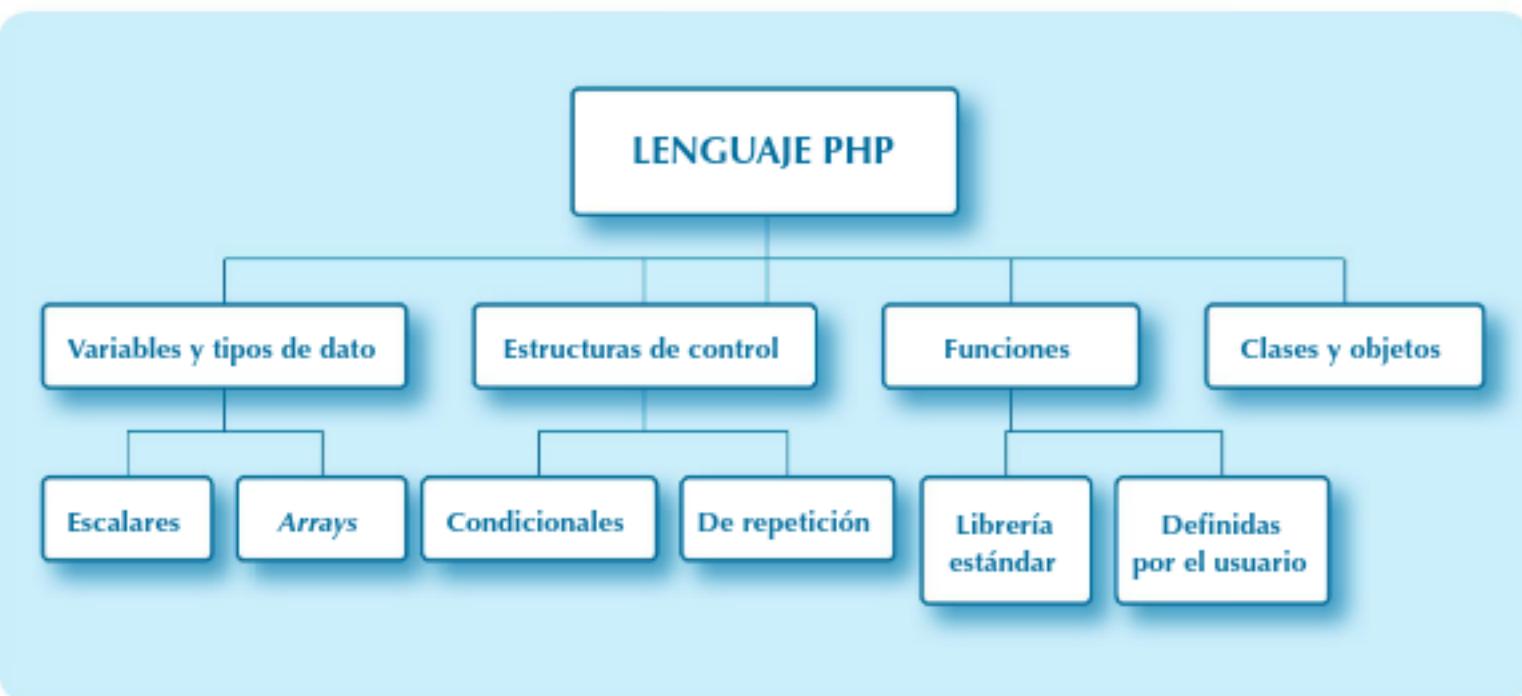


# Introducción al lenguaje PHP

## Objetivos

- ✓ Conocer la sintaxis básica de PHP.
- ✓ Entender cómo se integran PHP y HTML.
- ✓ Describir los tipos de datos existentes en PHP.
- ✓ Manejar las estructuras de control básicas.
- ✓ Aprender a utilizar los *arrays* asociativos.
- ✓ Presentar la notación de objetos en PHP.

## Mapa conceptual



## Glosario

**Array.** Es un tipo de dato compuesto habitual en los lenguajes de programación. Aunque los detalles varían entre lenguajes, en general, se parecen a vectores o listas ordenadas.

**Bucle.** Estructura de programación que permite repetir instrucciones.

**Estructura condicional.** Posibilita ejecutar o no una instrucción según se cumpla una condición.

**Función.** Conjunto de instrucciones que realiza una tarea concreta.

**Librería.** Las funciones relacionadas entre sí se agrupan en librerías o bibliotecas.

**Programación orientada a objetos.** Paradigma de programación basado en la idea de objetos, elementos que agrupan variables y funciones.

**Script.** Programa sencillo, habitualmente ejecutado por un intérprete en lugar de ser compilado.

**Variable.** Posición en la memoria del ordenador identificada por un nombre. La variable almacena datos. Estos datos son el valor de la variable.

## 2.1. PHP y HTML. Código incrustado

PHP es el lenguaje de programación para desarrollo web en el lado del servidor. Desde su aparición en 1994 ha tenido gran aceptación y se puede decir que es lenguaje más extendido para el desarrollo en el lado del servidor. Aunque no es la única opción, lo normal es que el intérprete de PHP sea un módulo del servidor web.

El lenguaje PHP es flexible y permite programar pequeños *scripts* con rapidez. Comparado con lenguajes como Java, requiere escribir menos código y, en general, resulta menos engorroso. La sintaxis de los elementos básicos es bastante parecida a la de lenguajes muy extendidos como Java y C. Por estos motivos, es un lenguaje rápido de aprender para las personas con alguna experiencia en programación.

En este capítulo se presentan la sintaxis y los elementos básicos del lenguaje PHP. Se espera que el lector esté familiarizado con los conceptos básicos de programación estructurada y orientada a objetos.

En el desarrollo web es muy habitual utilizar PHP incrustado dentro ficheros HTML. El código PHP se introduce dentro del HTML utilizando la etiqueta `<?php` para abrir el bloque de PHP y la etiqueta `?>` para cerrarlo.

El ejemplo **hola\_mundo.php** muestra una página HTML completa con un bloque PHP incrustado en las líneas 7-10. El bloque tiene una única sentencia que sirve para mostrar la cadena “Hola mundo”.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Hola mundo</title>
5      </head>
6      <body>
7          <?php
8              echo "Hola mundo";
9          ?>
10     </body>
11 </html>
```

Al solicitar la página al servidor web, el resultado es:

Hola mundo

Si se consulta el código fuente de la página (pulsando **Ctrl + u**), se obtiene:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hola mundo</title>
    </head>
    <body>
        Hola mundo
    </body>
</html>
```

Se puede observar que el servidor web ha modificado una parte del fichero. El bloque PHP ha desaparecido y en su lugar aparece “Hola mundo”. El servidor web procesa los bloques de PHP y los sustituye por su salida, es decir, por lo que muestran las sentencias del bloque. En este caso se utiliza `echo`, que muestra el valor de una variable o de una cadena de texto.



### TOMA NOTA

Para probar los ejemplos del libro hay que instalar el entorno de trabajo como se explica en el capítulo 1. Para probar este ejemplo accede a [http://localhost/cap2/hola\\_mundo.php](http://localhost/cap2/hola_mundo.php). Para el resto de los ejemplos solo hay que cambiar el nombre del fichero. Recuerda arrancar el servidor web.

## 2.2. Sintaxis de PHP

El elemento básico en PHP es el bloque. Un bloque PHP está delimitado por las etiquetas correspondientes y contiene una serie de sentencias separadas por punto y coma.

```
<?php  
    sentencial;  
    sentenciaN;  
? >
```

### RECUERDA

- ✓ Cuando un fichero contiene solo PHP se recomienda no cerrar la etiqueta del último bloque. Puede dar problemas si la respuesta del servidor involucra varios ficheros.

## 2.3. Variables y tipos de dato

Una de las características principales de PHP es que es un lenguaje *no fuertemente tipado*. Esto quiere decir que no es necesario indicar el tipo de dato al declarar una variable. De hecho, las variables no se declaran, se crean la primera vez que se les asigna un valor. El tipo de dato depende del valor con que se inicialicen.

Esto agiliza la escritura de programas, pero también tiene inconvenientes. Si no se presta atención, puede dar lugar a código de baja de calidad y, a medida que las aplicaciones crecen, pueden darse errores difíciles de depurar.

### 2.3.1. Declaración de variables

En PHP los identificadores de las variables van siempre precedidos por el carácter '\$'. El identificador de la variable debe comenzar por una letra o un guion bajo ('\_'), y puede estar formado por números, letras y guiones bajos.

Para declarar una variable, solo hay que asignarle un valor:

```
$nombre = valor;
```

Por ejemplo, esta sentencia declara la variable \$entero, que será de tipo integer porque se inicializa con un entero.

```
$entero = 4;
```

También es posible cambiar el tipo de dato de una variable simplemente asignándole un valor de otro tipo de dato, como se puede ver en el ejemplo **tipos\_dato.php** (líneas 8-12). El ejemplo utiliza la función `gettype()`, que devuelve el tipo de dato de una variable.

```
1 <?php
2 /* declaración de variables */
3 $entero = 4; // tipo integer
4 $numero = 4.5; // tipo coma flotante
5 $cadena = "cadena"; // tipo cadena de caracteres
6 $bool = TRUE; //tipo booleano
7 /* cambio de tipo de una variable */
8 $a = 5; // entero
9 echo gettype($a); // imprime el tipo de dato de a
10 echo "<br>";
11 $a = "Hola"; // cambia a cadena
12 echo gettype($a); // se comprueba que ha cambiado
```

La salida del ejemplo confirma que la variable cambia de tipo de dato.

```
integer
string
```

### 2.3.2. Asignación por copia y por referencia

En principio, la asignación de variables se realiza mediante copia. Es decir, si hacemos:

```
$a = $b;
```

se crea una nueva variable *a* y se le asigna el valor que tenga *b*. Las variables *a* y *b* representan posiciones diferentes de memoria, aunque tengan el mismo valor después de la asignación.

También es posible definir una referencia a una variable utilizando el operador *ampersand*:

```
$var2 = &$var1;
```

En este caso \$var2 no es una nueva variable con el valor de \$var1. Por el contrario, \$var2 apunta a la misma dirección de memoria que \$var1, de manera que \$var1 y \$var2 son en realidad dos nombres para el mismo dato. En el ejemplo **copia.php** se muestra que, al cambiar el valor de una referencia, se modifica también el de la variable referenciada.

```
<?php
$var1 = 100;
$var2 = &$var1; // asignación por referencia
```

```

$var3 = $var1; // asignación por copia
echo "$var2<br>";// muestra 100
$var2 = 300; // cambia el valor de $var2
echo "$var1<br>";// $var1 también cambia
$var3 = 400; // este cambio no afecta a $var1
echo $var1;

```

La salida de este ejemplo será:

100  
300  
300

### 2.3.3. Variables no inicializadas

Si se intenta utilizar una variable antes de asignarle un valor, se genera un error de tipo E\_NOTICE, pero no se interrumpe la ejecución del *script*. Si una variable no inicializada aparece dentro de una expresión, dicha expresión se calcula tomando el valor por defecto para ese tipo de dato. En el ejemplo **no\_init.php**, se puede ver lo que ocurre al utilizar una variable no inicializada dentro de una expresión.

```

1 <?php
2 $var1 = 100;
3 $var3 = 100 + $var2;//$var2 no existe, se toma como 0
4 echo "$var3 <br>"; // muestra 100
5 $var3 = 100 * $var2; // $var2 no existe, se toma como 0
6 echo "$var3 <br>"; // muestra 0

```

En la línea 3, se intenta sumar una variable no inicializada. Como el valor por defecto para un entero es 0, el resultado de la suma es 100. En la línea 5, se intenta multiplicar por una variable no inicializada y, al multiplicar por 0, el resultado es 0.

La salida muestra un mensaje de error por cada intento de utilización de una variable no inicializada.

```

Notice: Undefined variable: var2 in C:\xampp\htdocs\cap2\ejeinicializaciones.php on line 3
100
Notice: Undefined variable: var2 in C:\xampp\htdocs\cap2\ejeinicializaciones.php on line 5
0

```

### 2.3.4. Constantes

Para definir constantes se utiliza la función `define()`, que recibe el nombre de la constante y el valor que queremos darle.

```
define("LIMITE", 1000);
```

Es habitual utilizar identificadores en mayúsculas para las constantes.

### 2.3.5. Tipos de datos escalares

PHP ofrece cuatro tipos de datos escalares: *integer*, *float*, *boolean* y *string*.

#### A) Números

Para representar números enteros se usa el tipo de dato *integer*. El tamaño y los valores máximo y mínimo de un entero dependen de la plataforma, y se pueden conocer mediante los constantes `PHP_INT_SIZE`, `PHP_INT_MAX` y `PHP_INT_MIN`, respectivamente.

Para números reales, se utiliza el tipo *float*. El tamaño también depende de la plataforma, pero suele ofrecer una precisión de 14 decimales. En cualquier caso, la precisión de los *float* presenta los mismos problemas que en otros lenguajes y los redondeos pueden dar sorpresas.

La conversión entre *integer* y *float* es automática. Si se recibe un *float* cuando se espera un *integer*, se trunca. Si al realizar una operación sobre un entero el resultado supera los valores límite o tiene decimales, se convierte a *float*. También se pueden utilizar los operadores de conversión, `(int)` o `(float)`.

El ejemplo `tipos_numericos.php` muestra las diferentes opciones disponibles para literales *integer* o *float* y algunas operaciones entre ellos.

```
<?php
echo PHP_INT_SIZE.'<br>';
echo PHP_INT_MIN.'<br>';
echo PHP_INT_MAX.'<br>';
$a = 0b100; // en binario
$a = 0100; // octal
$a = 0x100; // hexadecimal
$a = 3/2; // la división entre enteros no da problemas
echo $a.'<br>'; // 1.5
$b = 7.5;
$a = (int)$b; //casting a int
echo $a.'<br>'; // 7, se trunca
$b = 7e2; // notación científica
$b = 7E2;
```

#### B) Cadenas

El tipo de dato *string* permite almacenar cadenas de caracteres. Para delimitar una cadena es posible utilizar comillas simples o dobles, pero hay una diferencia. Si se utilizan comillas dobles (también llamadas *comillas mágicas*), las variables que aparezcan dentro de la cadena se sustituirán por su valor. Las comillas dobles son muy prácticas, ya que es más rápido insertar directamente las variables que montar las cadenas con varias concatenaciones.

```
<?php
$var = "Paco";
$a = "Hola $var <br>";
$b = 'Hola $var';
echo $a;
echo $b;
```

La salida será:

```
Hola Paco
Hola $var
Hola Paco
```

En el primer caso, `$var` se sustituye por su valor, “Paco”. En el segundo, se interpreta como texto normal. La última línea del *script* muestra el operador de concatenación entre cadenas, “.”.



#### TOMA NOTA

A la hora de formatear la salida hay que tener en cuenta que esta va a ser procesada como HTML por un navegador web, es decir, los saltos de línea se ignoran y los espacios en blanco consecutivos colapsan. Los caracteres de escape como ‘\n’, ‘\r’ y ‘\t’ son ignorados por el navegador. Para introducir un salto de línea la opción más sencilla es incluir la etiqueta de salto de línea de HTML (“`<br>`”) en la salida, como se puede ver en los ejemplos anteriores.

### C) Booleanos

Este tipo de dato es para variables *booleanas*. Solo pueden tomar los valores TRUE y FALSE, verdadero y falso. Este es el tipo de dato que se obtiene, entre otros casos, como resultado de los operadores de comparación y se utiliza en sentencias condicionales y bucles.

Cuando se espera un valor *booleano* y se recibe otro tipo de dato, se aplican las reglas de conversión recogidas en el cuadro 2.1.

#### CUADRO 2.1

##### Conversión implícita a boolean

Tipo	Valor como boolean
integer	Si es 0 se toma FALSE, en otro caso como TRUE
float	Si es 0.0 se toma FALSE, en otro caso como TRUE
string	Si es una cadena vacía o “0”, se toma como FALSE, en otro caso como TRUE
variables no inicializadas	FALSE
null	FALSE
array	Si no tiene elementos se toma FALSE, en otro caso como TRUE

### D) Otros tipos de dato

Además de los tipos de datos dato escalares en PHP también existen los siguientes tipos de datos:

1. *array*. Para representar colecciones de elementos. Los *arrays* de PHP son muy potentes y se explican con detalle en el apartado 2.2.5.
2. *object*, PHP tiene soporte completo para la programación orientada a objetos. Se explica en el apartado 2.2.8.
3. *callable*. Un tipo de dato especial para representar funciones de *callback*, funciones que se pasan a otras funciones.
4. *null*. El tipo de dato *null* representa una variable que no ha sido asignada. Solo puede tomar un único valor, NULL. Se considera que una variable es de tipo *null* si se le asigna el valor NULL o no está inicializada.
5. *resource*. Este tipo de dato representa recursos externos, como una conexión a una base de datos.

**CUADRO 2.2**  
Tipos de dato en PHP

Tipo	Descripción
integer	Números enteros
float	Números reales en coma flotante
string	Cadenas de caracteres
boolean	Booleanos, TRUE o FALSE
array	Colección de elementos identificados
object	Un objeto es una instancia de una clase
callable	Para las funciones de <i>callback</i>
null	Para representar variables no asignadas
resource	Representa recursos externos

### 2.3.6. Ámbito de las variables

El ámbito de una variable es la parte del código en que esta es visible. Una variable declarada en un fichero PHP está disponible en ese fichero y en los ficheros que se incluyan desde este.

Por otro lado, las funciones definen un ámbito local, de manera que las variables que se declaran en las mismas solo son accesibles desde la propia función. Además, desde la función no se puede acceder a otras variables que no sean las locales o sus argumentos.

Para definir variables globales hay dos opciones. La palabra reservada `global` y la variable predefinida `$_GLOBALS`. Las variables globales son accesibles desde cualquier función o fichero de la aplicación.

### 2.3.7. Variables predefinidas

En PHP hay muchas variables predefinidas disponibles. Contienen información sobre el servidor, datos enviados por el cliente o variables de entorno. Dentro de las variables predefinidas

hay un grupo de ellas, las *superglobales*, que están disponibles en cualquier ámbito. Cada una de ellas guarda información de un tipo.

Por ejemplo, en `$_SERVER` hay información sobre el servidor en el que está alojada la página. El script `global_server.php` muestra algunos de los datos disponibles.

```
<?php
    echo "Ruta dentro de htdocs: ". $_SERVER['PHP_SELF'];
    echo "Nombre del servidor: ". $_SERVER['SERVER_NAME'];
    echo "Software del servidor: ". $_SERVER['SERVER_SOFTWARE'];
    echo "Protocolo: ". $_SERVER['SERVER_PROTOCOL'];
    echo "Método de la petición: ". $_SERVER['REQUEST_METHOD'];
```

Las variables *superglobales* (cuadro 2.3) son muy relevantes para el desarrollo de aplicaciones web y se irán poniendo en práctica a lo largo del libro.

### CUADRO 2.3 Variables superglobales

Nombre	Descripción
<code>\$GLOBALS</code>	Variables globales definidas en la aplicación
<code>\$_SERVER</code>	Información sobre el servidor
<code>\$_GET</code>	Parámetros enviados con el método GET (en la URL)
<code>\$_POST</code>	Parámetros enviados con el método POST (formularios)
<code>\$_FILES</code>	Ficheros subidos al servidor
<code>\$_COOKIE</code>	Cookies enviadas por el cliente
<code>\$_SESSION</code>	Información de sesión
<code>\$_REQUEST</code>	Contiene la información de <code>\$_GET</code> , <code>\$_POST</code> y <code>\$_COOKIE</code>
<code>\$_ENV</code>	Variables de entorno

## 2.4. Comentarios

En PHP se pueden utilizar comentarios:

- De bloque, encerrados entre “`/*`” y “`*/`”.
- De línea, comenzando por “`//`” o por “`#`”.

```
<?php
    //comentario de línea
    $a = 0; // comentario de línea
    /* comentario
    de bloque
    */
    # comentario de una sola línea
```

Como en cualquier lenguaje, comentar adecuadamente el código se considera una buena práctica de programación.

## 2.5. Estructuras de control

PHP cuenta con las estructuras de control habituales en la programación estructurada para realizar sentencias condicionales y de repetición. La sintaxis es muy parecida a la de Java o C.

### 2.5.1. Estructuras condicionales

Las estructuras condicionales de PHP son `if`, `if-else`, `if-elseif` y `switch`.

La sentencia condicional más sencilla es la estructura `if`. La sintaxis general es:

```
if (condición)
    instrucción
```

Si se cumple la condición, se ejecuta la instrucción. Si no se cumple, no se ejecuta. Para agrupar dentro del `if` más de una sentencia, se encierran entre llaves.

```
if (condición){
    instrucción 1
    instrucción n
}
```

La condición es una expresión que se evalúa a verdadero o falso, siguiendo las normas de conversión a `boolean` si es necesario.

En el siguiente ejemplo solo se cumple el segundo `if` y, por tanto, la salida del programa sería “Es mayor que cero”.

```
<?php
$var = 3;
if($var < 0) echo "Es menor que cero";
if ($var > 0){
    echo "Es mayor que cero";
}
```

Cuando se utiliza un `if` se puede añadir un `else`. Las instrucciones dentro del `else` solo se ejecutan cuando la condición del `if` no se cumple.

```
<?php
$var = 3;
if($var < 0){
    echo "Es menor que cero";
} else{
    echo "Es mayor o igual que cero";
}
```

Si se anidan varias sentencias condicionales, se puede usar `elseif`, que es equivalente a `else if`.

```
<?php
$var = 3;
if ($var == 1) {
    echo "Es un uno";
} elseif ($var == 2) {
    echo "Es un dos";
} elseif ($var == 3) {
    echo "Es un tres";
} else{
    echo "No es un uno, ni un dos, ni un tres"
}
```

La primera condición que se cumpla es la que se ejecuta. Si no se cumple ninguna, se ejecuta el `else` final (si lo hay).

Para agrupar varios `if` puede ser útil la estructura `switch`, también habitual en otros lenguajes. El ejemplo `switch.php` es equivalente al anterior, pero más fácil de leer.

```
<?php
$var = 3;
switch($var){
    case 1:
        echo "Es un 1";
        break;
    case 2:
        echo "Es un 2";
        break;
    case 3:
        echo "Es un 3";
        break;
    default:
        echo "No es un 1, ni un 2, ni un 3";
}
```

Según el valor de `$var`, se ejecutará un `case` u otro. La sección `default` se ejecuta cuando no se da ninguno de los `case`.

#### RECUERDA

- ✓ Si no hay un `break` al final de un `case`, la ejecución continúa con el siguiente.

### 2.5.2. Estructuras de repetición

Las estructuras de repetición o bucles sirven para repetir un conjunto de instrucción mientras se dé una condición. PHP cuenta con las estructuras habituales: `for`, `while` y `do-while`, que tienen la misma sintaxis que en Java o C.

Para el bucle **for**, la sintaxis es:

```
for(instrucciones de inicialización; condición; instrucciones de iteración) {
    instrucciones del bucle;
}
```

Las instrucciones de inicialización se realizan una única vez al llegar al bucle. Las instrucciones dentro del cuerpo del bucle se repetirán mientras se cumpla la condición.

Después de ejecutar las instrucciones de inicialización se evalúa la condición. Si se cumple, se ejecutan las instrucciones del bucle y las instrucciones de iteración. Después se vuelve a comprobar la condición. El proceso se repite hasta que la condición deja de cumplirse y a partir de ahí la ejecución continúa con las instrucciones que estén después del bucle.

El ejemplo **bucle\_for.php** muestra un bucle **for** básico que se repite cinco veces. La variable **\$i** se inicializa a cero y su valor se incrementa en uno tras cada iteración. El bucle se repite mientras **\$i** valga menos que cinco, es decir, de cero a cuatro.

```
<?php
    for($i = 0; i < 5; $i = $i + 1){
        echo "$i <br>";
    }
```

El bucle **while** también sigue la sintaxis habitual:

```
while (condición) {
    instrucciones;
}
```

Igual que con el **for**, las instrucciones se ejecutan mientras se cumpla la condición. El **while** no cuenta con instrucciones de inicialización o iteración, pero se pueden añadir antes y al final del bucle, respectivamente. El ejemplo **bucle\_while.php** tiene la misma salida que el anterior.

```
<?php
    $i = 0;
    while($i < 5){
        echo "$i <br>";
        $i = $i +1;
    }
```

El bucle **do-while** es similar, pero la condición se evalúa después de ejecutar las ejecuciones del bucle.

```
do {
    instrucciones;
} while (condición);
```

Este ejemplo es equivalente a los anteriores.

```
<?php
    $i = 0;
    do{
        echo "$i <br>";
        $i = $i +1;
    } while ($i < 5);
```

El bucle `do-while` se diferencia de los anteriores en que las instrucciones dentro del bucle se ejecutan por lo menos una vez, como se puede ver en el siguiente ejemplo:

```
<?php
    $i = 0;
    do {
        echo "En el do-while: $i <br>";
        $i = $i + 1;
    } while ($i < 0);
    while ($i < 0) {
        echo "En el while: $i <br>";
        $i = $i + 1;
    }
}
```

La salida de este ejemplo será:

En el do-while: 0

Dentro de un bucle es posible usar las sentencias `break` y `continue`. La primera sirve para salir del bucle o `switch` en el que aparece. En el ejemplo `break.php`, el bucle acaba antes de llegar a cinco porque al llegar a tres se ejecuta el `break`.

```
<?php
    $i = 0;
    while ($i < 5){
        echo "$i <br>";
        $i++; // es lo mismo que $i = $i + 1;
        if ($i == 3){
            break;
        }
    }
}
```

En PHP la sentencia `break` admite un número, que representa el número de niveles de anidación de los que debe salir. Así se puede salir de un bucle anidado utilizando una única secuencia `break` como se puede ver a continuación.

```
<?php
    echo "Primer for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
            if ($j == 1) {
                break; //es lo mismo que poner break 1
            }
        }
    }
    echo "Segundo for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
            if ($j == 1){
                break 2;
            }
        }
    }
}
```

En el primer bucle anidado se utiliza un `break` sin pasarle ningún número, que es lo mismo que pasarle un uno, y por tanto solo sale del bucle interior. El bucle exterior se repetirá tres veces y el interior seis, dos por cada vez que se ejecute el exterior, con los valores de `$j=0` y `$j=1`. En el segundo bucle anidado, al ejecutarse la línea `x` se acaban los dos bucles. Esto ocurre en la primera iteración del bucle externo y la segunda del interno. La salida será:

Primer for anidado:

```
i: 0 j: 0
i: 0 j: 1
i: 1 j: 0
i: 1 j: 1
i: 2 j: 0
i: 2 j: 1
```

Segundo for anidado:

```
i: 0 j: 0
i: 0 j: 1
```

Esto también se aplica a la sentencia condicional `switch`. Por ejemplo, si un `switch` está dentro de un bucle, utilizando `break 2` se sale de ambos.

```
<?php
    $i = 0;
    echo "Primer switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
    $i = 0;
    echo "Segundo switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break 2;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
```

La salida de este ejemplo es:

Primer switch anidado:

Es un cero

Es un uno

Segundo switch anidado:

Es un cero

La sentencia `continue` fuerza una nueva iteración del bucle. Las instrucciones que estén dentro del bucle, pero después de `continue` no se ejecutan y, si se cumple la condición del bucle, se ejecuta una nueva iteración. Si se trata de un bucle `for`, se ejecutan las instrucciones de autoincremento antes de evaluar la condición.

En el ejemplo `continue.php` se puede observar el funcionamiento de `continue`.

```
<?php
    for($i = 0; $i < 5; $i = $i + 1){
        if ($i == 3){
            continue;
        }
        echo "$i <br>";
    }
```

Cuando `$i` vale tres se ejecuta el `continue`, por lo que el `echo` a continuación no ejecuta esa iteración. La salida será:

0  
1  
2  
4

### Actividad propuesta 2.1



Escribe un programa que calcule el factorial de un número.

Recuerda que el factorial solo está definido para números enteros mayores o iguales que cero.

### 2.5.3. Otras estructuras de control

Es posible incluir otros ficheros utilizando las sentencias `include` y `require`.

```
include "mifichero.php";
require "mifichero.php";
```

Se diferencian solo en el tratamiento de errores. Si no se encuentra el fichero especificado, `include` genera un error de tipo `E_NOTICE`, pero `require` genera un error `E_FATAL`, lo que implica el fin de la ejecución del *script*. Si no hay error, en ambos casos el fichero especificado se

ejecuta. Si el fichero es una librería con funciones, estas se declaran y quedan disponibles para el fichero que lo incluye. Si el fichero contiene sentencias fuera de una función, estas se ejecutan.

El ejemplo **requerir.php** incluye otro fichero.

```
<?php  
    $a = "variable del principal";  
    require "ejerequerido.php";  
    $b = "otra variable del principal";  
    echo "En el script principal";
```

El fichero requerido es el siguiente:

```
<?php  
    echo "En el fichero requerido <br>";  
    echo $a;  
    echo $b;
```

Al solicitar al servidor **cap2/requerir.php** se obtiene esta salida:

En el fichero requerido

variable del principal

Notice: Undefined variable: b in C:\xampp\htdocs\cap2\ejerequerido.php on line 4

En el script principal

Se puede observar que:

- Las instrucciones del fichero requerido se ejecutan.
- Las variables del fichero principal están disponibles en el requerido si se habían inicializado antes de requerirlo. Esto ocurre con \$a, pero no con \$b, que se inicializa después y, por tanto, se muestra un mensaje de error. Es decir, el ámbito de las variables del fichero principal incluye los ficheros requeridos.

También están disponibles las sentencias **require\_once** e **include\_once**, análogas a las anteriores, pero con la diferencia de que solo se ejecutan los ficheros especificados si no han sido incluidos o requeridos con anterioridad.

**TOMA NOTA**

La inclusión de ficheros tiene una diferencia con otros lenguajes. Supongamos que el fichero A incluye al fichero B. Las rutas relativas que aparezcan en B se interpretarán a partir del directorio de A.

Para solucionarlo, en el fichero B se utiliza **dirname(\_\_FILE\_\_)** que devuelve la ruta del fichero:

```
include( dirname(__FILE__)."\\".'fichero.php');
```



Para acabar con las estructuras de control, la sentencia `return` finaliza la ejecución del fichero cuando se encuentra fuera de una función. En el siguiente ejemplo, el `echo` final no se llega a ejecutar, porque se ha ejecutado antes el `return`.

```
<?php
    $a = 0;
    if ($a == 0){
        return;
    }
    echo "Después del if";
```

Si el *script* se estaba ejecutando mediante un `require` o `include`, la ejecución continúa en el fichero que lo incluyó. Si se pasa como argumento un número entero, este se devuelve como valor de retorno.

Si se encuentra dentro de una función, finaliza la ejecución de esta y la función devuelve el argumento del `return`. La ejecución continua en el fichero o función que llama a la función.

## 2.6. Operadores

PHP cuenta con los operadores habituales para operaciones aritméticas, lógicas, de manipulación de cadenas y demás.

Entre los operadores de comparación cabe señalar los operadores “`==`” y “`!=`”, llamados Idéntico y No Idéntico, que no están presentes en todos los lenguajes. El operador Idéntico se usa para comparar dos expresiones y se evalúa como verdadero cuando las dos expresiones tienen el mismo valor y además el mismo tipo de dato. Se diferencia del operador Igual, “`=`”, en que este, cuando las expresiones no tienen el mismo tipo de dato, intenta convertirlas antes de compararlas. El operador Idéntico es útil para evitar sorpresas inesperadas en la conversión de datos. En el ejemplo **identico.php** se puede comprobar la diferencia.

```
<?php
    $a = 3;
    $b = "3";
    if ($a == $b){
        echo "Son iguales <br>";
    }else{
        echo "No son iguales <br>";
    }
    if ($a === $b){
        echo "Son idénticos <br>";
    }else{
        echo "No son idénticos <br>";
    }
```

La primera comparación permite la conversión de tipos, y la cadena “3” se convierte al entero 3. La salida será:

Son iguales  
No son idénticos



## CUADRO 2.4 Operadores

### Operadores de comparación

<code>e1 === e2</code>	Idéntico. Verdadero si las dos expresiones son del mismo tipo y tienen el mismo valor.
<code>e1 == e2</code>	Igual. Verdadero si las dos expresiones son iguales tras la conversión de tipos, si es necesaria.
<code>e1 !== e2</code>	No idéntico.
<code>e1 != e2, e1 &lt;&gt; e2</code>	No igual.
<code>e1 &gt;= e2, e1 &gt; e2, e1 &lt;= e2, e1 &lt; e2</code>	Mayor o igual, mayor, menor o igual, menor.
<code>e1 ?? e2 ?? e3</code>	Comenzando por la izquierda, devuelve la primera expresión no nula.

### Operadores aritméticos

<code>+e1</code>	Como operador unario, sirve para convertir la expresión a <i>integer</i> o <i>float</i> , según corresponda.
<code>-e1</code>	Como operador unario, cambio de signo.
<code>e1 + e2, e1 - e2, e1 * e2, e1 / e2</code>	Suma, resta, multiplicación, división.
<code>e1 % e2</code>	Módulo.
<code>e1 ** e2</code>	Potencia.

### Operadores lógicos

<code>e1 and e2, e1 &amp;&amp; e2</code>	Y. Verdadero si las dos expresiones se evalúan a TRUE.
<code>e1 or e2, e1    e2</code>	O. Verdadero si una o las dos expresiones se evalúan a TRUE.
<code>e1 xor e2</code>	O exclusivo. Verdadero si solo una de las expresiones se evalúa a TRUE.
<code>!e1</code>	Devuelve TRUE si la expresión es FALSE y viceversa.

### Operadores a nivel de bit

<code>e1 &amp; e2, e1   e2, e1 ^ e2, ~e1</code>	Y, O, O exclusivo y negación.
<code>\$var» N</code>	<code>\$var = 5; echo \$var &gt;&gt; 1;</code> Desplaza N bits de \$var hacia la izquierda.
<code>\$var «N</code>	Desplaza N bits de \$var hacia la derecha.

### Operadores de asignación

<code>\$var1 = e1</code>	Asignación por valor.
<code>\$var2 = &amp;\$var2</code>	Asignación por referencia.
<code>\$var += e1, \$var -= e1, \$var *= e1, \$var /= e1</code>	Equivalentes a <code>\$var = \$var + e1, \$var = \$var - e1...</code> Válido para cualquier operador binario aritmético, de cadenas o arrays.

### Otros operadores

<code>\$var++, \$var--</code>	Devuelve \$var, luego le suma (resta) 1.
<code>++\$var, --\$var</code>	Suma (resta) 1 a \$var, devuelve el valor actualizado.
<code>\$cad1. \$cad2</code>	Concatena dos cadenas.

## ~~2.7. Arrays~~

Los *arrays* en PHP son una estructura muy flexible y potente. Unifica en un solo tipo lo que en otros lenguajes se consigue con *arrays* básicos, vectores, listas o diccionarios. Los elementos de un *array* se identifican por una clave, que puede ser un entero o una cadena. Los elementos guardan un orden dentro del *array*. Este orden está determinado por el orden de los elementos al declarar el *array* o al añadir nuevos.

Para declarar un *array* se puede utilizar la función `array()`.

```
$var = array(
    clave1 => valor1,
    ...
    claven => valorn
);
```

También es posible utilizar la notación de corchetes

```
$arr = [
    clave1 => valor1,
    ...
    claven => valorn
];
```

Se puede acceder a cada elemento del *array* por su clave utilizando los corchetes, como es habitual en otros lenguajes:

```
$arr[clave] = valor;
```

El ejemplo **arrays1.php** muestra cómo declarar y utilizar los *arrays*. En el ejemplo se utiliza la función `print_r()`, que muestra información sobre la variable que se le pase como argumento. Si se le pasa un *array*, muestra las claves y valores de todos los elementos.

```
<?php
    $arr1 = [
        0 => 444,
        1 => 222,
        2 => 333,
    ];
    print_r($arr1);
    echo "<br>". "pos 0: ". $arr1[0]. "<br>";
    $arr1[0] = 555;
    print_r($arr1);
    echo "<br>";
    $arr2 = array (
        "1111A" => "Juan Vera Ochoa",
        "1112A" => "Maria Mesa Cabeza",
        "1113A" => "Ana Puertas Peral"
    );
    $arr2["1113A"] = "Ana Puertas Segundo";
```

Para recorrer un *array* lo habitual es utilizar el bucle `foreach`.

```
foreach($arr as $valor) {
    ...
}
```

El cuerpo del bucle se ejecuta una vez por cada elemento del *array* `$arr`. En cada iteración del bucle `$valor` almacena el elemento correspondiente.

También se puede utilizar especificando una variable para la clave,

```
foreach($arr as $clave => $valor){ ... }
```

En este caso, en cada iteración también se dispondrá de la clave del elemento correspondiente en la variable `$clave`.

```
<?php
    $arr2 = array (
        "1111A" => "Juan Vera Ochoa",
        "1112A" => "Maria Mesa Cabeza",
        "1113A" => "Ana Puertas Peral"
    );
    foreach ($arr2 as $nombre) {
        echo "$nombre <br>";
    }
    foreach ($arr2 as $codigo => $nombre) {
        echo "Código: $codigo Nombre: $nombre <br>";
    }
}
```

Si se pretende modificar el contenido del *array* al recorrerlo con un bucle `foreach`, hay que utilizar una referencia al declarar las variables del bucle. En el ejemplo **foreach\_modificar.php**, el primer bucle no utiliza referencias y, por tanto, no modifica el *array*. El segundo bucle es la forma correcta de hacerlo.

```
<?php
    $arr1 = array(
        "Viernes" => 22,
        "Sábado" => 34
    );
    /* no modifica el array */
    foreach ($arr1 as $cantidad) {
        $cantidad = $cantidad * 2;
    }
    print_r($arr1);
    echo "<br>";
    /* modifica el array */
    foreach ($arr1 as &$cantidad) {
        $cantidad = $cantidad * 2;
    }
    print_r($arr1);
```

La salida del programa será:

```
Array ( [Viernes] => 22 [Sábado] => 34 )
Array ( [Viernes] => 44 [Sábado] => 68 )
```

También es posible declarar un *array* sin usar claves. En ese caso, se asignan como claves por defecto enteros consecutivos empezando por 0. Si se añade un nuevo elemento sin especificar clave, la clave será el número siguiente a la mayor clave entera presente en el *array*.

```
<?php
    $arr1 = array(10, 20, 30, 40);
    print_r($arr1);
    echo "<br>";
    $arr1[] = 5;
    print_r($arr1);
    echo "<br>";
    $arr1[10] = 6;
    $arr1[] = 5;
    print_r($arr1);
    echo "<br>";
```

Salida:

```
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 )
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 [4] => 5 )
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 [4] => 5 [10] => 6 [11] => 5 [12] => 5 )
```

Algunos operadores también están disponibles para los *arrays*, pero su significado no es exactamente el mismo. Por ejemplo, el operador “Idéntico” solo será verdadero si los dos *arrays* tienen todos los elementos iguales tanto en clave como en valor y además están en el mismo orden. Para el operador “Igual”, el orden de las claves no importa.

```
<?php
    $arr1 = array(
        1 => "3000",
        2 => "4000",
    );
    $arr2 = array(
        1 => 3000,
        2 => 4000,
    );
    $arr3 = array(
        2 => "4000",
        1 => "3000",
    );
    if($arr1 == $arr2){
        echo "arr1 y arr2 son iguales <br>";
    }else{
        echo "arr1 y arr2 no son iguales <br>";
    }
    if($arr1 == $arr3){
        echo "arr1 y arr3 son iguales <br>";
    }else{
        echo "arr1 y arr3 no son iguales <br>";
    }
    if($arr1 === $arr2){
        echo "arr1 y arr2 son idénticos <br>";
    }else{
        echo "arr1 y arr2 no son idénticos <br>";
    }
    if($arr1 === $arr3){
        echo "arr1 y arr3 son idénticos <br>";
    }else{
        echo "arr1 y arr3 no son idénticos <br>";
    }
}
```

La salida será:

```
arr1 y arr2 son iguales
arr1 y arr3 son iguales
arr1 y arr2 no son idénticos
arr1 y arr3 son idénticos
```

El operador “+” utilizado con dos *arrays* devuelve su unión. El *array* resultante contendrá primero los elementos del primer *array* (el que aparece a la izquierda del operador) y a continuación los del segundo, pero sin repetir claves. Si hay elementos con la misma clave en los dos *arrays*, el del segundo no se añade al resultado.

```
<?php
    $arr1 = array(
        10 => "3000",
        20 => "4000",
        30 => "6000",
    );
    print_r($arr1);
    echo "<br>";
    $arr2 = array(
        10 => "8000",
        15 => "6000",
        20 => "4000",
    );
    print_r($arr2);
    echo "<br>";
    $arr3 = $arr1 + $arr2;
    print_r($arr3);
```

El resultado será:

```
Array ( [10] => 3000 [20] => 4000 [30] => 6000 )
Array ( [10] => 8000 [15] => 6000 [20] => 4000 )
Array ( [10] => 3000 [20] => 4000 [30] => 6000 [15] => 6000 )
```

#### CUADRO 2.5 Operadores para arrays

Operador	Descripción
<code>\$a1 === \$a2</code>	Idéntico. Verdadero si los dos <i>arrays</i> tienen las claves y valores iguales, en el mismo orden y del mismo tipo.
<code>\$a1 !== \$a2</code>	No idéntico.
<code>\$a1 == \$a2</code>	Igual. Verdadero si los dos <i>arrays</i> tienen las claves y valores iguales.
<code>\$a1 != \$a2, \$a1 &lt;&gt; \$a2</code>	No igual.
<code>\$a1 + \$a2</code>	Unión. Devuelve un <i>array</i> con los elementos de ambos.

## 2.8. Funciones y librerías

Una función es un conjunto de instrucciones que realiza una tarea concreta. Las funciones pueden recibir argumentos, los datos que necesitan para llevar a cabo su cometido. Por ejemplo, una función que abre un fichero puede recibir como argumento el nombre del fichero que tiene que abrir.

Las funciones pueden devolver o no un valor. A diferencia de lo que ocurre en otros lenguajes, en PHP no es necesario declarar el tipo de dato que devuelve una función y, de hecho, puede devolver tipos de datos diferentes según el caso. Por ejemplo, en PHP hay muchas funciones que devuelven FALSE en caso de error y no devuelven un *boolean* si no hay error.

### 2.8.1. Funciones predefinidas

PHP cuenta con una gran cantidad de funciones predefinidas para las tareas más habituales.

Entre las más utilizadas están las relacionadas con las variables. Como en PHP las variables no se declaran explícitamente, hay una serie de funciones que permiten conocer el estado y el tipo de dato de una variable. Las más importantes son:

- `is_null($var)`. Devuelve TRUE si \$var es NULL, FALSE en otro caso.
- `isset($var)`. Devuelve TRUE si \$var ha sido inicializada y su valor no es NULL, FALSE en otro caso.
- `unset($var)`. Elimina la variable. Ya no contará como inicializada.
- `empty($var)`. Devuelve TRUE si \$var no ha sido inicializada o su valor es FALSE, FALSE en otro caso.
- `is_int($var), is_float($var), is_bool($var), is_array($var)`. Devuelven TRUE si \$var es entero, float, booleano o array respectivamente, y FALSE en otro caso.
- `print_r($var)` y `var_dump($var)`. Muestran información sobre \$var.

En el ejemplo **funciones\_variables.php** se pueden ver las diferencias entre estas funciones.

```
<?php
    $var1 = 4;
    $var2 = NULL;
    $var3 = FALSE;
    $var4 = 0;
    echo "var 1";
    var_dump(isset($var1)); // TRUE
    var_dump(is_null($var1)); // FALSE
    var_dump(empty($var1)); // FALSE
    echo "var 2";
    var_dump(isset($var2)); // FALSE
    var_dump(is_null($var2)); // TRUE
    var_dump(empty($var2)); // TRUE
    echo "var 3";
    var_dump(isset($var3)); // TRUE
    var_dump(is_null($var3)); // FALSE
    var_dump(empty($var3)); // TRUE
    echo "var 4";
```

```

var_dump(empty($var4)); // TRUE, EL 0 COMO BOOLEAN ES FALSE
echo "unset";
unset($var1);
var_dump(isset($var1)); // FALSE

```

A lo largo de los siguientes capítulos se irán presentando más funciones según sean necesarias. En el cuadro 2.6 se resumen algunas de las más útiles.

### **CUADRO 2.6** Algunas funciones útiles en PHP

<b>Funciones de variables</b>	
isset(\$var)	TRUE si la variable está inicializada y no es NULL
is_null(\$var)	TRUE si la variable es NULL
empty(\$var)	TRUE si la variable no está inicializada o su valor es FALSE
is_int(\$var), is_float(\$var), is_bool(\$var), is_array(\$var)	Para comprobar el tipo de dato de \$var
intval(\$var), floatval(\$var), boolval(\$var), strval(\$var)	Para obtener el valor de \$var como otro tipo de dato
<b>Funciones de cadenas</b>	
strlen(\$cad)	Devuelve la longitud de \$cad
explode(\$cad, \$token)	Parte una cadena utilizando \$token como separador. Devuelve un array de cadenas
implode(\$token, \$array)	Crea una cadena larga a partir de un array de cadenas, entre cadena y cadena se introduce \$token
strcmp(\$cad1, \$cad2)	Compara las dos cadenas. Devuelve 0 si son iguales, -1 si \$cad1 es menor y 1 si \$cad1 es mayor
strtolower(\$cad), strtoupper(\$cad)	Devuelven \$cad en mayúsculas o minúsculas, respectivamente
str(\$cad1, \$cad2)	Busca la primera ocurrencia de \$cad2 en \$cad1. Si no aparece devuelve FALSE, si aparece devuelve \$cad1 desde donde comienza la ocurrencia
<b>Funciones de arrays</b>	
ksort(\$arr), krsort(\$arr)	Ordena el array por clave en orden ascendente o descendente
sort(\$arr), rsort(\$arr)	Ordena el array por valor en orden ascendente o descendente
array_values(\$arr)	Devuelve los valores de \$arr
array_keys(\$arr)	Devuelve las claves de \$arr
array_key_exists(\$arr, \$cla)	Devuelve verdadero si algún elemento de \$arr tiene clave \$cla
count(\$arr)	Devuelve el número de elementos del array

## 2.8.2. Funciones definidas por el usuario

~~X~~ La sintaxis para definir una función es:

```
function nombre(argumentos){  
    <instrucciones>  
}
```

La función puede tener o no argumentos. Se declaran en la cabecera de la función, entre los paréntesis, separados por comas. Los argumentos son variables locales a la función y solo existen mientras se esté ejecutando. En PHP no hace falta declarar el tipo de dato que devuelve la función. De hecho, es posible que una función devuelva tipos de datos diferentes según el caso.

Por ejemplo, la siguiente función recibe dos números y devuelve su suma.

```
<?php  
    function suma($a, $b) {  
        return $a + $b;  
    }  
    echo suma(4,8).'  
';  
    $var1 = 35;  
    $var2 = 5;  
    $var3 = suma($var1, $var2);  
    echo $var3.'  
';
```

Es posible especificar valores por defecto para los argumentos. Si al llamar a la función no se usa el argumento, se toma el valor por defecto.

```
<?php  
    function saludar($nombre = 'usuario'){  
        echo "Hola $nombre  
";  
    }  
    saludar();  
    saludar("Ana");
```

La salida será:

Hola usuario  
Hola Ana

### Actividad propuesta 2.2



Escribe una función para calcular potencias. Recibirá como argumentos la base y el exponente, que es opcional y tiene valor por defecto 2 (elevar al cuadrado).

### 2.8.3. Paso de argumentos por copia y por valor

En PHP los argumentos se pasan por copia. Esto quiere decir que, cuando se llama a una función con una variable como argumento, se crea una variable local a la función en la que se copia el valor del argumento. Por tanto, si la función modifica el argumento, estos cambios no tienen efecto en la variable original.

Esto es lo que ocurre en la función `duplicarMal()` del siguiente ejemplo. El argumento se multiplica por dos, pero el valor de `$var1` no cambia. Para solucionarlo hay dos opciones. Devolver el nuevo valor y reasignar, como se hace con la función `duplicar()`, o utilizar una referencia, como se hace en `duplicar2()`. Para utilizar una referencia solo hace falta añadir el símbolo “&” antes del argumento.

```
<?php
    function duplicarMal($a){
        $a = $a *2;
    }
    function duplicar($a){
        return $a *2;
    }
    function duplicar2(&$a){
        $a = $a *2;
    }
    $var1 = 5;
    duplicarMal($var1);
    echo "$var1 <br>";
    $var1 = duplicar($var1);
    echo "$var1 <br>";
    duplicar2($var1);
    echo "$var1 <br>";
```

La salida del programa será:

5  
10  
20

### 2.8.4. Funciones como argumentos

En PHP es posible pasar funciones como argumentos a otras funciones. Es una característica avanzada que se utiliza entre otras cosas para las funciones de *callback*. Solo hay que pasar el nombre de la función entre comillas como argumento. La función que lo recibe podrá utilizarla si le pasan los argumentos adecuados.

En el ejemplo **calculador.php** se utiliza esta característica. La función `calculador()` recibe como argumentos dos números y también la función que debe aplicarles. Según qué función se le pase como argumento, devolverá un valor u otro.

```
<?php
    function calculador($operacion, $numa, $numb){
        $resul = $operacion($numa, $numb);
```

```

        return $resul;
    }
    function sumar($a, $b){
        return $a + $b;
    }
    function multiplicar($a, $b){
        return $a * $b;
    }
    $a = 4;
    $b = 5;
    $r1 = calculador("multiplicar", $a, $b);
    echo "$r1 <br>";
    $r2 = calculador("sumar", $a, $b);
    echo "$r2 <br>";

```

### Actividad propuesta 2.3



Escribe una función para calcular el factorial de un número, que recibirá como argumento. Devolverá el factorial o -1 si el argumento no es válido.

## 2.9. Excepciones y errores

El sistema de control de errores de PHP ha ido evolucionando a lo largo de las versiones. En el sistema básico, se generan errores de diferentes tipos, representados por un número. Por otro lado, desde PHP 5 hay un sistema de excepciones similar al de Java y otros lenguajes utilizando bloques `try/catch/finally`. Finalmente, en PHP 7 aparecieron las excepciones de clase Error.

### 2.9.1. Errores

En el sistema básico, ante determinadas condiciones (por ejemplo, utilizar una variable no inicializada) PHP genera un error. Hay diferentes tipos de errores, cada uno asociado con un número y una constante predefinida. Se puede controlar cómo se comporta PHP antes los errores mediante tres directivas del fichero `php.ini`:

- `error_reporting`: indica qué errores deben reportarse. Lo normal es utilizar `E_ALL`, es decir, todos.
- `display_errors`: señala si los mensajes de error deben aparecer en la salida del `script`. Esta opción es apropiada durante el desarrollo, pero no en producción.
- `log_errors`: indica si los mensajes de error deben almacenarse en un fichero. Es especialmente útil en producción, cuando no se muestran los errores en la salida.
- `error_log`: si la directiva anterior está activada, es la ruta en la que se guardan los mensajes de error.

El valor de la directiva *error\_reporting* es un número, pero para especificarlo lo habitual es utilizar las constantes predefinidas y el operador *or* a nivel de bit.

**CUADRO 2.7**  
**Tipos de error**

Código	Constante	Descripción
1	E_ERROR	Error fatal en tiempo de ejecución. La ejecución del <i>script</i> se detiene
2	E_WARNING	Advertencia en tiempo de ejecución. El <i>script</i> no se detiene
4	E_PARSE	Error de sintaxis al compilar
8	E_NOTICE	Notificación. Puede indicar error o no
16	E_CORE_ERROR	Error fatal al iniciar PHP
32	E_CORE_WARNING	Advertencia al iniciar PHP
64	E_COMPILE_ERROR	Error fatal al compilar
128	E_COMPILE_WARNING	Advertencia fatal al compilar
256	E_USER_ERROR	Error generado por el usuario
512	E_USER_WARNING	Advertencia generada por el usuario
1024	E_USER_NOTICE	Notificación generada por el usuario
2048	E_STRICT	Sugerencias para mejorar la portabilidad
4096	E_RECOVERABLE_ERROR	Error fatal capturable
8192	E_DEPRECATED	Advertencia de código obsoleto
16384	E_USER_DEPRECATED	Como la anterior, generada por el usuario
32767	E_ALL	Todos los errores



#### Actividad propuesta 2.4

Modifica el fichero *php.ini* para que los errores *E\_NOTICE* no se muestren por pantalla (*display\_errors*).

Comprueba la diferencia accediendo a *localhost/cap2/no\_init.php*.

- **Funciones relacionadas**

La función *error\_reporting()* permite cambiar el valor de la directiva *error\_reporting* en tiempo de ejecución.

También es posible definir una función propia para que se encargue de los errores utilizando `set_error_handler()`. La función que se ocupe de los errores tendrá que tener la siguiente firma:

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

El siguiente ejemplo muestra cómo utilizar `set_error_handler()` para manejar los errores con una función propia.

```
<?php
    function manejadorErrores($errno, $str, $file, $line){
        echo "Ocurrió el error: $errno";
    }
    set_error_handler("manejadorErrores");
    $a = $b; // causa error, $b no está inicializada
```

La salida será:

Ocurrió el error: 8

## 2.9.2. Excepciones

Otra opción para indicar un error es lanzar una excepción. Para controlar las excepciones se utilizan bloques `try/catch/finally`, como en Java. Cuando se lanza una excepción y no es capturada por un bloque `catch`, la ejecución del programa se detiene. Si es capturada, se ejecuta el código del bloque correspondiente.

Para capturar una excepción se introduce la instrucción que puede causarla dentro de un bloque `try` y se añade el bloque `catch` correspondiente. Se puede añadir un bloque `finally`, que se ejecuta después del `try/catch`, haya habido excepción o no.

```
try{
    instrucciones;
}catch(Exception e){
    instrucciones;
}finally{
    instrucciones;
}
```

En el ejemplo **excepciones.php** se utiliza una función que recibe dos argumentos y lanza una excepción si el segundo es cero. Para lanzar una excepción se utiliza `throw`, que recibe como argumento un objeto de clase `Exception` o de alguna subclase.

```
1  <?php
2      function dividir($a, $b){
3          if ($b==0){
4              throw new Exception('El segundo argumento es 0');
5      }
```

```

6         return $a/$b;
7     }
8     try{
9         $resul1 = dividir(5, 0);
10        echo "Resul 1 $resul1". "<br>";
11    }catch(Exception e){
12        echo "Excepción: ". $e->getMessage(). "<br>";
13    }finally{
14        echo "Primer finally";
15    }
16    try{
17        $resul2 = dividir(5, 2);
18        echo "Resul 2 $resul2". "<br>";
19    }catch(Exception e){
20        echo "Excepción: ". $e->getMessage(). "<br>";
21    }finally{
22        echo "Segundo finally";
23    }

```

En la primera llamada a `dividir()`, línea 9, se produce una excepción. Por tanto, el resto del bloque `try`, el `echo`, no se ejecuta. Sí se ejecutan el `catch` y el `finally` correspondientes.

En la segunda llamada a `dividir()`, línea 17, no se produce excepción. En este caso se ejecutan el `echo` del `try` y a continuación el del bloque `finally`. El bloque `catch` no se ejecuta.



### Actividad propuesta 2.5

Adapta la actividad 2.3 para que controle si el argumento es negativo utilizando una excepción.

### 2.9.3. Excepciones Error

En PHP 7 aparecieron las excepciones de tipo Error. No heredan de la clase `Exception`, así que para capturarlas hay que usar:

```

catch (Error $e){
    ...
}

```

o, alternativamente, la clase `Throwable`, de la que se derivan tanto `Error` como `Exception`:

```

catch (Throwable $e){
    ...
}

```

En el cuadro 2.8. se muestran las excepciones Error predefinidas.

#### CUADRO 2.8 Excepciones Error

Nombre	Descripción	Hereda de
Error	Clase base para las excepciones Error	
ArithmetricError	Error en operaciones matemáticas. Hereda del anterior	Error
DivisionByZeroError	Intento de división por cero. Hereda del anterior	ArithmetricError
AssertionError	Ocurre cuando falla una llamada a assert()	Error
ParseError	Error al compilar	Error
TypeError	Ocurre cuando una expresión no tiene el tipo de dato que se espera	Error
ArgumentCountError	Ocurre al llamar a una función con menos argumentos de los necesarios. Hereda del anterior	TypeError

## 2.10. Clases y objetos

PHP tiene soporte completo para la programación orientada a objetos. Permite definir clases, herencia, interfaces y los demás elementos habituales. Para declarar una clase se utiliza la palabra reservada **class**. Los atributos se declaran utilizando su nombre, los modificadores que pueda tener y opcionalmente un valor por defecto.

```
class Clase{
    private $att1 = 10; // con valor por defecto
    private $atr2; // sin valor por defecto
    private static $atr3 = 0; // estático
    ...
}
```

Los atributos y métodos se pueden declarar como estáticos, de manera que no habrá uno por objeto, sino uno por clase. Se utiliza la palabra reservada **static**.

Se puede crear un constructor para la clase con el método **\_\_construct()**. Este método forma parte de los *métodos mágicos*, nombres reservados a tareas concretas y que comienzan por dos guiones bajos, “**\_\_**”.

Para crear un nuevo objeto se utiliza el operador **new** seguido del nombre de la clase:

```
$obj = new Clase();
```

Para los atributos y métodos se pueden utilizar los siguientes modificadores de visibilidad:

- **public**. Se pueden utilizar desde dentro y fuera de la clase.
- **private**. Pueden emplearse desde la propia clase.
- **protected**. Se pueden utilizar dentro de la propia clase, las derivadas y las antecesoras.

Normalmente los atributos se declaran como privados y se crean métodos públicos para acceder a ellos. Para acceder a los métodos y atributos se utiliza:

```
$objeto->propiedad;
$objeto->método(argumentos);
```

Los métodos son funciones definidas dentro de una clase. En los métodos no estáticos se puede utilizar la palabra reservada `this`, que representa el objeto desde el que se invoca el método.

El ejemplo **PersonayCliente.php** muestra cómo crear una clase y crear y manejar objetos de esta. Para empezar, declara la clase Persona, con atributos para nombre, apellido y DNI. Los tres son privados. La clase tiene un constructor, el método `__construct()`, que recibe valores para los tres atributos. A continuación, se declaran una serie de métodos públicos para leer y escribir los atributos, los llamados *getters* y *setters* (líneas 11-23).

```
1  <?php
2  class Persona {
3      private $DNI;
4      private $nombre;
5      private $apellido;
6      function __construct($DNI, $nombre, $apellido) {
7          $this->DNI = $DNI;
8          $this->nombre = $nombre;
9          $this->apellido = $apellido;
10     }
11     public function getNombre() {
12         return $this->nombre;
13     }
14     public function getApellido() {
15         return $this->apellido;
16     }
17     public function setNombre($nombre) {
18         $this->nombre = $nombre;
19     }
20
21     public function setApellido($apellido) {
22         $this->apellido = $apellido;
23     }
24     public function __toString() {
25         return "Persona: ".$this->nombre." ".$this->apellido;
26     }
27 }
```

En esta clase también se utiliza `__toString()` (líneas 24-26), otro método mágico que se usa para generar una cadena de texto con la información del objeto. Cuando se pasa un objeto a `echo`, se representa usando el valor de retorno de `__toString()`.

En las líneas 46-52 se crea un objeto de la clase Persona y se manipula.

```
45 // crear una persona
46 $per = new Persona("1111111A", "Ana", "Puertas");
47 // mostrarla, usa el método __toString()
48 echo $per. "<br>";
49 // cambiar el apellido
50 $per->setApellido("Montes");
```

```
51 // volver a mostrar
52 echo $per. "<br>";
```

La salida de este fragmento será:

Persona: Ana Puertas  
Persona: Ana Montes

Para crear una clase que herede de otra, se utiliza la palabra clave `extends`. La clase derivada tendrá los mismos atributos y métodos que la clase base y podrá añadir nuevos o sobrescribirlos.

En el ejemplo anterior se declara también la clase Cliente, que hereda de persona y añade un atributo saldo. Incluye también los métodos `getSaldo()` y `setSaldo()` y sobrescribe el método `_toString()` de la clase base. También hay un constructor que incluye el nuevo atributo y utiliza el constructor de la clase base.

```
class Cliente extends Persona{
    private $saldo = 0;
    function __construct($DNI, $nombre, $apellido, $saldo){
        parent::__construct($DNI, $nombre, $apellido);
        $this->$saldo = $saldo;
    }
    public function getSaldo(){
        return $this->saldo;
    }
    public function setSaldo($saldo){
        $this->saldo = $saldo;
    }
    public function __toString(){
        return "Cliente: ". $this->getNombre();
    }
}
```

En las líneas 54-56 se crea un objeto de la clase cliente y se muestra. La salida será “Cliente: Pedro”.

```
$cli = new Cliente("22222245A", "Pedro", "Sales", 100);
// lo muestra
echo $cli. "<br>";
```

## Resumen

- PHP es el lenguaje más extendido para el desarrollo web en el lado del servidor.
- Los ficheros de PHP mezclan HTML y PHP. El servidor sustituye los bloques de PHP por su salida.
- En PHP no hace falta declarar las variables, se declaran la primera vez que se usan.
- Tampoco es necesario definir su tipo de dato, se infiere del valor de inicialización.

- El paso de parámetros y la asignación se realizan por defecto por copia, pero se pueden usar referencias.
- Las funciones de PHP pueden ser parámetros de otras funciones. Es una característica muy usada en las librerías PHP.
- PHP cuenta con muchas funciones integradas para las tareas más habituales: manipulación de cadenas, arrays, bases de datos, ficheros...
- Los arrays de PHP son un tipo de dato muy potente similar a los que en otros lenguajes se llaman *mapas* o *diccionarios*.
- Hay tres sistemas de control de errores en PHP: errores, excepciones y excepciones de clase Error.
- PHP tiene soporte completo para la programación orientada a objetos.

## Ejercicios propuestos



1. Escribe un *script* para resolver ecuaciones de segundo grado,  $ax^2 + bx + c = 0$ . Si la ecuación no tiene soluciones reales, hay que mostrar un mensaje de error.
2. Crea una función para resolver la ecuación de segundo grado. Esta función recibe los coeficientes de la ecuación y devuelve un array con las soluciones. Si no hay soluciones reales, devuelve FALSE.
3. Almacena la función anterior en el fichero **matemáticas.php**. Crea un fichero que la incluya y la utilice.
4. Escribe una función que reciba una cadena y comprueba si es un palíndromo.
5. Escribe una función que reciba un array de números, y un número, el límite. La función tiene que devolver un nuevo array que contenga solo los elementos del array original menores que el límite.
6. Escribe un *script* para probar las funciones del cuadro 2.6.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. En PHP, las variables:
  - a) No tienen tipo de dato.
  - b) Tienen un tipo de dato que puede cambiar.
  - c) Tienen un tipo de dato que no puede cambiar.
2. En PHP, las funciones:
  - a) Tienen que declarar el tipo de dato que devuelven.
  - b) Pueden devolver diferentes tipos de datos según el caso.
  - c) No tienen que declarar el tipo de dato que devuelven, pero siempre tienen que devolver el mismo.

3. En los *arrays* de PHP, si no se asigna clave al insertar un elemento:
- a) Da error.
  - b) Se inserta sin clave.
  - c) Se le asigna una clave numérica.
4. El orden de los elementos dentro de un *array* está determinado por:
- a) Su clave.
  - b) Su valor.
  - c) El orden de inserción.
5. Al utilizar una variable no inicializada:
- a) Se genera un error de tipo E\_NOTICE y la ejecución del script se detiene.
  - b) Se genera un error de tipo E\_NOTICE y la ejecución del script continúa.
  - c) No se genera ningún error.
6. La salida de estas líneas de código es:
- ```
$arr = array(2,2,2,2);
foreach($arr as $elemento){
    $elemento = $elemento *2;
}
foreach($arr as $elemento){
    echo $elemento. " ";
}
```
- a) 2 2 2 2.
  - b) 4 4 4 4.
  - c) Hay un error.
7. La salida de estas líneas de código es:
- ```
$a = 4;
$b = &$a;
$a = 6;
echo $b;
```
- a) 4
  - b) 6
  - c) 8.
8. En PHP, los argumentos:
- a) Se pasan por copia.
  - b) Se pasan por valor.
  - c) Se pueden pasar por copia o por valor.
9. Sobre los bloques de PHP:
- a) Se deben cerrar siempre.
  - b) Si el fichero acaba con un bloque, no se cierra.
  - c) Si el fichero contiene solo bloques de PHP, el último no se cierra.

10. Dos arrays son idénticos si:

- a) Todos los elementos tienen el mismo valor y están en el mismo orden.
- b) Todos los elementos son iguales en clave y valor y además están en el mismo orden.
- c) Todos los elementos son iguales en clave y valor.

#### SOLUCIONES:

1. **[a] [b] [c]**

2. **[a] [b] [c]**

3. **[a] [b] [c]**

4. **[a] [b] [c]**

5. **[a] [b] [c]**

6. **[a] [b] [c]**

7. **[a] [b] [c]**

8. **[a] [b] [c]**

9. **[a] [b] [c]**

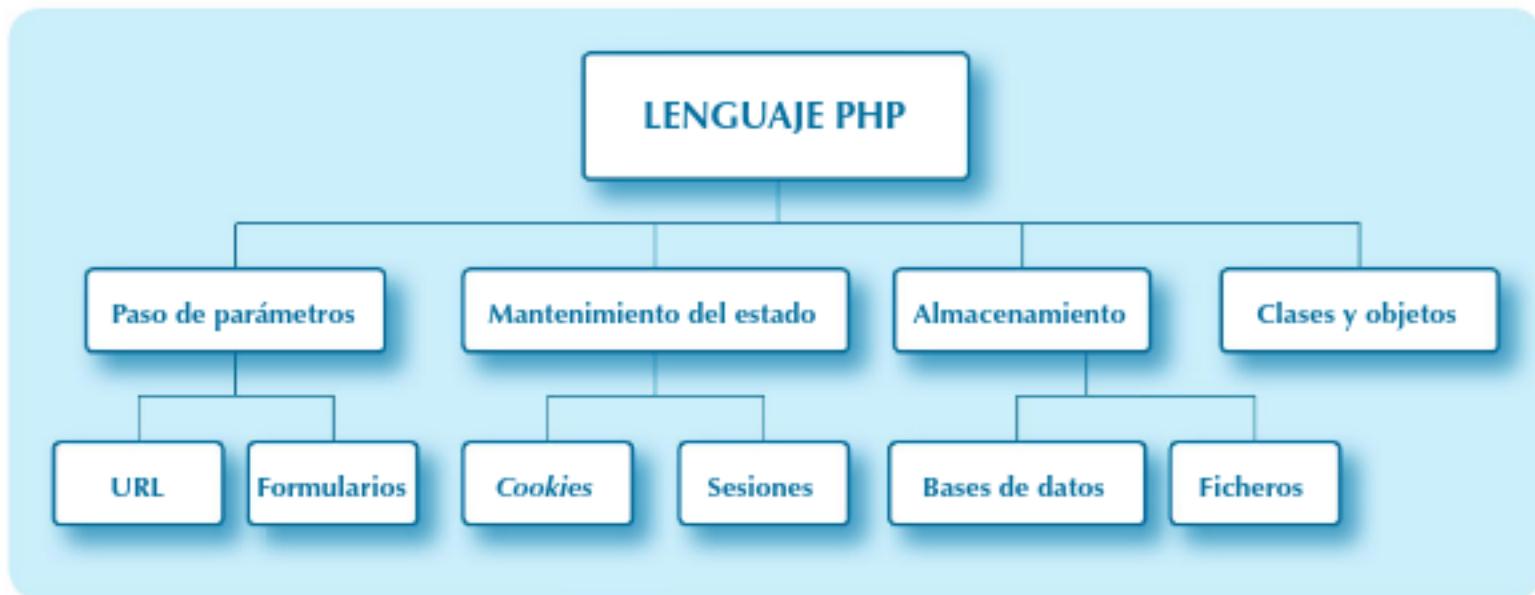
10. **[a] [b] [c]**

# Desarrollo de aplicaciones web con PHP

## Objetivos

- ✓ Conocer los mecanismos de paso de parámetros a un *script*.
- ✓ Procesar y validar formularios.
- ✓ Utilizar *cookies*.
- ✓ Aprender a emplear las sesiones y las variables de sesión.
- ✓ Manejar bases de datos.
- ✓ Enviar correos desde PHP.
- ✓ Dominar las herramientas de pruebas y depuración.

## Mapa conceptual



## Glosario

**Base de datos no relacional.** Sistema gestor de bases de datos que no sigue el modelo relacional. Por ejemplo, bases de datos XML.

**Cookie.** Fichero que almacenan los servidores en los clientes. Puede almacenar información sobre la última visita o las preferencias del usuario, entre otras cosas.

**Formulario.** Elemento que permite al usuario introducir datos que se envían servidor.

**Sesión.** Las sesiones permiten que las páginas de una misma aplicación comparten información.

**XML.** eXtensible Markup Language, “lenguaje de marcas extensibles”. Es un estándar del W3C.

**XPath.** El lenguaje XPath se utiliza para buscar información en ficheros XML. Es un estándar del W3C.

**XSD.** Lenguaje basado en XML para validar documentos XML. Validar consiste en verificar que un fichero tiene una estructura determinada. Es un estándar del W3C.

**XSL/XSLT.** Lenguaje basado en XML para realizar transformaciones sobre documentos XML. El resultado puede ser XML, HTML... Es un estándar del W3C.

### 3.1. Paso de parámetros

El protocolo HTTP define ocho métodos o verbos para establecer una comunicación entre cliente y servidor. Los más habituales son GET (obtener) y POST (enviar). El método GET es el que se utiliza habitualmente para solicitar páginas web a un servidor, por ejemplo, al seguir

un vínculo o introducir una dirección a mano en la barra del navegador. El método POST se utiliza sobre todo para enviar formularios al servidor.

Cuando se usa el método GET se pueden pasar parámetros al servidor en la URL. A la ruta normal para acceder a una página se le añade el carácter "?" como indicador de que empieza la lista de parámetros. Cada parámetro tiene un nombre, a la izquierda del igual, y un valor, a la derecha. Los argumentos están separados entre sí por el carácter *ampersand*. Por ejemplo, si se accede con el navegador a:

`http://localhost/cap3/hola_nombre.php?nombre=Ana`

Se solicita al servidor el fichero **/cap3/hola\_nombre.php** del servidor *localhost* y se le pasa un parámetro llamado "nombre" con valor "Ana".

Con la siguiente URL se añadiría otro parámetro para el apellido:

`http://localhost/cap3/hola_nombre.php?nombre=Ana&apellido=Luna`

Se puede acceder a los argumentos de la URL desde un bloque PHP usando el *array* superglobal `$_GET`, que tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del *array*.

El fichero **hola\_nombre.php** muestra un mensaje personalizado usando el valor del parámetro nombre.

```
<?php  
echo "Hola ". $_GET["nombre"];
```

Si se accede con la ruta

`http://localhost/cap3/hola_nombre.php?nombre=Ana,`

se obtendrá el mensaje "Hola Ana".

Si se accede con

`http://localhost/cap3/hola_nombre.php,`

se obtendrá:

`Notice: Undefined index: nombre in C:\xampp\htdocs\cap3\eje3_1.php on line 2 Hola`

Para controlar si los parámetros se han pasado correctamente se pueden utilizar las funciones `empty()` o `is_null()`. Las dos devuelven TRUE cuando el parámetro no está presente en la URL. La diferencia está en los parámetros presentes, pero sin valor, por ejemplo:

`http://localhost/cap3/hola_nombre.php?nombre`

En este caso, `empty($_GET["nombre"])` devuelve TRUE, pero `is_null($_GET["nombre"])` devuelve FALSE.

El ejemplo **hola\_comprobacion.php** mejora el anterior para mostrar un mensaje de error si no se pasa el parámetro nombre.

```
<?php
    if (empty($_GET["nombre"])) {
        echo "Error, falta el parámetro nombre";
    }else {
        echo "Hola ". $_GET["nombre"];
    }
}
```

### Actividad propuesta 3.1



Escribe un fichero que reciba dos parámetros, num1 y num2, y muestre su suma. Hay que comprobar que los dos argumentos existan y sean números.

## 3.2. Formularios

Los formularios HTML son la forma más habitual de enviar datos a un servidor. Permiten que el usuario rellene varios campos mediante diferentes tipos de controles (campos de texto, botones de radio...) y lo envíe al servidor al pulsar un botón. El servidor procesa los datos del formulario y genera la respuesta.

Un formulario sencillo de *login* en HTML se escribiría así:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <form action = "login_basico.php" method = "POST">
            <input name = "usuario" type = "text">
            <input name = "clave" type = "password">
            <input type = "submit">
        </form>
    </body>
</html>
```

El atributo *action* del formulario especifica la ruta del *script* al que se enviará el formulario para que lo procese. Si se usa una ruta relativa, se toma como referencia la localización en el servidor del fichero que contenga el formulario.

El atributo *method* especifica el método HTPP, que se usará para la petición. En los formularios lo habitual es utilizar POST, pero también es posible utilizar GET. Si se utiliza POST, los parámetros no aparecen en la URL.

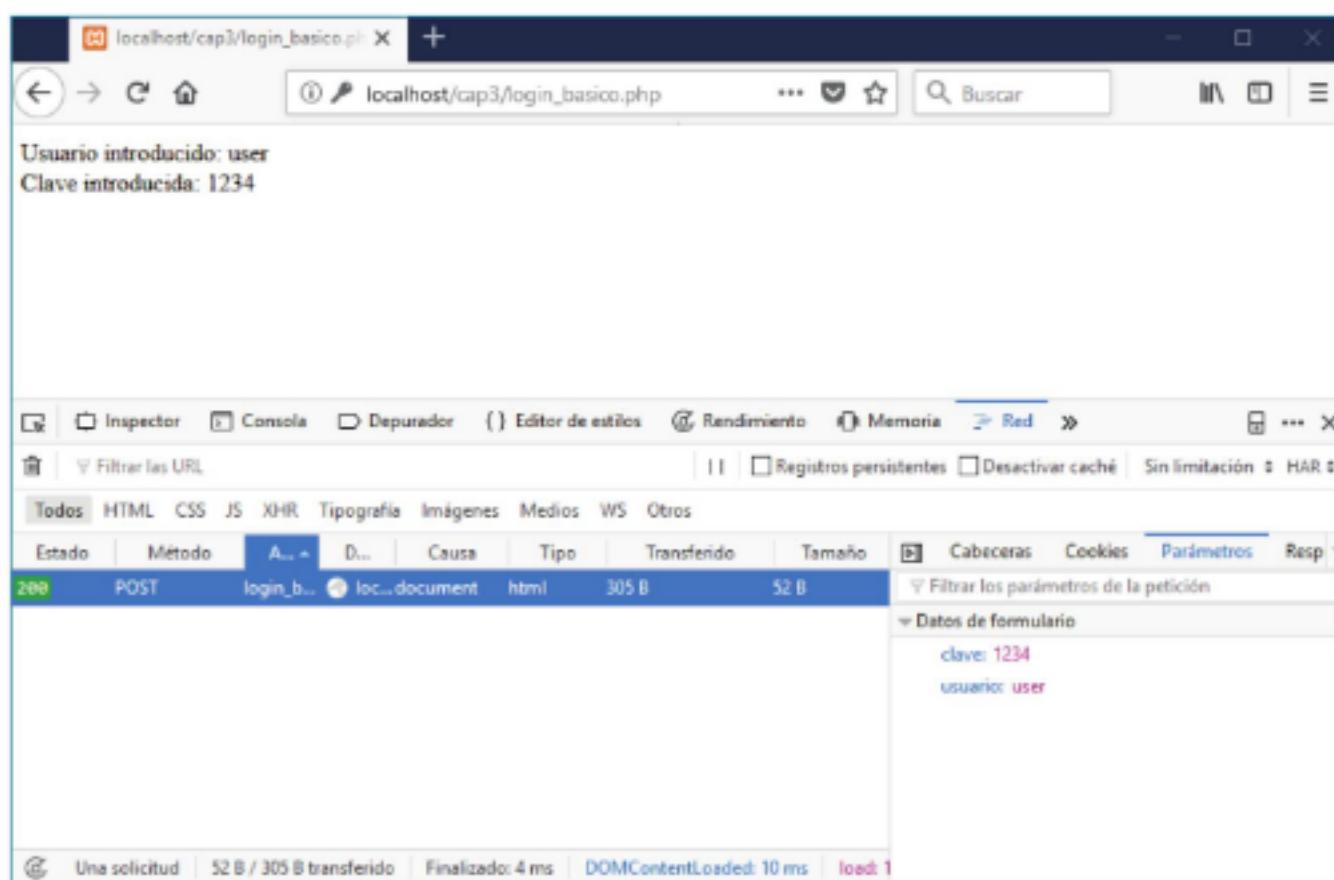
Dentro del elemento *form* se introducen los campos, que tendrá que llenar el usuario, y los botones de envío y para limpiar los campos. Para los campos que se envían hay que usar el atributo *name*, que sirve para identificarlo dentro del *script*. El envío se produce al pulsar el botón.

En el *script* al que se envía el formulario los parámetros están disponibles en el *array* superglobal `$_POST`. La clave de cada argumento dentro del *array* es el atributo `name` del elemento correspondiente en el formulario.

El fichero **login\_basico.php**, al que se envía el formulario anterior, muestra los valores introducidos en él:

```
<?php
echo "Usuario introducido: ". $_POST['usuario']. "<br>";
echo "Clave introducida: ". $_POST['clave'];
```

Al usar el método POST los parámetros no se muestran en la URL, pero se pueden consultar desde la consola del navegador. En Firefox está disponible en Menú > Desarrollador Web > Red. Una vez abierta, muestra las peticiones que se realizan desde el navegador. Si se selecciona la correspondiente al envío del formulario, se pueden consultar los parámetros en la parte derecha.



**Figura 3.1**  
Herramientas de desarrollador en Firefox.

### 3.2.1. Formulario de *login*

En general un formulario de *login* se encarga de comprobar los datos introducidos y, según sean correctos o no, da acceso al sistema al usuario o muestra un mensaje de error. El fichero **login\_falso.php** se encarga de procesar un formulario de *login*. Para simular un formulario de *login*, hay que comprobar que el usuario y la contraseña sean correctos. Si el usuario es “usuario” y la clave es “1234”, se redirige a la página de bienvenida. En caso contrario, lo hace a una página de error. Para la redirección se usa la función `header`, que sirve para escribir en la cabecera de la respuesta HTTP.

```
<?php
/* si va bien redirige a bienvenido.html
si va mal, mensaje de error */
if ($_POST['usuario']=="usuario" and $_POST["clave"]=="1234"){
    header("Location:bienvenido.html");
} else {
    header("Location:error.html");
}
```

**RECUERDA**

- ✓ Hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función `header()` antes de que se empiece a escribir la salida. Si se intenta llamar a `header()` después de haber realizado un `echo`, se producirá un error.

### 3.2.2. Formulario y procesamiento en un solo fichero

En ocasiones el formulario HTML y el bloque PHP que lo procesa se integran en un solo fichero, en el que hay que distinguir entre dos casos. Cuando se accede al formulario para rellenarlo y cuando se envía para procesarlo.

Cuando se accede a la página usando el método GET, es decir, introduciendo la dirección en el navegador, al seguir un vínculo o como resultado de una redirección con `header(Location:)`, se muestra el formulario. En cambio, si se accede mediante POST quiere decir que el cliente está enviado el formulario. Se puede diferenciar entre los dos métodos de acceso consultando `$_SERVER["REQUEST_METHOD"]`.

El ejemplo **form\_en\_uno.php** une los dos ficheros del anterior en uno solo y, en caso de error:

- Muestra un mensaje apropiado encima del formulario.
- Mantiene el valor introducido en el campo usuario.

```
<?php
/* si va bien redirige a principal.php si va mal, mensaje de error */
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if($_POST['usuario']=="usuario" and $_POST["clave"]=="1234"){
        header("Location: principal.php");
    }else{
        $err = true;
    }
}
?>
<!DOCTYPE html>
<html>
```

```

<head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
</head>
<body>
    <?php if(isset($err)) {
        echo "<p> Revise usuario y contraseña</p>";
    }?>
    <form method = "POST"
        action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
        <label for = "usuario">Usuario</label>
        <input value = "<?php if(isset($usuario))echo $usuario;?>">
        id = "usuario" name = "usuario" type = "text">
        <label for = "clave">Clave</label>
        <input id = "clave" name = "clave" type = "password">
        <input type = "submit">
    </form>
</body>
</html>

```

**TOMA NOTA**

Cuando el formulario llama al mismo fichero se recomienda usar:

`action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"`

en lugar del nombre del fichero como aparece en el ejemplo. La variable *superglobal* `$_SERVER["PHP_SELF"]` contiene el nombre del fichero y la función `htmlspecialchars()` sirve para filtrar los caracteres por seguridad.

El bloque inicial de PHP con la comprobación de usuario y contraseña se ejecuta solo cuando se accede por el método POST, es decir, al enviar el formulario.

En ese caso, se comprueban los datos y, si son correctos, se reenvía a la página de bienvenida. Con el reenvío termina la ejecución de **form\_en\_uno.php**. Si no son correctos, se crea la variable `$err` con valor TRUE y el script continúa. El bloque de HTML que contiene el formulario se mostrará tanto si el método es GET como si es POST y la comprobación de datos falló.

Hay un segundo bloque de PHP antes del formulario para imprimir un mensaje de error si la variable `$err` existe, es decir, si se ha enviado el formulario con datos incorrectos. Si se accede por GET el bloque se ejecuta, pero la condición no se cumple y no se muestra el mensaje.

### 3.2.3. Subida de ficheros

Un caso especial en los formularios es la subida de ficheros al servidor. En el formulario hay que usar el atributo `enctype="multipart/form-data"` y el método POST. Para el fichero se usa una etiqueta `<input type = "file">`. Con este control se abre una ventana para que el usuario pueda escoger un fichero en su equipo.