [, mixed $options ]] )
```

La primera palabra (***mixed***) determina el tipo de valor devuelto por la función. En este caso el tipo “mixed” indica que puede ser cualquier tipo. (tipo mixto). Si la función no devuelve ningún valor se indica el tipo “void”.

Los parámetros se indican separados por comas indicando el tipo de valor esperado y el identificador del parámetro. Los tipos son los siguientes:

- “***boolean***” → Valor lógico.
- “***integer***” → Valor entero.
- “***double***” → Valor decimal.
- “***string***” → Cadena de caracteres.
- “***array***” → Matriz de valores.
- “***object***” → Objeto.
- “***resource***” → Recurso de PHP.
- “***callable***” → Función pasada como argumento.

Si los parámetros aparecen entre corchetes significa que son opcionales. Si van acompañados de un valor, este es su valor predeterminado si no reciben argumento.

En el caso de la función ***filter\_input***, ésta recibe un primer parámetro (***\$type***) obligatorio de tipo numérico entero, seguido de un parámetro (***\$variable\_name***) también obligatorio de tipo cadena, y dos parámetros más opcionales: ***\$filter*** de tipo entero con el valor de la constante ***FILTER\_INPUT*** como valor predeterminado, y ***\$options*** de tipo mixto sin valor predeterminado.

## Ámbito de variables

El ámbito de una variable es la región en el código donde existe y puede utilizarse.

- **Variables locales** → Son aquellas declaradas dentro de una función. Sólo son accesibles dentro de la función.
- **Variables globales** → Son aquellas declaradas en la raíz de código ( fuera de cualquier función ). Son accesibles desde cualquier línea de código salvo dentro de las funciones.
- **Variables superglobales** → Las variables superglobales definidas por PHP ( \$\_POST, \$\_GET, \$\_SERVER ) son accesibles tanto fuera como dentro de las funciones.

**Ejemplo:** El siguiente código muestra una función valor que declara una variable local \$v, la cual es accesible dentro de la función, pero no fuera.

```
<?php
function valor()
{
    $v = 10;
    echo "Dentro funcion: $v <br/>"; // Muestra valor '10'
}
// Código principal
Valor();
echo "Fuera funcion: $v <br/>"; // Muestra error variable no definida.
?>
```

En el ejemplo, la última sentencia genera un error debido a que su declaración está comprendida dentro del código de la función, por lo que resulta inexistente en el exterior.

**Ejemplo:** El siguiente código muestra como una variable declarada a nivel global es accesible desde cualquier punto del código pero no en el interior de las funciones:

```
<?php
$v = 10;
function valor()
{
    echo "Valor dentro: $v<br/>"; // Muestra error variable no definida.
}
// Código principal
Valor();
echo "Valor fuera: $v<br/>"; // Muestra valor '10'
?>
```

En el ejemplo; el valor mostrado por la última sentencia es 10 puesto que ese es el valor asignado a la variable global \$v. Dentro de la función se muestra un error de variable no definida al no ser accesible.

### Palabra clave *'global'*

Al indicar la palabra clave global delante de la declaración de una variable se indica que ya existe en un ámbito superior. En el caso de las funciones, si se desea acceder a una variable global debe declararse anteponiendo la palabra clave **global**:

```
<?php
    $v = 10;

    function valor()
    {
        global $v;
        echo "Valor dentro: $v<br/>"; // Muestra valor '10'
    }

    // Código principal
    valor();
    echo "Valor fuera: $v<br/>"; // Muestra valor '10'
?>
```

La variable global es modificada desde dentro de la función de modo que el nuevo valor puede ser empleado fuera por otras funciones.

### Variables estáticas

Las variables locales es que se crean y destruyen cada vez que se llama a la función. Las variables estáticas son un tipo variables locales que mantienen su valor entre llamadas a una función. Las variables estáticas se declaran anteponiendo la palabra clave **static**:

**Ejemplo:** Supóngase la siguiente función **contador()**. Cada vez que se invoca declara una variable local con el valor 0 e incrementa su valor retornándolo:

```
<?php
    function contador() {
        $veces = 0;
        $veces++;
        return $veces;
    }

    echo contador()."<br>"; // Muestra 1
    echo contador()."<br>"; // Muestra 1
    echo contador()."<br>"; // Muestra 1
?>
```

El resultado devuelto por la función es siempre el mismo ya que la variable local **\$veces** se crea y destruye con cada llamada a la función.

Si declaramos la variable **\$veces** como *estática*, se crea con la primera llamada pero no destruye manteniendo el valor para las siguientes llamadas:

```
<?php
    function contador() {
        static $veces = 0;
        $veces++;
        return $veces;
    }

    echo contador()."<br>"; // Muestra 1
    echo contador()."<br>"; // Muestra 2
    echo contador()."<br>"; // Muestra 3
?>
```

**Ejemplo:** Supóngase una aplicación web “Prueba” provista de una página que muestra un formulario en el que el usuario introduce un valor numérico y se muestra la tabla de multiplicar del valor indicado al pulsar el botón “Mostrar tabla de multiplicar”.

TABLA

Fichero c:\xampp\htdocs\prueba\multiplicar.php:

```
<?php
    $valor = "";
    $ok = true;
    if ( count($_POST) > 0 ) {
        $valor = filter_input(INPUT_POST, "valor", FILTER_SANITIZE_SPECIAL_CHARS);
        if ( is_numeric($valor) ) {
            $valor = intval($valor);
        } else {
            $ok = false;
        }
    }
}

?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            <label for="valor">TABLA</label>
            <input type="text" id="valor" name="valor"/>
            <input type="submit" value="Mostrar tabla de multiplicar">
        </form>
        <div>
            <?php
                if ( $valor != "" ) {
                    echo "<table>";
                    for($n = 1; $n <= 10; $n++ ) {
                        $producto = $n * $valor;
                        echo "<tr><td>$valor * $n = $producto</td></tr>";
                    }
                    echo "</table>";
                } elseif ( $ok == false ) {
                    echo "ERROR - VALOR NO VALIDO";
                }
            ?>
        </div>
    </body>
</html>
```

Para generar el código HTML correspondiente a la tabla de multiplicar podemos definir una función **mostrarTabla()** que incorpore el código recibiendo como parámetro **\$tabla** el valor de la tabla a mostrar:

```
function mostrarTabla( $tabla ) {
    echo "<table>";
    for($n = 1; $n <= 10; $n++ ) {
        $producto = $n * $tabla;
        echo "<tr><td>$tabla * $n = $producto</td></tr>";
    }
    echo "</table>";
}
```

Para validar si el usuario ha introducido un valor numérico podemos crear una función **validar()** que reciba como parámetros **\$id** el identificador del valor a, y un parámetro por referencia **&\$valor** que asigna al argumento el valor numérico convertido. La función retorna un valor lógico cierto si la conversión es correcta, o falso sino.

```
function validar( $id, &$valor ) {
    $cadena = filter_input(INPUT_POST, $id, FILTER_SANITIZE_SPECIAL_CHARS);
    if ( is_numeric($cadena) ) {
        $valor = intval($cadena);
        return true;
    } else {
        return false;
    }
}
```

Usando estas dos funciones el código de la página queda del siguiente modo:

Fichero: c:\xampp\htdocs\prueba\multiplicar.php:

```
<?php
function mostrarTabla( $tabla ) {
    echo "<table>";
    for($n = 1; $n <= 10; $n++ ) {
        $producto = $n * $tabla;
        echo "<tr><td>$tabla * $n = $producto</td></tr>";
    }
    echo "</table>";
}

function validar( $id, &$valor ) {
    $cadena = filter_input(INPUT_POST, $id, FILTER_SANITIZE_SPECIAL_CHARS);
    if ( is_numeric($cadena) ) {
        $valor = intval($cadena);
        return true;
    } else {
        return false;
    }
}

$valor = "";
$ok = true;
if ( count($_POST) > 0 ) {
    $ok = validar("valor", $valor);
}

?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <form action="" method="post">
        <label for="valor">TABLA</label>
        <input type="text" id="valor" name="valor"/>
        <input type="submit" value="Mostrar tabla de multiplicar">
    </form>
    <div>
        <?php
        if ( $valor != "" ) {
            mostrarTabla( $valor );
        } elseif ( $ok == false ) {
            echo "ERROR - VALOR NO VALIDO";
        }
        ?>
    </div>
</body>
</html>
```

### Variables a funciones.

PHP permite asignar una función como valor para una variable.

El siguiente código muestra una función *sumar()* asignada a una variable *\$fn* como valor. La función *sumar()* pasa entonces a ser invocable a través de la variable *\$fn*:

```
// Declaración de la funcion
function sumar($a, $b) {
    return $a + $b;
}

// Asignacion a la variable $fn
$fn = sumar;

// Llamada a la funcion a través de la variable.
$resultado = $fn(2,3);
echo $resultado;
```

## ***Inclusión de archivos***

En una aplicación web PHP podemos distinguir dos tipos de archivos PHP por su contenido:

- *Páginas web dinámicas* → Estos son archivos que incluyen HTML junto con secciones de código PHP. El código HTML define el diseño de la página en el que se incluye el contenido generado por las secciones de código PHP anidadas.
- *Scripts de PHP* → Estos son archivos que sólo incluyen código PHP. Estos pueden generar el código HTML de una página al completo, o realizar otras operaciones como acceder a bases de datos, manejar ficheros, validar datos del usuario..., etc.

Generar una página web mediante PHP puede implicar bastantes operaciones, como validar los datos enviados del usuario, recuperar datos solicitados de una base de datos, generar el código HTML para mostrar los resultados.

La programación de todas estas tareas puede descomponerse en funciones, pero para que todo el código no tenga que estar en un mismo archivo se emplea la inclusión de archivos. La inclusión permite agregar al código de una página o script de PHP, el código de otra página o script existente.

La división de un proyecto en diferentes archivos favorece la reutilización del código y la estructuración de la aplicación haciendo que cada fichero incluya el código o funciones necesarias para ciertas operaciones como: acceder a bases de datos, gestionar ficheros, generación páginas, etc.

### **Ámbitos de variables en inclusión**

Las variables globales definidas en el archivo incluido son accesibles para el código del archivo que lo incluye y viceversa.

Las variables globales definidas en el código de un archivo incluido dentro de una función se comportarán como variables locales de la función.

Dado que un archivo puede contener tanto funciones, como secciones sueltas de código PHP, como HTML; deben incluirse siempre los indicadores `<?php` y `?>` delimitando el código PHP contenido.



## Funciones de inclusión de archivos

PHP define dos funciones para la inclusión de archivos: ***include()*** y ***require()***.

La función ***require()*** incluye el contenido de un archivo de igual manera que ***include()***. Este fichero se localiza en relación a la ubicación de la página solicitada.

La diferencia está en que con ***include()*** si el archivo no se encuentra se genera un mensaje de advertencia pero el código continúa ejecutándose. En el caso de ***require()***, se genera un error fatal y se interrumpe la ejecución con un mensaje de error.

### ***include()***

Esta función incluye el código de un archivo dentro del de otro en punto exacto donde se invoca la función:

```
include (<nombre_archivo>):
```

El parámetro ***<nombre\_archivo>*** debe referirse al nombre del archivo a incluir. PHP buscará el archivo en todas las carpetas incluidas en la directiva de configuración ***include\_path*** del archivo de configuración ***php.ini***.

Si ***<nombre\_archivo>*** se corresponde con la URL de un archivo existente en otro servidor debe estar habilitada la directiva ***allow\_url\_fopen*** dentro del archivo ***php.ini*** para permitirse la inclusión.

### ***include\_once()***

```
include_once(<nombre_archivo>);
```

Esta función es equivalente a ***include()***, pero se diferencia en que impide la inclusión del mismo archivo más de una vez. Esta situación puede darse al incluir archivos que a su vez incluyen otros archivos, con lo que algún archivo podría resultar incluido de forma inadvertida más de una vez.

### ***require()***

```
require(<nombre_archivo>);
```

Equivale a ***include()*** salvo que si el archivo no se encuentra provoca un error y detiene la ejecución del código.

### ***require\_once()***

```
include_once(<nombre_archivo>);
```

La función es equivalente a ***include\_once()***. Evita la inclusión por duplicado de un archivo y provoca un error en caso de no encontrarse éste.

**Ejemplo:** En el ejemplo anterior del formulario que muestra la tabla de multiplicar del valor indicado, ahora podemos pasar las funciones **validar()** y **mostrarTabla()** a un fichero de código aparte ( *funciones.php* ) e incluirlo en la página mediante la función **require()**.

Fichero *c:\xampp\htdocs\prueba\funciones.php*:

```
<?php
function mostrarTabla( $tabla ) {
    echo "<table>";
    for($n = 1; $n <= 10; $n++ ) {
        $producto = $n * $tabla;
        echo "<tr><td>$tabla * $n = $producto</td></tr>";
    }
    echo "</table>";
}

function validar( $id, &$valor ) {
    $cadena = filter_input(INPUT_POST, $id, FILTER_SANITIZE_SPECIAL_CHARS);
    if ( is_numeric($cadena) ) {
        $valor = intval($cadena);
        return true;
    } else {
        return false;
    }
}
```

Fichero *c:\xampp\htdocs\prueba\multiplicar.php*:

```
<?php
require "funciones.php";

$valor = "";
$error = false;
if ( esRecarga() == true ) {
    $valor = obtenerValorNumero("valor");
    if ( $valor === false ) {
        $error = true;
    }
}

?>

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            <label for="valor">TABLA</label>
            <input type="text" id="valor" name="valor"/>
            <input type="submit" value="Mostrar tabla de multiplicar">
        </form>
        <div>

            <?php
            if ( $valor != "" ) {
                mostrarTabla( $valor );
            } elseif ( $error == true ) {
                echo "ERROR - VALOR NO VALIDO";
            }
            ?>

        </div>
    </body>
</html>
```


La ventaja de extraer las funciones vistas en el *funciones.php* e incluirlo en la página web es que éste podría incluirse en otras páginas para reutilizar sus funciones.


## La carpeta de inclusión ( *include\_path* )

La carpeta de inclusión es una carpeta especial para almacenar archivos de PHP para ser incluidos en otros archivos. Su ubicación es determinada por la directiva *include\_path* del archivo de configuración de PHP (*php.ini*). Su ubicación puede obtenerse mediante la función *get\_include\_path()*.

(\*) En el caso de XAMPP, la carpeta de inclusión se ubica de manera predeterminada en la carpeta: "c:\xampp\php\PEAR".

Cuando las funciones *include()* y *require()* no localizan el fichero a incluir ambas intentan localizarlo en la carpeta de inclusión. En el ejemplo anterior si cambiamos el nombre del fichero *funciones.php*, veremos que se muestran los siguientes mensajes:

 Warning: require(funciones.php): failed to open stream: No such file or directory in C:\xampp7\htdocs\prueba\inicio.php on line 12				
Call Stack				
#	Time	Memory	Function	Location
1	0.0030	357072	{main}()	.../inicio.php:0

 Fatal error: require(): Failed opening required 'funciones.php' (include_path='C:\xampp7\php\PEAR') in C:\xampp7\htdocs\prueba\inicio.php on line 12				
Call Stack				
#	Time	Memory	Function	Location
1	0.0030	357072	{main}()	.../inicio.php:0

(\*) El primer mensaje es de advertencia indicando que no se ha encontrado el archivo en la ubicación local, el segundo indica que no se ha encontrado dentro de la carpeta de inclusión. Este mensaje es crítico en el caso de la función *require()*, y de advertencia en el caso de *include()*.

La ubicación de la carpeta de inclusión puede modificarse, o agregarse nuevas ubicaciones mediante la función de PHP *set\_include\_path()*. Esta función requiere rutas absolutas.

**Ejemplo:** Supóngase que en el ejercicio anterior creamos una carpeta de inclusión con el nombre *includes*, y dentro copiamos el fichero *funciones.php*. Ahora para que la sentencia *require("funciones.php")* funcione debemos agregar la ruta absoluta a la carpeta *includes* como carpeta de inclusión:

```
// Obtencion ruta absoluta de la carpeta include.
$ruta = $_SERVER['DOCUMENT_ROOT']. "/prueba/includes";
// Agregado de la ruta a la directiva include_path.
set_include_path(get_include_path() . PATH_SEPARATOR . $ruta);
```

La clave "*DOCUMENT\_ROOT*" de la variable superglobal *\$\_SERVER* devuelve la ruta absoluta de la carpeta de publicación del servidor web que debemos componer a la ruta relativa de la carpeta *includes* ( *prueba/includes* ), para obtener la ruta absoluta:

C:\xampp\htdocs\prueba\includes

La ruta se agrega a las carpetas de inclusión ya existentes mediante *set\_include\_path()* concatenándola a la ya existente mediante el separador ";" ( *PATH\_SEPARATOR* ).

## Programación Web en PHP

La ubicación de los archivos quedaría del siguiente modo:

Fichero `c:\xampp\htdocs\prueba\includes\funciones.php`:

```
<?php
function mostrarTabla( $tabla ) {
    echo "<table>";
    for($n = 1; $n <= 10; $n++ ) {
        $producto = $n * $tabla;
        echo "<tr><td>$tabla * $n = $producto</td></tr>";
    }
    echo "</table>";
}

function validar( $id, &$valor ) {
    $cadena = filter_input(INPUT_POST, $id, FILTER_SANITIZE_SPECIAL_CHARS);
    if ( is_numeric($cadena) ) {
        $valor = intval($cadena);
        return true;
    } else {
        return false;
    }
}
```

Fichero: `c:\xampp\htdocs\prueba\multiplicar.php`

```
<?php
// Obtencion ruta absoluta de la carpeta include.
$ruta = $_SERVER['DOCUMENT_ROOT']. "/prueba/includes";
// Agregado de la ruta a la directiva include_path.
set_include_path(get_include_path() . PATH_SEPARATOR . $ruta);

include "funciones.php";
$valor = "";
$ok = true;
if ( count($_POST) > 0 ) {
    $ok = validar("valor", $valor);
}
?>

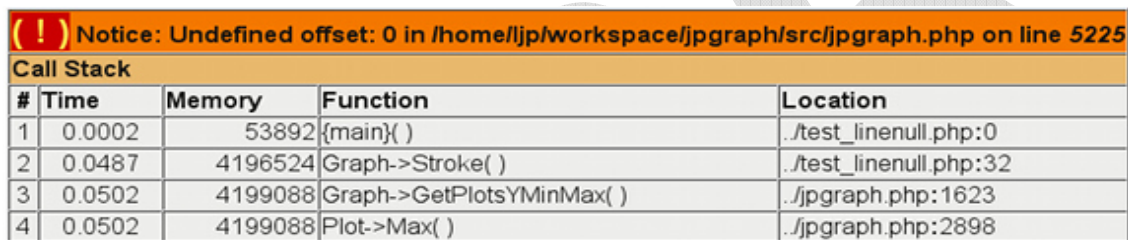
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <form action="" method="post">
            <label for="valor">TABLA</label>
            <input type="text" id="valor" name="valor"/>
            <input type="submit" value="Mostrar tabla de multiplicar">
        </form>
        <div>
            <?php
            if ( $valor != "" ) {
                mostrarTabla( $valor );
            } elseif ( $ok == false ) {
                echo "ERROR - VALOR NO VALIDO";
            }
            ?>
        </div>
    </body>
</html>
```

## Control de errores.

El código PHP puede lanzar errores al ejecutarse en algunas ocasiones como cuando se hace accede a una variable no declarada, se llama a una función inexistente, o se incluye un fichero que no es accesible. Estos se agrupan en:

- **Errores** (*ERROR*) → Indican errores serios que no permiten continuar la ejecución del código.
- **Avisos** (*WARNING*) → Indica errores menores que permiten continuar la ejecución a pesar de ellos.
- **Notificaciones** (*NOTICE*) → Indica posibles errores que no afectan a la ejecución.

De manera predeterminada, el intérprete de PHP muestra estos errores al usuario de la aplicación incluyéndolos en la página generada:



(!) Notice: Undefined offset: 0 in /home/ljp/workspace/ljpgraph/src/ljpgraph.php on line 5225				
Call Stack				
#	Time	Memory	Function	Location
1	0.0002	53892	{main}()	./test_linenull.php:0
2	0.0487	4196524	Graph->Stroke()	./test_linenull.php:32
3	0.0502	4199088	Graph->GetPlotsYMinMax()	./ljpgraph.php:1623
4	0.0502	4199088	Plot->Max()	./ljpgraph.php:2898

*Ejemplo de mensaje de notificación.*

Los mensajes indican el tipo de error (*ERROR*, *WARNING*, *NOTICE*), acompañado de un mensaje descriptivo y la información de traza que indica la línea en la que se ha producido el error y las llamadas a funciones que se han ejecutado hasta llegar a ella.

Estos mensajes son útiles para los programadores durante la fase de desarrollo de la aplicación, pero no resultan apropiados en aplicaciones ya finalizadas. Esto puede ajustarse mediante las directiva ***display\_errors*** y ***error\_reporting*** del archivo de configuración de PHP:

La directiva ***display\_errors*** determina si se muestran mensajes de error al usuario. El valor por defecto es mostrar errores, salvo en servidores para aplicaciones finales donde se desaconseja (*Production Value*):

```
; display_errors
; Default Value: On
; Development Value: On
; Production Value: Off
```

La directiva ***error\_reporting*** determina qué tipos de errores, avisos y notificaciones son mostrados.

```
; error_reporting
; Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
; Development Value: E_ALL
; Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT
```

## Programación Web en PHP

Los diferentes valores pueden combinarse para indicar qué errores se muestran un mensaje al producirse. Algunos valores como **E\_ALL** representan a todos los errores, avisos y notificaciones juntos.

(\*) El operador “|” es la disyunción y permite añadir tipos de errores. El operador “&” es la conjunción y se emplea para eliminar tipos de errores junto con el operador “~”

Valor	Constante	Descripción	Nota
1	<b>E_ERROR</b> ( <a href="#">integer</a> )	Errores Fatales en tiempo de ejecución. Éstos indican errores que no se pueden recuperar, tales como un problema de asignación de memoria. La ejecución del script se interrumpe.	
2	<b>E_WARNING</b> ( <a href="#">integer</a> )	Advertencias en tiempo de ejecución (errores no fatales). La ejecución del script no se interrumpe.	
4	<b>E_PARSE</b> ( <a href="#">integer</a> )	Errores de análisis en tiempo de compilación. Los errores de análisis deberían ser generados únicamente por el analizador.	
8	<b>E_NOTICE</b> ( <a href="#">integer</a> )	Avisos en tiempo de ejecución. Indican que el script encontró algo que podría señalar un error, pero que también podría ocurrir en el curso normal al ejecutar un script.	
16	<b>E_CORE_ERROR</b> ( <a href="#">integer</a> )	Errores fatales que ocurren durante el arranque inicial de PHP. Son como un <b>E_ERROR</b> , excepto que son generados por el núcleo de PHP.	
32	<b>E_CORE_WARNING</b> ( <a href="#">integer</a> )	Advertencias (errores no fatales) que ocurren durante el arranque inicial de PHP. Son como un <b>E_WARNING</b> , excepto que son generados por el núcleo de PHP.	
64	<b>E_COMPILE_ERROR</b> ( <a href="#">integer</a> )	Errores fatales en tiempo de compilación. Son como un <b>E_ERROR</b> , excepto que son generados por Motor de Script Zend.	
128	<b>E_COMPILE_WARNING</b> ( <a href="#">integer</a> )	Advertencias en tiempo de compilación (errores no fatales). Son como un <b>E_WARNING</b> , excepto que son generados por Motor de Script Zend.	
256	<b>E_USER_ERROR</b> ( <a href="#">integer</a> )	Mensaje de error generado por el usuario. Es como un <b>E_ERROR</b> , excepto que es generado por código de PHP mediante el uso de la función de PHP <a href="#">trigger_error()</a> .	
512	<b>E_USER_WARNING</b> ( <a href="#">integer</a> )	Mensaje de advertencia generado por el usuario. Es como un <b>E_WARNING</b> , excepto que es generado por código de PHP mediante el uso de la función de PHP <a href="#">trigger_error()</a> .	
1024	<b>E_USER_NOTICE</b> ( <a href="#">integer</a> )	Mensaje de aviso generado por el usuario. Es como un <b>E_NOTICE</b> , excepto que es generado por código de PHP mediante el uso de la función de PHP <a href="#">trigger_error()</a> .	
2048	<b>E_STRICT</b> ( <a href="#">integer</a> )	Habíltelo para que PHP sugiera cambios en su código, lo que asegurará la mejor interoperabilidad y compatibilidad con versiones 5.4.0 posteriores de PHP de su código.	Desde PHP 5 pero no incluido en <b>E_ALL</b> hasta PHP 5.4.0
4096	<b>E_RECOVERABLE_ERROR</b> ( <a href="#">integer</a> )	Error fatal capturable. Indica que ocurrió un error probablemente peligroso, pero no dejó al Motor en un estado inestable. Si no se captura el error mediante un gestor definido por el usuario (vea también <a href="#">set_error_handler()</a> ), la aplicación se abortará como si fuera un <b>E_ERROR</b> .	Desde PHP 5.2.0
8192	<b>E_DEPRECATED</b> ( <a href="#">integer</a> )	Avisos en tiempo de ejecución. Habíltelo para recibir avisos sobre código que no funcionará en futuras versiones.	Desde PHP 5.3.0
16384	<b>E_USER_DEPRECATED</b> ( <a href="#">integer</a> )	Mensajes de advertencia generados por el usuario. Son como un <b>E_DEPRECATED</b> , excepto que es generado por código de PHP mediante el uso de la función de PHP <a href="#">trigger_error()</a> .	Desde PHP 5.3.0
32767	<b>E_ALL</b> ( <a href="#">integer</a> )	Todos los errores y advertencias soportados, excepto del nivel <b>E_STRICT</b> antes de PHP 5.4.0.	32767 en PHP 5.4.x, 30719 en PHP 5.3.x, 6143 en PHP 5.2.x, 2047 anteriormente

### Configuración de visualización de errores.

La función **`error_reporting()`** permite indicar los errores, avisos y notificaciones que se desea mostrar cuando se produzcan modificando la configuración de la directiva `error_reporting`. Esto es útil cuando no puede modificarse al archivo de configuración.

La función recibe como parámetro un valor entero compuesto a partir de las constantes que representan los errores, avisos y notificaciones que se desean mostrar:

```
// Desactivar toda notificación de error
error_reporting(0);
// Notificar solamente errores de ejecución
error_reporting(E_ERROR | E_WARNING | E_PARSE);
// Notificar E_NOTICE también puede ser bueno (para informar de variables
// no inicializadas o capturar errores en nombres de variables ...)
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
// Notificar todos los errores excepto E_NOTICE
error_reporting(E_ALL ^ E_NOTICE);
// Notificar todos los errores de PHP (ver el registro de cambios)
error_reporting(E_ALL);
// Notificar todos los errores de PHP
error_reporting(-1);
```

Si no se desea mostrar ningún mensaje de error de ningún tipo también es posible simplemente modificar el valor de la directiva “`display_errors`” del archivo de configuración de PHP, o mediante la función **`ini_set()`**.

Las funciones **`ini_get()`** y **`ini_set()`** permiten obtener y modificar desde PHP el valor de las directivas del archivo de configuración de PHP.

**Ejemplo:** El siguiente código desactiva la visualización de errores para el fichero en el que se ejecute:

```
// Asigna el valor false a la visualización de errores
// para este script.
ini_set('display_errors', false);
```

Con la directiva **`display_error`** deshabilitada cualquier aviso o notificación que puedan darse no serán mostradas al usuario. En caso de darse un error crítico se envía al usuario una respuesta HTTP con un código de error 500 ( *Server Error* ).

## Gestión de errores

Gestionar un error es básicamente indicarle al intérprete de PHP qué hacer en caso de que se produzca un error. Para ello se puede emplear la función **`error_handler()`**.

```
mixed set_error_handler ( callable $error_handler
                        [, int $error_types = E_ALL | E_STRICT ] )
```

La función declara una función manejadora (tipo *Callable*) indicada como primer parámetro que será invocada por cada error que se produzca de los tipos indicados en el segundo parámetro (*\$error\_types*). En caso de omitirse el segundo parámetro, la función maneja todos los errores posibles.

La función manejadora los siguientes parámetros de la función `error_handler()` cada vez que es invocada:

```
bool handler ( int $errno , string $errstr
              [, string $errfile
              [, int $errline
              [, array $errcontext ]]] )
```

- *\$errno* → Recibe un valor entero con tipo de error, el valor puede compararse con el de las constantes `E_NOTICE`, `E_WARNING`..., etc.
- *\$errstr* → Recibe una cadena con el mensaje descriptivo.
- *\$errfile* → Recibe la ruta física del fichero de PHP donde ha tenido origen el error.
- *\$errline* → Recibe el nº de línea de código donde surgió el error, aviso o notificación.
- *\$errcontext* → Estado de variables superglobales.

**Ejemplo:** Supóngase el siguiente código PHP:

```
<?php
    $a = 10;
    $b = $a / 0;
    $c = $a + b;
    echo "$a<br />$b<br />$c<br />";
?>
```

Al ejecutarlo se muestran los siguientes mensajes suponiendo que la directiva **`error_reporting`** tenga asignado el valor **`E_ALL`** (mostrar todos los errores, avisos y notificaciones):

(!) Warning: Division by zero in C:\xampp7\htdocs\prueba\inicio.php on line 46				
Call Stack				
#	Time	Memory	Function	Location
1	0.0100	356024	{main}()	...\inicio.php:0

(!) Notice: Use of undefined constant b - assumed 'b' in C:\xampp7\htdocs\prueba\inicio.php on line 47				
Call Stack				
#	Time	Memory	Function	Location
1	0.0100	356024	{main}()	...\inicio.php:0



Ahora supóngase que añadimos una función llamada “*error\_handler*” cuyo nombre pasamos como argumento para la función ***set\_error\_handler()***, para que maneje los errores que se produzcan mostrando un mensaje por pantalla:

```
<?php
// Funcion manejadora de errores
function error_handler( $code,$message,$file,$line,$context ) {
    switch($code) {
        case E_WARNING:
            echo "<h3>AVISO: $message</h3>en $file ($line)"; break;
        case E_NOTICE:
            echo "<h3>DETALLE: $message</h3>en $file ($line)"; break;
    }
};
// Asignacion de nueva funcion manejadora para todos los avisos
set_error_handler("error_handler", E_ALL);

$a = 10;
$b = $a / 0;
$c = $a + b;
echo "$a<br />$b<br />$c<br />";
?>
```

La visualización por pantalla ahora es la siguiente:

**AVISO: Division by zero**

en C:\xampp7\htdocs\prueba\inicio.php (42)

**DETALLE: Use of undefined constant b - assumed 'b'**

en C:\xampp7\htdocs\prueba\inicio.php (43)10

### Errores no manejables.

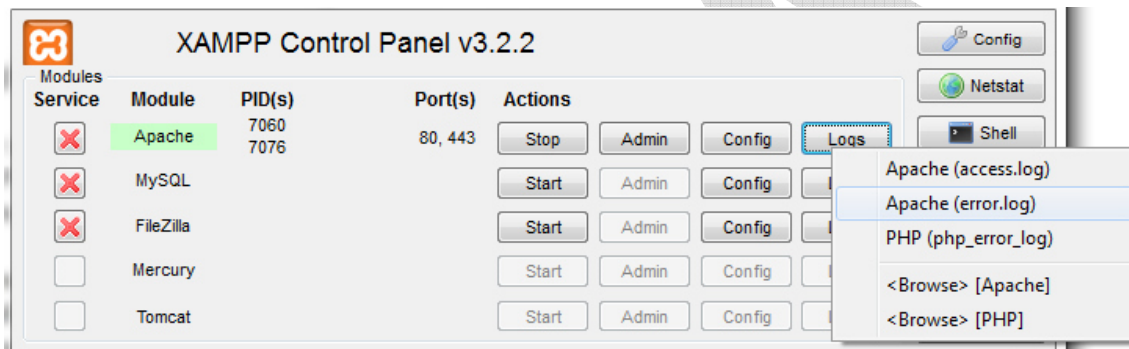
La función *set\_error\_handler()* no puede capturar los errores de los siguientes tipos: ***E\_ERROR***, ***E\_PARSE***, ***E\_CORE\_ERROR***, ***E\_CORE\_WARNING***, ***E\_COMPILE\_ERROR***, ***E\_COMPILE\_WARNING***, y la mayoría de ***E\_STRICT*** ocasionados en el archivo desde donde se llamó a *set\_error\_handler()*.

## Registro de errores

El registro de errores es un mecanismo por el que los errores, avisos y notificaciones de PHP que puedan producirse se registran en un registro de errores sin mostrar mensaje alguno al usuario.

El registro de errores se activa mediante la directiva **log\_errors** del archivo de configuración de PHP. Una vez habilitada los errores pueden registrarse en el archivo de registro del propio servidor Apache (C:\xampp\apache\logs\error), o en el archivo designado por la directiva **error\_log** (C:\xampp\php\logs\php\_error\_log) según la configuración de permisos del servidor web.

En XAMPP ambos archivos son accesibles desde el panel de control pulsando el botón “Logs” de Apache:



Acceso a ficheros de registro de errores de Apache y PHP

## Lanzamiento de mensajes para el registro de errores

PHP permite registrar mensajes en el registro de errores con la función **error\_log()**. Estos mensajes suelen emplearse con fines de depuración de la aplicación de manera totalmente transparente al usuario:

```
bool error_log ( string $message [, int $message_type =0  
                [, string $destination  
                [, string $extra_headers ]]] )
```

**Ejemplo:** El siguiente código lanza un mensaje para el registro de errores.

```
error_log("Mensaje para registro de errores", 0);
```

## Lanzamiento de errores personalizados

Además de los errores lanzados por el intérprete y las funciones del propio PHP (*errores nativos*), también es posible lanzar errores desde nuestro propio código (*errores de usuario*). Estos pueden ser de tipo error, aviso o notificación, y son mostrados igualmente al usuario según la directiva ***error\_reporting***:

Valor	Constante	Descripción	Nota
256	<code>E_USER_ERROR(integer)</code>	Mensaje de error generado por el usuario. Es como un <code>E_ERROR</code> , excepto que es generado por código de PHP mediante el uso de la función de PHP <code>trigger_error()</code> .	
512	<code>E_USER_WARNING(integer)</code>	Mensaje de advertencia generado por el usuario. Es como un <code>E_WARNING</code> , excepto que es generado por código de PHP mediante el uso de la función de PHP <code>trigger_error()</code> .	
1024	<code>E_USER_NOTICE(integer)</code>	Mensaje de aviso generado por el usuario. Es como un <code>E_NOTICE</code> , excepto que es generado por código de PHP mediante el uso de la función de PHP <code>trigger_error()</code> .	
16384	<code>E_USER_DEPRECATED(integer)</code>	Mensajes de advertencia generados por el usuario. Son como un <code>E_DEPRECATED</code> , excepto que es generado por código de PHP mediante el uso de la función de PHP <code>trigger_error()</code> .	Desde PHP 5.3.0

Para lanzar errores de usuario se emplea la función ***trigger\_error()***. La función recibe un primer parámetro (*\$error\_msg*) de tipo cadena con el mensaje de error que se mostrará el usuario, y un segundo parámetro (*\$error\_type*) que indica su tipo según las constantes (***E\_USER\_ERROR***, ***E\_USER\_WARNING***, ***E\_USER\_NOTICE***):

```
bool trigger_error ( string $error_msg
                    [, int $error_type = E_USER_NOTICE ] )
```

**Ejemplo:** El siguiente código genera un aviso personalizado ( ***E\_USER\_WARNING*** ) con el mensaje indicado:

```
if ($divisor == 0) {
    trigger_error("Cannot divide by zero", E_USER_WARNING);
}
```

Los errores, avisos y notificaciones personalizadas se muestran al usuario de manera predeterminada igual que los errores nativos de PHP según la configuración de las directivas ***"display\_errors"*** y ***"error\_reporting"***. De igual modo, estos también se registran en el registro de errores si está habilitada la directiva ***"log\_errors"***.

## Ejercicios

Para la realización de estos y los sucesivos ejercicios debe seguirse esta estructura:

Las funciones deben implementarse en un fichero llamado **funlib.php**. Esta es un archivo con código PHP exclusivamente que no contiene ningún código HTML.

Debe crearse una página PHP **ejercicios.php** provista de código HTML. Esta página debe mostrar una tabla <table> por cada ejercicio. Cada tabla contendrá dos filas: la primera debe mostrar el número del ejercicio, y la segunda el resultado de la llamada a la función PHP solicitada en el ejercicio.

Para poder invocar a las funciones incluidas en **funlib.php** desde **ejercicios.php**, debe referenciarse mediante la función **require\_once**:

### Ejemplo:

#### Código de fichero **funlib.php**

```
<?php
    /**
     * @author SyToo
     * @copyright 2012
     */

    // Ejercicio 1
    function FechaActual() {
        date_default_timezone_set("Europe/Madrid");
        return Date("r");
    }
?>
```

#### Código de página: **pagina.php**

```
<?php
    require_once "funlib.php"
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
        <meta name="author" content="SyToo" />
        <title>Ejercicios</title>
    </head>
    <body>
        <table width="300px">
            <tr><thead>EJERCICIO 1</thead></tr>
            <tr><tbody>
                <?php
                    echo( FechaActual());
                ?>
            </tbody></tr>
        </table>
    </body>
</html>
```

## Programación Web en PHP

**1.-** Los siguientes ejercicios deben implementarse en un fichero único llamado `funlib.php`. Para probar las funciones puede utilizarse una página `index.php`.

**2.-** Crea una función en PHP llamada `println` que muestra por pantalla el valor pasado como parámetro seguido de un salto de línea.

**3.-** Crea una función `aleatorio()` que requiera dos parámetros `$max` y `$min` y retorne un valor aleatorio entero comprendido entre los valores dados. Investiga para ello la función `rand()` en la documentación online de PHP.

**4.-** Crea tres funciones `dia()` y `mes()` que devuelvan respectivamente una cadena con el nombre del día y mes actuales. Investiga para ello la función `date()` en la documentación online de PHP.

*(\*) Para este ejercicio necesitas emplear las funciones `Date` y `date_default_timezone_set` mostradas en el ejemplo anterior. Consulta sus parámetros en la web de referencia de la PHP:*

<http://www.php.net/manual/es/function.date.php>

**5.-** Crea una función `operaciones()` que reciba tres parámetros `$v1`, `$v2` y `$oper`; y realice las siguientes operaciones:

- Si `$oper` vale 1, la función devuelve la suma de `$v1` y `$v2`.
- Si `$oper` vale 2; la función devuelve la resta de `$v1` y `$v2`.
- Si `$oper` vale 3; la función devuelve el producto de `$v1`, y `$v2`.
- Si `$oper` vale 4; la función devuelve el cociente de `$v1` y `$v2`.

En caso de que no se indique ningún valor para el parámetro `$oper`, debe devolverse la suma de `$v1` y `$v2`.

**6.-** Crea una función `sumatorio()` que requiera un parámetro de entrada y devuelva el sumatorio de todos los valores que se le van pasando con cada invocación de la función.

**7.-** Crea dos funciones `producto_iterativo()` y `producto_recursoivo()` que requieran ambas dos parámetros y devuelvan el producto de los valores dados. Se requiere que la función `producto_iterativo` calcule la multiplicación mediante sumas, y que la función `producto_recursoivo` lo haga mediante llamadas así misma.

**8.-** Crear una función `validarEdad()` que requiera un parámetro de entrada y devuelva un valor de tipo lógico. La función debe devolver un valor lógico cierto si y solo si el valor dado es de tipo entero y está comprendido entre 18 y 65.

**9.-** Crear una función `media()` de múltiples parámetros de entrada que retorne como resultado la media aritmética de todos los valores recibidos en la invocación,

**10.-** Crea una función `tiempo()` con dos parámetros de entrada; `$hora` y `$minuto`. La función debe incrementar en uno el valor del parámetro `$minuto`. Si `$minuto` alcanza entonces el valor 60, debe asignársele el valor 0 e incrementar en uno el valor del parámetro `$hora`.