

## Unidad 6

---



# Programación de documentos web utilizando lenguajes de script de servidor

## Implantación de aplicaciones web



# Índice



6.1. Generalidades sobre PHP

6.2. Trabajando con PHP

6.3. Elementos del lenguaje: variables y constantes, tipos de datos, operadores, instrucciones

6.3.1. Variables y constantes

6.3.2. Instrucciones del lenguaje

6.3.3. Operadores

6.4. Entornos de trabajo

6.5. Integración formularios-lenguaje



# Introducción

Existen numerosos métodos para crear una página web, como un desarrollo rápido mediante plantillas o usar directamente un lenguaje de etiquetado como es HTML. Pero es cierto que HTML, aunque se acompañe de CSS carece prácticamente de dinamismo ninguno. Este dinamismo ayuda a que la página permita interactuar y realizar diversas funciones. Por eso, para poder realizar estas funciones se necesita, además de un lenguaje de etiquetado como HTML, un sistema gestor de bases de datos y algún lenguaje de programación de script que funcione en el lado servidor.

Son multitud los lenguajes de script del servidor, pero predominan tres: ASP, JSP y PHP. Con cualquiera de ellos podemos acceder a una base de datos y además, cualquiera de los sitios construidos se visualizará prácticamente igual.

En nuestro caso, vamos a trabajar con PHP y esta unidad nos va a servir como introducción a dicho lenguaje.

## Al finalizar esta unidad

- + Aprenderemos qué es un lenguaje de script del lado del servidor.
- + Conoceremos los tipos de datos.
- + Estudiaremos el uso de variables en el lenguaje de script PHP.
- + Describiremos los distintos tipos de instrucciones de PHP y su funcionamiento.
- + Definiremos las instrucciones en la resolución de scripts en PHP.
- + Conoceremos cómo trabajar con PHP en entornos de desarrollo.
- + Distinguiremos los métodos para procesar los datos enviados desde un formulario a una página PHP.
- + Aprenderemos a programar documentos web.
- + Integraremos los formularios HTML en el lenguaje PHP.



# 6.1.

## Generalidades sobre PHP

PHP se trata de un lenguaje de programación para el desarrollo web en el lado del servidor. Fue creado en 1994 por el programador danés-canadiense Rasmus Lerdorf, y desde entonces ha contado con una gran aceptación y se ha convertido en uno de los lenguajes más utilizados para el desarrollo en el lado del servidor.

El lenguaje PHP es un lenguaje flexible y que posibilita realizar pequeños *scripts* con rapidez. Si se compara, por ejemplo, con Java, se requiere escribir menos código en PHP y, por lo general, resulta menos engorroso. La sintaxis que se utiliza para los elementos básicos es muy similar a la que se utiliza en otros lenguajes muy extendidos como Java y C, por lo que es un lenguaje rápido de aprender para aquellas personas que ya cuentan con cierta experiencia en la programación.

En esta unidad se va a presentar la sintaxis y los elementos básicos del lenguaje PHP, y se espera que el lector tenga ciertos conocimientos con los conceptos básicos de programación estructurada y orientada a objetos.

Cuando se desarrolla, es común utilizar el PHP incrustado dentro de ficheros HTML, y para ello se utiliza la etiqueta `<?php` para abrir el bloque de programación en PHP y la etiqueta `?>` para cerrarlo.

En el siguiente ejemplo se muestra una página HTML completa que incluye un bloque PHP incrustado en las líneas 7-10. Como se puede observar, el bloque cuenta con una única línea para que en la página del servidor se muestre la cadena "Hola mundo".

De modo que, cuando se solicite la página al servidor web, el resultado que se mostrará será:

Hola mundo

Al consultar el código fuente de la página web (pulsando *Ctrl* + *u*), se obtiene:

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

Imagen 2. Código fuente del ejemplo "Hola mundo"

El servidor web ha modificado el bloque de PHP y muestra ahora la salida, que es "Hola mundo". Se utiliza *echo* para que se muestre el valor de una variable o de una cadena de texto.

```
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>
```

Imagen 1. Ejemplo de bloque PHP en fichero HTML





## 6.2.

### Trabajando con PHP

Para trabajar con PHP es suficiente con un editor de texto que nos permita empezar a programar con PHP. Como se trata de un lenguaje multiplataforma, existen diferentes entornos de desarrollo, pero todos tendrán en común que necesitan de un servidor web y de un servidor de bases de datos para trabajar,

Tenemos varias opciones para trabajar con PHP, una de las más usadas es un paquete integrado como es *XAMPP*, que ya lleva por defecto el entorno necesario para trabajar con PHP, instalando los servidores de bases de datos, el servidor web y el lenguaje. Para mayor comodidad, se recomienda usar un IDE o entorno de desarrollo integrado específico, pero esto es opcional.

Da igual que entorno se use, siempre hay que seguir una serie de pasos para que un *script* en PHP funcione, y estos son:



## 6.3.

### Elementos del lenguaje: variables y constantes, tipos de datos, operadores, instrucciones

#### 6.3.1. Variables y constantes

PHP es un lenguaje con *tipado débil*, es decir, que al declarar una variable no es necesario que se declare de qué tipo de dato se va a tratar. De hecho, las variables no se declaran, si no que se crean la primera vez y se les asigna un valor, por lo que el tipo de dato que pueden almacenar estas variables dependerá del valor con el que se inicialicen.

Esto influye en la velocidad de escritura de los programas y en su complejidad, pero también presenta algunos inconvenientes, como que puede dar lugar a código de baja calidad si no se presta la suficiente atención y, a medida que las aplicaciones crecen en complejidad, pueden llegar a ser difíciles de depurar.

#### Declaración de variables

Los identificadores de las variables, en PHP, siempre van precedidos por el carácter '\$', y deben de empezar por una letra, o bien por un guion bajo ('\_'), y puede contener números, letras y guiones bajos.

Para declarar una variable solo se necesita que se le asigne un valor:

```
$variable = valor;
```

Por ejemplo, en la siguiente instrucción se declara la variable *\$numero\_entero*, y será de tipo *integer* porque se le ha asignado un primer valor de número entero.

```
$numero_entero = 3;
```



El valor de la variable se puede modificar simplemente con asignarle otro tipo de valor, al igual que el tipo de variable, como se puede ver en el siguiente ejemplo.

```
1 <?php
2     $variable = 3; // entero
3     echo gettype($variable); // imprime en la consola el tipo de dato de "variable"
4     echo "<br>"; // aplica un salto de línea
5     $variable = "Hola mundo"; // cambia a string
6     echo gettype("$variable"); // comprobamos en consola que ha cambiado
7 ?>
```

Imagen 3. Cambio de tipo de dato en PHP

La salida del ejemplo nos verifica que la variable cambia de tipo de dato:

integer  
string

### Asignación por copia y por referencia

En principio, la asignación de variables se realiza mediante copia:

```
$variable1 = $variable2;
```

En este caso se creará una nueva variable (2) con el valor que tenga la anterior (1). Las dos variables representan posiciones distintas en la memoria, aunque representen el mismo valor en el momento de la asignación.

Se puede también definir una referencia a una variable utilizando el operador *ampersand*:

```
$variable2 = &$variable1;
```

En este caso, \$variable2 no se trata de una nueva variable con el valor de \$variable1, sino que \$variable2 apunta a la misma dirección de memoria que \$variable1, es decir, en realidad estas dos variables son dos nombres para el mismo dato, como se puede observar en el siguiente ejemplo:

```
1 <?php
2     $variable1 = 5;
3     $variable2 = &$variable1; // asignación por referencia
4     $variable3 = $variable1; // asignación por copia
5     echo "$variable2<br>"; // imprime el valor de 5
6     $variable2 = 10; // cambia el valor de $variable2
7     echo "$variable1<br>"; // $variable1 también cambia
8     $variable3 = 20; // este cambio no afecta a $variable1
9     echo $variable1;
10 ?>
```

Imagen 4. Asignación por copia y referencia

En este caso, los valores que se mostrarán en la salida serán:

5  
10  
10



## Variables no inicializadas

Puede darse la situación de que se intente utilizar una variable antes de que se le haya asignado un valor, entonces se generará un error de tipo *E\_NOTICE*, pero que no detendrá la ejecución del *script*. Si esta variable no inicializada se encuentra dentro de una expresión, esta expresión se calculará tomando un valor por defecto para ese tipo de dato.

```
1 <?php
2     $variable1 = 1;
3     $variable3 = 2 + $variable2; // $variable 2 no existe, se toma como 0
4     echo "$variable3 <br>"; // 2 + 0 = 2
5     $variable3 = 2 * $variable2; // $variable 2 no existe, se toma como 0
6     echo "$variable3 <br>"; // 2 * 0 = 0
7 ?>
```

Imagen 5. Operaciones con variables no inicializadas

En la línea 3 se intenta realizar una operación con una variable no inicializada, el valor que se toma por defecto es 0, por lo que el resultado de la operación será 2. Lo mismo sucede en la línea 5, donde esta vez, al ser una multiplicación y tomar como valor por defecto 0, el valor resultante es 0.

El *script* no se detiene, pero muestra una serie de errores en la consola haciendo referencia a la utilización de variables no inicializadas:

```
PHP Warning: Undefined variable $variable2 in
/workspace/Main.php on line 3
```

```
2
```

```
PHP Warning: Undefined variable $variable2 in
/workspace/Main.php on line 5
```

```
0
```

## Constantes

Las constantes son valores que no pueden ser modificados o alterados durante la ejecución del *script*, y para declararlas se utiliza la función *define()*, donde se introduce el nombre de la constante y el valor que se le quiere dar.

```
define("CONSTANTE", 50);
```

Es común utilizar las mayúsculas para identificar las constantes.

## Tipos de datos escalares

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



## Ámbito de las variables

El ámbito de una variable se define como aquella parte del código en la que es visible, es decir, si se declara una variable en un fichero cualquiera de PHP, esa variable sólo estará disponible en ese fichero y en los ficheros que se incluyan desde este. Por otra parte, las funciones definen un ámbito local, por lo que las variables que hayan sido definidas dentro de la función no son accesibles desde otras funciones, y, siguiendo la misma lógica, las variables que no sean locales, o sus argumentos, no podrán ser utilizadas dentro de la función.

Para poder utilizar una variable que sea accesible desde cualquier función o fichero de la aplicación se utiliza la palabra reservada *global*, y la variable predefinida `$_GLOBALS`.

## Variables predefinidas

En PHP existen una gran cantidad de variables predefinidas disponibles, contienen información sobre el servidor, datos que ha enviado el cliente o variables de entorno. Dentro de las variables predefinidas existe un grupo de ellas, denominadas *superglobales*, que están disponibles en cualquier ámbito. Cada una guarda un tipo de información.

```
1 <?php
2     echo "Nombre del servidor: " . $_SERVER['SERVER_NAME'];
3     echo "Software del servidor: " . $_SERVER['SERVER_SOFTWARE'];
4     echo "Protocolo: " . $_SERVER['SERVER_PROTOCOL'];
5 ?>
```

Imagen 6. Ejemplo de uso de la variable `$_SERVER`

Variables superglobales	
Nombre	Descripción
<code>\$_GLOBALS</code>	Variables globales
<code>\$_SERVER</code>	Información del servidor
<code>\$_GET</code>	Parámetros enviados con el método <i>GET</i> (en la URL)
<code>\$_POST</code>	Parámetros enviados con el método <i>POST</i> (formularios)
<code>\$_FILES</code>	Ficheros subidos al servidor
<code>\$_COOKIE</code>	<i>Cookies</i> 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





### 6.3.2. Instrucciones del lenguaje

PHP dispone de las estructuras de control habituales que permiten modificar el flujo de ejecución de las instrucciones de un programa.

#### Instrucciones de entrada, salida y asignación

Las instrucciones de entrada en PHP son las que permiten recibir los datos desde el exterior de un programa para luego trabajar con estos datos. En PHP, se necesita por lo general de un control de formulario para poder recoger los valores.

Las instrucciones de salida permiten que se muestre cierta información en la pantalla con la instrucción *echo* o la instrucción *print*.

Por último, las instrucciones de asignación se usan para a raíz de la interpretación de alguna instrucción, almacenar el resultado en una variable.

#### Instrucciones alternativas o condicionales

En PHP, las estructuras condicionales son *if*, *if-else*, *if-elseif* y *switch*.

La sintaxis para el *if* es:

```
if (condición)
    instrucción
```

En el caso de que se cumpla la condición establecida, la instrucción que se sitúa dentro del *if* se ejecutará, y si no se cumple, no se ejecutará. Si se quiere introducir más de una sentencia dentro del *if*, se encerrarán entre llaves.

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

Para evaluar la condición se utilizará verdadero o falso, y, en caso de ser necesario, se seguirán las normas de conversión a *boolean*.

```
1 <?php
2     $variable = 1;
3     if($variable > 3) echo "Es mayor que tres";
4     if($variable < 3){
5         echo "Es menor que tres";
6     }
7 ?>
```

Imagen 7. Ejemplo de condicional

La salida que se mostrará con este ejemplo es:

**Es menor que tres**

La estructura *else* se utiliza para ejecutar instrucciones cuando la condición del *if* no se cumple.



```
1 <?php
2     $variable = 1;
3     if($variable > 3){
4         echo "Es menor que tres";
5     }else{
6         echo "Es menor o igual a tres";
7     }
8 ?>
```

Imagen 8. Condicional con *else*

La salida que se mostrará en este ejemplo es:

**Es menor o igual que tres**

Si se necesitan utilizar varias sentencias condicionales puede utilizarse el *elseif*, que es equivalente a *else if*.

```
1 <?php
2     $variable = 1;
3     if($variable == 1){
4         echo "Es un uno";
5     }elseif($variable == 2){
6         echo "Es un dos";
7     }else if($variable == 3){
8         echo "Es un tres";
9     }else{
10        echo "No es ni un uno, ni un dos, ni un tres";
11    }
12 ?>
```

Imagen 9. Condicionales anidados

Cuando existen varios condicionales anidados, la primera condición que se cumple es la que se ejecuta, y en caso de que no se cumpla ninguna condición, se ejecutará el *else* final (en caso de que lo haya).

Si se desean agrupar varios *if*, puede ser útil aplicar la estructura del *switch*, esta estructura está presente también en otros lenguajes. El siguiente ejemplo realiza la misma función que el anterior, pero con una estructura más legible.

```
1 <?php
2     $variable = 1;
3     switch($variable){
4         case 1:
5             echo "Es un uno";
6             break;
7         case 2:
8             echo "Es un dos";
9             break;
10        case 3:
11            echo "Es un tres";
12            break;
13        default:
14            echo "No es un uno, ni un dos, ni un tres";
15    }
16 ?>
```

Imagen 10. Estructura del *switch*

Dependiendo del valor que tenga *\$variable*, se ejecutará un *case* u otro, y la sección *default* se ejecutará en el caso de que el valor de *\$variable* no coincida con ninguno de los *case*.

El *break* al final de cada *case* sirve para que, cuando el valor de la variable coincida con el *case*, deje de evaluar los demás casos, si no se colocara el *break*, aun cuando haya encontrado una coincidencia, seguiría evaluando los demás casos, y podría producirse que se ejecutaran dos o más casos.



## Instrucciones repetitivas

Las estructuras de repetición, o bucles, sirven para repetir una secuencia de instrucciones de código repetidas veces mientras se cumpla una condición. Se tienen las estructuras de *for*, *while* y *do-while*, y se utiliza la misma sintaxis que en Java o en C.

La sintaxis del bucle *for* es:

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

La instrucción de inicialización solo se ejecuta cuando se entra por primera vez al bucle, y las instrucciones del bucle se ejecutarán hasta que se deje de cumplir la condición del bucle.

Una vez se ha ejecutado la instrucción de inicialización se comprueba la condición, en caso de que se cumpla, se entrará en el bucle y se seguirán las instrucciones de iteración. Luego se vuelve a comprobar si se cumple la condición, y en caso de que se cumpla se volverá a realizar la instrucción del bucle. Y este proceso se repetirá hasta que la condición deje de cumplirse, y a partir de ahí se seguirá ejecutando el código que está después del bucle.

```
1 <?php
2   for($i = 0; $i < 4; $i = $i + 1){
3       echo \"$i <br>\";
4   }
5 ?>
```

Imagen 11. Bucle for

En el ejemplo anterior se tiene un bucle *for* que se repite 4 veces. Se tiene una variable *\$i* que se inicializa en cero, y su valor aumenta en una unidad en cada iteración mientras que sea menor que 4, en el momento en el que se *\$i* vale 4, la condición del bucle *for* ya no se cumple y se sale del bucle.

La salida de este *script* es:

```
0
1
2
3
```

La sintaxis del bucle *while* es:

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

Se sigue la misma lógica que con el bucle *for*, mientras se cumpla una condición se ejecutará el bucle. El *while* no utiliza instrucciones de inicialización, pero se pueden colocar antes del bucle, y tampoco instrucciones de bucle, pero se pueden colocar dentro de este.



```

1 <?php
2     $i = 0; // instrucciones de inicialización
3     while($i < 4){
4         echo "$i <br>"; // instrucciones del bucle
5         $i = $i + 1; // instrucciones de iteración
6     }
7 ?>

```

Imagen 12. Ejemplo de bucle *while*

En el bucle *do-while* se utiliza una sintaxis similar, con la peculiaridad de que la condición se comprueba después de ejecutar las instrucciones del bucle.

```

do{
    instrucciones;
}while (condición);

```

```

1 <?php
2     $i = 0;
3     do{
4         echo "$i <br>";
5         $i = $i + 1;
6     } while ($i < 4);
7 ?>

```

Imagen 13. Ejemplo de bucle *do-while*

Que en la estructura del *do-while* la condición se compruebe después de ejecutar el bucle provoca que las instrucciones del bucle se ejecutarán, al menos, una vez.

Dentro de un bucle se pueden utilizar las sentencias de *break* y *continue*.

El *break* sirve para abandonar inmediatamente el bucle o *switch* en el que aparezca, independientemente de que se cumpla la condición o no.

```

1 <?php
2     $i = 0;
3     while ($i < 4){
4         echo "$i <br>";
5         $i++; // esto funciona como $i = $i + 1
6         if ($i == 2){
7             break;
8         }
9     }
10 ?>

```

Imagen 14. Ejemplo de *break*

En el anterior ejemplo, aunque tenemos un bucle con la condición de que *\$i* tiene que ser menor que cuatro, dentro del bucle tenemos una instrucción de que si *\$i* llega a 2 se ejecute el *break*, por lo que se ha de abandonar el bucle, por lo que la salida de este *script* es:

```

0
1

```

En PHP en la sentencia de *break* se le puede introducir un número, que indica el número de niveles de anidación que debe de abandonar. De este modo se puede salir de un bucle anidado utilizando una única secuencia *break*.



```

1 <?php
2     echo "Primer for: <br>";
3     for ($i = 0; $i < 3; $i++) {
4         for ($j = 0; $j < 3; $j++) {
5             echo "i: $i, j: $j <br>";
6             if ($j == 1) {
7                 break; // es lo mismo que poner break 1
8             }
9         }
10    }
11    echo "Segundo for: <br>";
12    for ($i = 0; $i < 3; $i++) {
13        for ($j = 0; $j < 3; $j++) {
14            echo "i: $i, j: $j <br>";
15            if ($j == 1) {
16                break 2;
17            }
18        }
19    }
20 ?>

```

Imagen 15. Distintos tipos de *break*

Como se puede ver en el anterior ejemplo, se tiene un primer *for* que funciona a modo de contador, con la variable *\$i*, y dentro de este *for* se tiene otro con la variable *\$j*, el condicional que se sitúa en el interior se ejecuta cuando *\$j* llega a uno, y el *break* hace que se abandone únicamente el *for* del interior.

En el segundo *for* anidado se tiene la misma estructura, con el único cambio de que al *break* le sigue el número dos, indicando que, cuando se cumpla la condición de *\$j == 1*, se ejecutará el *break* y se abandonarán los dos bucles *for*. La salida quedaría, por tanto:

Primer bloque for:

```

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 bloque for:

```

i: 0, j: 0
i: 0, j: 1

```

Esto se aplica también a la sentencia condicional *switch*, por ejemplo, si se tiene un *switch* dentro de un bucle *while* y se ejecuta un *break 2* se sale de ambos.

La sentencia *continue* tiene como finalidad la de forzar una nueva repetición del bucle, de modo que las instrucciones que estén después de la sentencia *continue* no se ejecutarán, y, si se sigue cumpliendo la condición del bucle, se ejecutará una nueva iteración.

```

1 <?php
2     for ($i = 0; $i < 4; $i++) {
3         if ($i == 2) {
4             continue;
5         }
6         echo "$i <br>";
7     }
8 ?>

```

Imagen 16. Ejemplo de *continue*





En este caso, la salida es:

```
0
1
3
```

Observamos que, cuando se cumple la condición de `$i == 2`, se ejecuta la sentencia de *continue* y el bucle salta a la siguiente iteración.

### 6.3.3. Operadores

En PHP se pueden utilizar los operadores habituales para las operaciones aritméticas, lógicas, de manipulación de cadenas y demás. En los operadores de comparación cabe destacar los operadores `"=="` y `"!=="`, denominados Idéntico y No Idéntico, que no existen en todos los lenguajes. El operador Idéntico es utilizado para hacer comparaciones entre dos expresiones, y evalúa como verdadero cuando las dos expresiones tienen el mismo valor y tipo de dato. La diferencia con el operador Igual, `"=="`, radica en que, cuando se comparan expresiones que no tienen el mismo tipo de dato, intenta convertirlas antes de compararlas. El operador Idéntico se utiliza para evitar que surjan confusiones durante la conversión de datos.

```
1 <?php
2     $var1 = 1;
3     $var2 = "1";
4     if ($var1 == $var2) {
5         echo "Las variables son iguales <br>";
6     } else {
7         echo "Las variables son distintas <br>";
8     }
9     if ($var1 === $var2) {
10        echo "Las variables son idénticas <br>";
11    } else {
12        echo "Las variables no son idénticas <br>";
13    }
14 ?>
```

Imagen 17. Ejemplo de operadores de comparación

En el ejemplo, el primer operador de comparación realiza la conversión de la cadena "1" y la convierte a un número entero, por lo que detecta que las variables son iguales. La salida será:

```
Las variables son iguales
Las variables no son idénticas
```

Operadores de comparación	
<code>a === b</code>	Idéntico. Si las variables tienen el mismo tipo de dato y valor será verdadero.
<code>a == b</code>	Igual. Si las dos expresiones, tras la conversión de tipos, tienen el mismo valor, será verdadero.
<code>a !== b</code>	No idéntico.
<code>a != b, a &lt;&gt; b</code>	No igual.
<code>a &gt;= b, a &gt; b, a &lt;= b, a &lt; b</code>	Mayor o igual, mayor, menor o igual, menor.
<code>a ?? b ?? c</code>	Empezando por la izquierda, devuelve la primera expresión no nula.



Operadores aritméticos	
<code>+a</code>	Teniendo un único operador y argumento, sirve para transformar la expresión a <i>integer</i> o <i>float</i> , según corresponda.
<code>-b</code>	Teniendo un único operador y argumento, sirve para cambio de signo
<code>a + b, a - b, a * b, a / b</code>	Suma, resta, multiplicación, división.
<code>a % b</code>	Módulo.
<code>a ** b</code>	Potencia.
Operadores lógicos	
<code>a and b, a &amp;&amp; b</code>	Y, es verdadero si las dos expresiones se evalúan a TRUE.
<code>a or b, a    b</code>	O. Si una o las dos expresiones es TRUE, se evalúa como verdadero.
<code>a xor b</code>	O exclusivo. Es verdadero si solo una de las dos expresiones es TRUE.
<code>!a</code>	Es verdadero si la expresión es FALSE, y viceversa.
Operadores a nivel de bit	
<code>a &amp; b, a   b, a ^ b, ~a</code>	Y, O, O exclusivo y negación.
<code>\$a &gt;&gt; \$b</code>	Desplaza los bits de \$a, \$b pasos a la derecha (cada paso es dividir por dos).
<code>\$a &lt;&lt; \$b</code>	Desplaza los bits de \$a, \$b pasos a la izquierda (cada paso es multiplicar por dos).
Operadores de asignación	
<code>\$a = b</code>	Asignación por valor.
<code>\$a = &amp;\$b</code>	Asignación por referencia.
<code>\$a += b, \$a -= b, \$a *= b, \$a /= b</code>	Equivalente a $\$a = \$a + b$ , $\$a = \$a - b$ ... Es válido para cualquier operador binario aritmético, de cadenas o <i>arrays</i> .
Otros operadores	
<code>\$a++, \$a--</code>	Devuelve \$a, luego le suma (o resta) 1.
<code>++\$a, --\$a</code>	Suma (o resta) 1 a \$a, devuelve el valor actualizado
<code>\$str1 . \$str2</code>	Concatena dos cadenas.



# 6.4.

## Entornos de trabajo

Como IDE se puede utilizar un entorno ligero como Notepad++, y, en caso de que se prefiera un IDE con más funcionalidades, se puede optar por:

- > **Eclipse PDT.** Es la versión de Eclipse para PHP.
- > **Aptana.** Si se desea integrar el desarrollo en la parte del cliente puede ser una buena opción. Se basa en Eclipse y está disponible como *plugin* y también como aplicación independiente.
- > **PHPStorm.** Es un entorno muy completo, pero de pago. Tiene una versión de prueba de un mes.

Aunque estos entornos puedan parecer demasiado complejos o excesivos para los primeros ejemplos, a medida que se avance en complejidad es conveniente usar un buen IDE.

# 6.5.

## Integración formularios-lenguaje

A través de los formularios HTML se pueden enviar datos a un servidor, el usuario puede rellenar varios campos utilizando distintos tipos de controles, como campos de texto o botones de radio y son enviados al servidor al pulsar un botón. El servidor se encarga de procesar los datos del formulario que se ha recibido y genera la respuesta.

Un sencillo formulario de *login* en HTML quedaría de la siguiente manera:

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Formulario de acceso</title>
5    </head>
6    <body>
7      <form action = "credenciales.php" method = "POST">
8        <input name = "usuario" type = "text">
9        <input name = "contraseña" type = "password">
10       <input type = "submit">
11     </form>
12   </body>
13 </html>

```

Imagen 18. Formulario de acceso



Con el atributo *action* se introduce el destino donde se enviará el formulario para que se procese. En caso de usar una ruta relativa, se tomará como punto de partida la localización donde se encuentra el fichero que contenga el formulario.

Con el atributo *method* se especifica el método HTTP que se utilizará para la petición. Se suele utilizar POST, pero también se puede utilizar GET, aunque usando este último los parámetros aparecerán en la URL.

En el interior del elemento *form* se introducen los campos que deben ser rellenados por el usuario, y los botones de envío y para limpiar los campos. Para los campos que van a ser enviados se utiliza el atributo *name*, con el que se identifica dentro del *script*. Para enviar el formulario se debe apretar el botón de *Submit Query*.

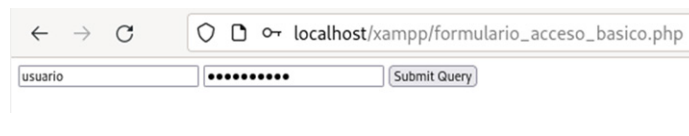


Imagen 19. Formulario

Para que el *script* pueda recibir los parámetros que han sido enviados se utiliza el *array* superglobal *\$\_POST*, y la clave de cada argumento dentro del *array* es el atributo *name* del elemento correspondiente del formulario.

```
1 <?php
2     echo var_dump($_POST). "<br>";
3 ?>
```

Imagen 20. Array recibido

Con el anterior *script* se puede observar qué elemento es el que se recibe con el formulario.

 string(7) "usuario" ["contraseña"]=> string(11) "contraseña" }" data-bbox="208 566 641 619"/>

Imagen 21. Array que se recibe con el formulario

Si modificamos el *script* del fichero que recibe la información del formulario:

```
1 <?php
2     echo "Usuario introducido: ". $_POST['usuario']. "<br>";
3     echo "Clave introducida: ". $_POST['contraseña'];
4 ?>
```

Imagen 22. Script del fichero *credenciales.php*

Los valores de salida que muestra son:

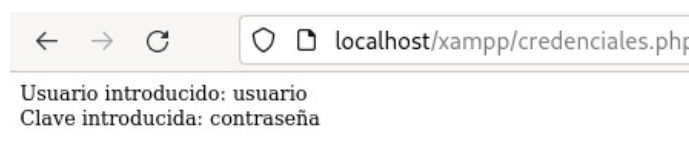


Imagen 23. Valores de salida de *credenciales.php*



Si en vez del método *POST* se hubiera utilizado el método *GET*, la URL que se mostraría sería:

`localhost/xampp/credenciales.php?usuario=usuario&contraseña=contraseña`

Si se ha utilizado el método *POST* y se desean conocer los parámetros, se pueden consultar desde la consola del navegador, el ejemplo que se va a mostrar se ha realizado desde el navegador Firefox, y se consultan en la parte inferior derecha, en el apartado de red y al seleccionar la petición correspondiente al envío del formulario.

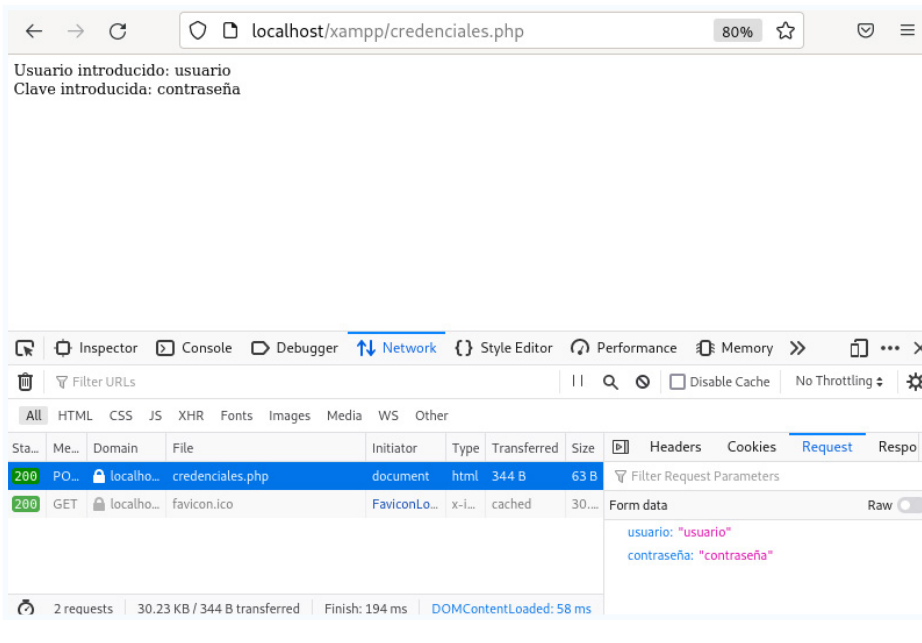


Imagen 24. Consultar parámetros desde la consola del navegador







 [www.universae.com](http://www.universae.com)

