



# PHP - Introducción

---



**Ciclo:** DAW  
**Módulo:** DWES  
**Curso:** 2020-2021  
**Autor:** César Guijarro Rosaleny



<b>Introducción.....</b>	<b>3</b>
<b>Crear archivos php.....</b>	<b>4</b>
Ejecutar archivos .php.....	5
Archivo principal.....	6
<b>Estructura archivo PHP.....</b>	<b>8</b>
Etiquetas de apertura y cierre.....	8
<b>Sintaxis básica PHP.....</b>	<b>11</b>
echo y print.....	11
Comentarios.....	14
Separador de instrucciones.....	14
<b>Variables.....</b>	<b>15</b>
Ámbito de las variables.....	17
Tipo de datos.....	19
Booleanos.....	20
Números enteros.....	21
Número de punto flotante.....	22
Cadena de caracteres.....	23
isset y unset.....	26
<b>Constantes.....</b>	<b>27</b>
<b>Operadores.....</b>	<b>28</b>
Operadores aritméticos.....	28
Operadores de comparación.....	28
Operadores para string.....	29
Operadores de asignación.....	29
Operadores para arrays.....	29
Operador ternario.....	30
Operadores nave espacial y fusión de null.....	31
<b>Estructuras de control.....</b>	<b>33</b>
Condicionales.....	33
Bucles.....	36
<b>Incluir archivos externos.....</b>	<b>39</b>
Include y require.....	39
<b>Excepciones.....</b>	<b>43</b>
<b>Bibliografía.....</b>	<b>45</b>

## Introducción

---

**PHP** (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

PHP es un lenguaje del lado del servidor. Lo que distingue a este tipo de lenguajes de los lenguajes del lado del cliente (como Javascript), es que el código es ejecutado en el servidor, y éste generará código HTML que será enviado al cliente. Por lo tanto, **el cliente recibirá el resultado de ejecutar el script, aunque no sabrá el código PHP que lo ha generado.**

Aunque es posible crear páginas con sólo código PHP (de hecho, es lo recomendable en la mayoría de casos como veremos más adelante), es muy habitual ver partes de código PHP incrustadas en páginas html.

A diferencia de los archivos html, los ficheros PHP no son interpretados por los navegadores, con lo que no podemos abrirlos directamente en ninguno de ellos. Tiene que ser el servidor HTTP el que los interprete y genere el correspondiente archivo .html, para lo cual tendrá que tener instalado un intérprete PHP.

PHP es un lenguaje débilmente tipado, lo que nos permite, entre otras cosas, no declarar el tipo de las variables. Por este motivo tiene muchos detractores, pero ¿significa eso que es un mal lenguaje de programación?. No, el hecho de que nos permita programar “mal” no significa que tengamos que hacerlo. Al final la eficacia y el rendimiento de un lenguaje de programación depende más de la forma en que programamos que del propio lenguaje.

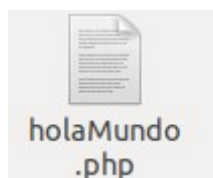
## Crear archivos php

---

Para crear archivos PHP podemos utilizar cualquier editor de textos (incluso el Notepad de Windows o el editor de texto de Linux), aunque lo recomendable es usar algún editor de textos enriquecido que nos de ventajas como resaltado de código, funciones de debug, referencia del lenguaje...

Los archivos PHP pueden tener cualquier nombre válido de fichero (aunque es bueno seguir una serie de recomendaciones para facilitar el trabajo) seguido de la extensión **.php**.

Vamos a empezar a programar con PHP. Para eso, crea el archivo *holaMundo.php* en la carpeta en *htdocs\_apache/DWS/Ejemplos/T1/Ej1*.



Dentro del archivo, escribe el siguiente código (aunque es muy sencillo, no te preocupes ahora por lo que hace el código, ya lo irás aprendiendo a lo largo del curso):

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

No te olvides de la etiqueta de apertura **<?php**. Por ahora no hace falta cerrarla, ya veremos después porqué.

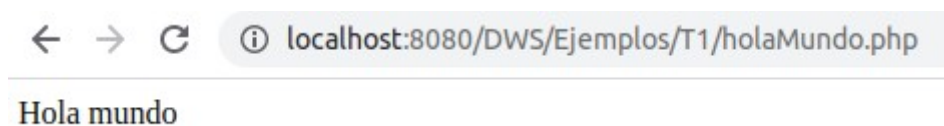
Si intentamos ejecutar el archivo haciendo doble clic, el sistema operativo nos dirá que no sabe con qué programa abrirlo o lo abrirá para editarlo con el editor de textos por defecto.

Si lo arrastramos a un navegador (Chrome, por ejemplo) para ejecutarlo directamente en él, nos abrirá el gestor de descargas para descargar nuestro archivo (lo cual, obviamente, no nos sirve de mucho).

## Ejecutar archivos .php

Vamos a ejecutar nuestro archivo *holaMundo.php* de forma que podamos ver el resultado esperado en nuestro navegador.

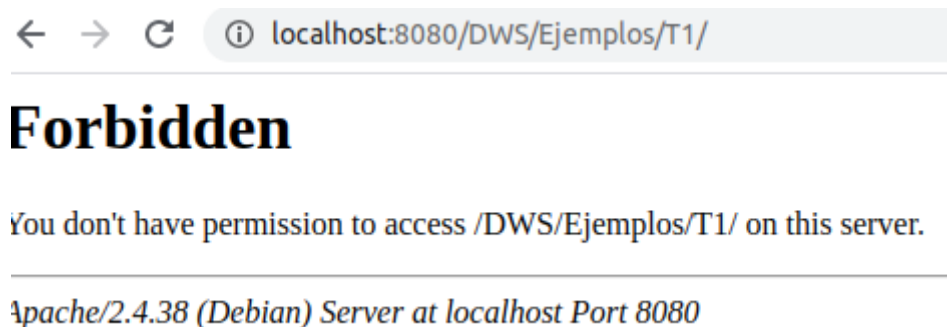
Arranca el servidor Apache y accede a la siguiente url <http://localhost/DWS/Ejemplos/T1/holaMundo.php>. Si todo ha ido bien, deberías ver algo parecido a ésto en tu navegador (el puerto 8080 detrás de localhost no tienes porque verlo en tu navegador. Incluso puedes ver otro diferente si has modificado el puerto en el que escucha Apache):



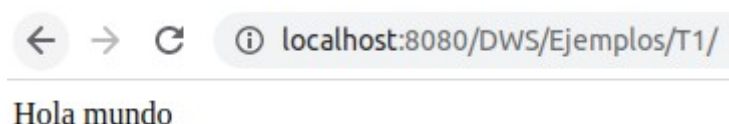
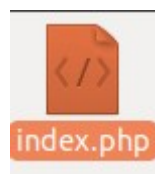
¡Enhorabuena! Acabas de crear tu primer script PHP.

## Archivo principal

Nuestro archivo se llama *holaMundo.php*, y para acceder a él desde el navegador tenemos que poner la url completa y el nombre del fichero. Si intentamos acceder a <http://localhost/DWS/Ejemplos/T1> sin especificar el nombre del archivo, el navegador nos dirá que no tenemos permiso para acceder a esa url (en realidad lo que está pasando es que no sabe que archivo abrir en esa ubicación).



Ahora vamos a renombrar nuestro archivo a *index.php* y volvemos a intentar acceder a <http://localhost/DWS/Ejemplos/T1> a ver que pasa.



¿Porqué ahora sí que nos muestra el contenido del archivo? La respuesta, como estarás imaginando, es sencilla. Al igual que ocurre con los archivos html, Apache asume que al acceder a una carpeta el archivo principal que tiene que cargar es *index.php*.

¿Y si tenemos otro archivo *index.html* en la misma carpeta? **No es recomendable tener dos mismos archivos *index* con distinta extensión en la misma carpeta.** Si es así, Apache mostrará el que aparece primero. En cualquier caso, como regla general no guardes dos *index* nunca en la misma carpeta.

**NOTA:** En la configuración de Apache podemos modificar el archivo que cargará por defecto cuando entremos en esa url.

Vuelve a renombrar el archivo anterior a *holaMundo.html* para que no nos lo cargue cuando ejecutemos los ejemplos de este tema (cuando nuestras webs sean más complejas, cada ejemplo estará en una carpeta diferente, con lo que ya no tendremos ese problema).

## Estructura archivo PHP

---

Un archivo PHP puede contener cualquier código válido de los archivos html (html, css, javascript...), así como trozos de código PHP.

Como hemos dicho antes, lo mejor es separar el código PHP del html en diferentes archivos (igual que siempre es recomendable separar el css o el javascript), aunque para webs sencillas se puede incrustar el PHP directamente en el html.

### Etiquetas de apertura y cierre

Para incrustar código PHP dentro del html tenemos que indicar al intérprete donde comienza y termina. Ésto lo hacemos mediante la etiqueta de apertura (**<?php**) y cierre (**?>**). Cuando PHP analiza un fichero busca esas etiquetas, y considera todo lo que hay dentro código PHP, mientras ignora el resto.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <?php
      echo "Hola mundo";
    ?>
  </body>
</html>
```



También existe la posibilidad de usar la etiqueta abreviada `<?` como apertura, aunque **su uso está desaconsejado**, porque puede generar conflictos, por ejemplo, con ciertas etiquetas usadas en XML (`<?xml`).

¿Porqué en nuestro fichero *holaMundo.php* no hemos usado la etiqueta de cierre? Cuando un fichero contiene sólo código PHP (o termina con código PHP sin nada detrás), es recomendable no usar la etiqueta de cierre, así se previene la adición de espacios en blanco o nuevas líneas accidentales después de la etiqueta de cierre, lo cual causaría efectos no deseados debido a que PHP comenzará la salida del búfer cuando no había intención por parte del programador de enviar ninguna salida en ese punto del script.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    Hola Mundo
  </body>
</html>

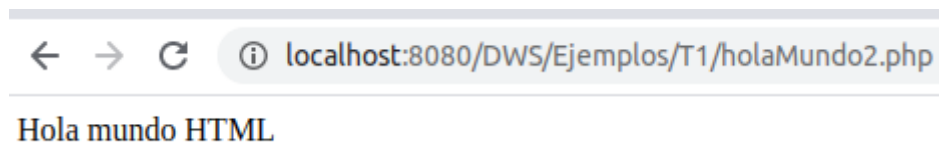
<?php
  # codigo PHP...
```

Los fragmentos PHP se pueden insertar en cualquier lugar del archivo. Lo único que hay que tener en cuenta es que el analizador interpretara el código de forma secuencial, con lo que mostrará la posible salida generada justo en el lugar donde está el código PHP.

Crea otro archivo *holaMundo2.php* con el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    Hola Mundo HTML
  </body>
</html>
```

Accede a <http://localhost/DWS/Ejemplos/T1/holaMundo2.php> desde el navegador y comprueba el resultado.



Si te fijas, el archivo anterior no contiene nada de código PHP, sin embargo el interprete de PHP lo analiza y lo sirve sin ningún problema. Es posible crear archivos PHP sin código PHP, aunque no tiene mucho sentido, ya que el archivo será analizado para buscar código PHP, con lo que se ralentizará el proceso.

Otra cosa a tener en cuenta es que dentro de las etiquetas **<?php ?>** sólo puede haber código PHP. Si metemos algo de código HTML (o de cualquier otro tipo que no sea PHP), el analizador nos dirá que hay errores en la página.

## Sintaxis básica PHP

---

### echo y print

La sentencia **echo** de PHP, imprime una o más cadenas **en el fichero**.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <?php
      echo "Hola mundo";
    ?>
  </body>
</html>
```

Cuando el interprete analiza el fichero anterior lo convierte en:

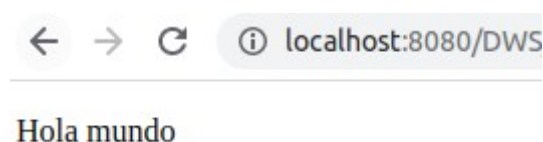
```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    Hola mundo;
  </body>
</html>
```

**NOTA:** La etiqueta **<?=** equivale a **<?php echo ""**. Por lo tanto, la instrucción **<?= "Hola Mundo">** es equivalente a **<?php echo "Hola Mundo" ?>**.

Dentro de un **echo** se pueden añadir etiquetas html.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <?php
      echo "<p>Hola mundo</p>";
    ?>
  </body>
</html>
```

El ejemplo anterior nos sirve para entender la diferencia entre imprimir la cadena en el fichero o en el navegador. Si ejecutamos el archivo anterior la salida es:



Si vemos el código fuente (código html) desde el navegador, nos saldrá algo parecido a ésto:

```
1 <!DOCTYPE html>
2 <html lang="es">
3   <head>
4     <meta charset="utf-8">
5     <title>Ejemplo</title>
6   </head>
7   <body>
8     <p>Hola mundo</p>
9   </body>
</html>
```

Si la sentencia **echo** imprimiera la cadena en el navegador, la salida debería ser:



Básicamente, lo que hace el interprete de PHP es transformar primero la página .php en .html y después enviarla al navegador (simplificándolo mucho, obviamente), con lo que el navegador interpreta las etiquetas html como toca.

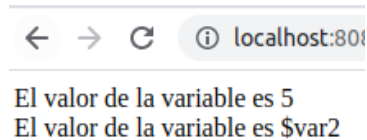
La etiqueta **print** funciona casi igual que **echo**. La diferencia principal es que con **echo** se pueden imprimir varias líneas a la vez separadas por coma, mientras que **print** sólo puede imprimir una línea.

```
<?php
    echo "Hola mundo", "Hola continente";
    print "Hola mundo";
    print "Hola continente";
?>
```

Podemos utilizar comillas simples o dobles para definir las cadenas a imprimir. La diferencia fundamental es que con comillas dobles PHP puede interpretar las variables que contiene y con las simples no.

```
<?php
    $var1 = 5;
    $var2 = 10;
    echo "El valor de la variable 1 es $var1<br />";
    echo 'El valor de la variable 2 es $var2';
?>
```

La salida del ejemplo anterior es:



El valor de la variable es 5  
El valor de la variable es \$var2

**NOTA:** PHP no inserta un salto de línea después de una sentencia **echo** o **print**, con lo que si queremos que haya un *intro* entre dos cadenas, deberemos añadir **<br />** al final de cada una de ellas.

## Comentarios

Podemos añadir comentarios a un fichero PHP mediante la etiqueta **//** si el comentario ocupa una sola línea, con **/\* \*/** si queremos comentar un bloque entero, o con **#** si queremos comentar el final de una línea.

```
<?php
    //comentario de una sólo línea
    $var1 = 5;
    /*comentario
    de varias líneas */
    $var2 = 10; #comentario al final de una línea
?>
```

## Separador de instrucciones

En PHP las diferentes instrucciones se tienen que separar con **;**. Un error común al empezar a programar con PHP es olvidar los punto y coma después de cada instrucción

## Variables

---

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable **es sensible a mayúsculas y minúsculas**.

```
<?php
    $nombre = "Hola";
    $Nombre = " mundo";
    echo $nombre . $Nombre; #imprime Hola mundo
?>
```

Como hemos dicho antes, no hace falta declarar el tipo de variable (ni siquiera definirla):

```
<?php
    echo "El valor de la variable 1 es $var1";
?>
```

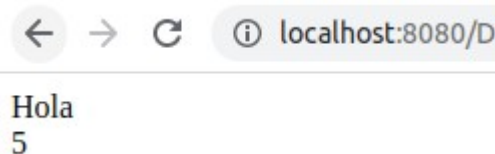
El anterior código no daría ningún error (aunque no hemos definido antes **\$var1**). Mostraría la siguiente salida:



The screenshot shows a web browser window with a navigation bar at the top containing back, forward, and refresh buttons, along with the address bar showing 'localhost:8080/DW...'. Below the navigation bar, the text 'El valor de la variable es' is displayed.

Incluso podemos cambiar el tipo de variable en un momento dado, sin que nos muestre ningún tipo de error.

```
<?php
    $var1 = "Hola <br />";
    echo $var1;
    $var1 = 5;
    echo $var1;
?>
```



No es necesario tampoco inicializar las variables, sin embargo la mayoría de veces es aconsejable. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo dependiendo del contexto en el que son usadas.

De forma predeterminada, las variables en PHP siempre se asignan por valor. Si queremos cambiar asignar por referencia debemos utilizar **&**.

```
<?php
    $var1 = 5;
    $var2 = $var1;
    $var3 = &$var1;
    echo $var2; #imprime 5
    echo $var3; #imprime 5
    $var1 = 7;
    echo $var2; #imprime 5
    echo $var3; #imprime 7
?>
```





## Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
<?php
    $a = 5;
    include "archivo2.php";
```

Aquí, la variable `$a` estará disponible en el interior del script incluido `archivo2.php`. Sin embargo, en el interior de las funciones definidas por el usuario se introduce un ámbito local a la función. Cualquier variable usada dentro de una función está, por omisión, limitada al ámbito local de la función. Por ejemplo:

```
<?php
    $a = 5;

    function test() {
        echo "El valor de a es $a";
    }

    test();
```

localhost:8080/DWS/clases/T2/clase03.php

(!) Notice: Undefined variable: a in /var/www/html/DWS/clases/T2/clase03.php on line 40				
Call Stack				
#	Time	Memory	Function	Location
1	0.1375	398344	{main}()	.../clase03.php:0
2	0.1375	398344	test()	.../clase03.php:43

El valor de a es

Tenemos varias formas de poder acceder a la variable `$a` del anterior ejemplo.



Podemos usar la palabra reservada **global** para indicarle a la función que la variable `$a` es una variable global.

```
<?php
    $a = 5;

    function test() {
        global $a;
        echo "El valor de a es $a";
    }

    test();
```



El valor de a es 5

Otra forma sería utilizar la variable predefinida **\$GLOBALS** para acceder a `$a`.

```
<?php
    $a = 5;

    function test() {
        echo "El valor de a es " . $GLOBALS["a"];
    }

    test();
```

## Tipo de datos

PHP admite diez tipos primitivos.

Cuatro tipos escalares:

- **boolean**
- **integer**
- **float** (número de punto flotante, también conocido como double)
- **string**

Cuatro tipos compuestos:

- **array**
- **object**
- **callable**
- **iterable**

Y finalmente dos tipos especiales:

- **resource**
- **NULL**

Vamos a comentar algunas particularidades de algunos de los anteriores tipos de datos.

## Booleanos

Este es el tipo más simple. Un *boolean* expresa un valor que indica verdad. Puede ser **TRUE** (verdadero) o **FALSE** (falso).

Se puede convertir explícitamente un valor al tipo *boolean* mediante el forzamiento de tipo **(bool)** o **(boolean)**.

```
<?php
    $var1 = 5;
    echo (bool)$var1;
?>
```

Sin embargo, en la mayoría de casos es innecesario, ya que un valor será convertido automáticamente a *boolean* según las siguiente regla:

- los siguientes valores se consideran **FALSE**:
  - el **boolean FALSE** mismo
  - el **integer 0** y **-0** (cero)
  - el **float 0.0** y **-0.0** (cero)
  - el valor **string vacío**, y el **string "0"**
  - un **array con cero elementos**
  - el tipo especial **NULL** (incluidas variables no establecidas)
  - objetos **SimpleXML** creados desde etiquetas vacías

Todos los demás valores se consideran **TRUE**.

## Números enteros

Los *integer* pueden especificarse mediante notación decimal (base 10), hexadecimal (base 16), octal (base 8) o binaria (base 2), opcionalmente precedidos por un signo (- o +).

Para utilizar la notación octal, se antepone al número un 0 (cero). Para utilizar la notación hexadecimal, se antepone al número 0x. Para utilizar la notación binaria, se antepone al número 0b.

```
<?php
    $num1 = 5; #número decimal
    $num2 = -5; #número negativo
    $num3 = 0123; #número octal
    $num4 = 0x1B3; #número hexadecimal
    $num5 = 0b11010; #número binario
?>
```

**NOTA:** Si mostramos el valor de las variables anteriores en un navegador mediante un **echo**, nos mostrará su valor en decimal.

El tamaño máximo de un entero depende de la plataforma, aunque el valor usual es aproximadamente 2.000 millones.

Para convertir explícitamente un valor al tipo *integer* se puede usar **(int)** o **(integer)**.

```
<?php
    $cadena1 = "5";
    $numero1 = (int)$cadena1;
?>
```

## Número de punto flotante

Los *float* se pueden especificar usando cualquiera de las siguientes sintaxis:

```
<?php
    $a = 1.234;
    $b = 1.2e3;
    $c = 7E-10;
    $d = 1_234.567; // a partir de PHP 7.4.0
?>
```

El tamaño de un *float* depende de la plataforma, aunque un valor común consiste en un máximo de aproximadamente 1.8e308 con una precisión cercana a los 14 dígitos decimales (el formato de 64 bit del IEEE).

Para convertir explícitamente un valor al tipo *float* se puede usar **(float)**.

```
<?php
    $cadena1 = "5.345LOI";
    echo (float)$cadena1; # imprimirá 5.345
?>
```

## Cadena de caracteres

Un *string* es una serie de caracteres donde cada carácter ocupa un *byte*. Por esa razón, PHP sólo admite cadenas de 256 caracteres, y no ofrece soporte nativo para Unicode.

Para especificar un valor de tipo *string* podemos utilizar, como hemos visto antes, las comillas dobles o sencillas. Para escapar caracteres, debemos emplear la barra invertida \. Además, existen otras dos representaciones adicionales: **herodoc** y **nowdoc**.

La sintaxis *herodoc* permite especificar cadenas de caracteres en varias líneas mediante el operador <<<. Después de este operador, se deberá proporcionar un identificador y justo después una nueva línea. A continuación va el propio string, y para cerrar la notación se pone el mismo identificador.

```
<?php
    $cadena = <<<EOS
        Hola, soy una cadena de texto
        que ocupa varias líneas.
        La cadena no terminará
        hasta que aparezca el identificador
        de fin de cadena
    EOS;
    echo $cadena;
?>
```

El texto *heredoc* se comporta como un *string* entre comillas dobles, lo que significa, entre otras cosas, que podemos usar variables dentro del texto sin necesidad concatenar cadenas.

*Nowdoc* es a los string con comillas simples lo mismo que *Heredoc* lo es a los string con comillas dobles. Un *nowdoc* se especifica de forma análoga a un *heredoc*, pero no se realiza ningún análisis dentro del *nowdoc*.

Un *nowdoc* se identifica con la misma secuencia empleada para *heredoc*, <<<, pero **el identificador que le sigue está delimitado con comillas simples**.

```
<?php
    $cadena = <<<'EOS'
        Hola, soy una cadena de texto
        que ocupa varias líneas.
        La cadena no terminará
        hasta que aparezca el identificador
        de fin de cadena
EOS;
    echo $cadena;
?>
```

La principal diferencia entre *herodoc* y *nowdoc* es la misma que entre las comillas dobles y sencillas. Dentro de un texto *herodoc* PHP cambiará el valor de una variable por su correspondiente valor, mientras que con *nowdoc* mostrará el nombre de la variable

```
<?php
    $palabra = 'Mundo'
    $cadena1 = <<<EOS
        Hola $palabra
EOS;
    $cadena2 = <<<'EOS'
        Hola $palabra
EOS;
    echo $cadena1; # imprimirá Hola Mundo
    echo $cadena2; # imprimirá Hola $palabra
?>
```



**NOTA:** Es muy importante señalar que **la línea con el identificador de cierre no debe contener ningún otro carácter, excepto un punto y coma (;)**. Esto, en especial, significa que **el identificador no debe estar sangrado**, y que **no debe existir ningún espacio ni tabulación antes o después del punto y coma**.

PHP proporciona una gran cantidad de variables predefinidas para todos los scripts. Algunos ejemplos son:

- **\$GLOBALS:** Hace referencia a todas las variables disponibles en el ámbito global
- **\$\_SERVER:** Información del entorno del servidor y de ejecución
- **\$\_GET:** Variables HTTP GET
- **\$\_POST:** Variables HTTP POST
- **\$\_FILES:** Variables de subida de ficheros HTTP
- **\$\_REQUEST:** Variables HTTP Request
- **\$\_SESSION:** Variables de sesión
- **\$\_ENV:** Variables de entorno
- **\$\_COOKIE:** Cookies HTTP

A lo largo del curso aprenderemos a utilizar varias de ellas para crear nuestros scripts.

## isset y unset

Para comprobar si una variable **está definida y no es null** podemos utilizar la función **isset(\$var)**, que devolverá **TRUE** si \$var existe y no es nula.

```
<?php
    if (isset($a)) {
        # codigo...
    } else {
        # codigo...
    }
?>
```

Es bastante habitual utilizar **isset** para comprobar si hemos recibido el parámetro por `$_GET` o `$_POST` (por ejemplo a la hora de recibir datos de un formulario).

Si queremos eliminar una variable utilizaremos **unset(\$var)**.

```
<?php
    unset($a)
?>
```


## Constantes

---

Podemos definir **constantas** usando la función **define()** o con la palabra reservada **const**. Si usamos la función *define()* debemos pasar como primer parámetro el nombre de la constante y el valor como segundo.

Antes de PHP 5.6, al emplear la palabra reservada *const*, solamente los datos escalares (*boolean*, *integer*, *float* y *string*) podían estar contenidos en constante. Desde PHP 5.6 en adelante, también es posible definir un array constante.

```
<?php
    define('frase', 'Hola mundo');
    const pi = 3.1416;
?>
```



Igual que con las variables, PHP ofrece un gran número de constantes predefinidas a cualquier script en ejecución. Algunas de ellas son:

- **PHP\_VERSION**, versión de PHP instalada
- **PHP\_EOL**, símbolo de fin de línea de la plataforma en uso
- **PHP\_INT\_MAX**, número entero más grande admitido
- **PHP\_INT\_MIN**, número entero más pequeño admitido
- **E\_ERROR**, constante de informe de error

## Operadores

---

### Operadores aritméticos

<b>\$a + \$b</b>	Suma de \$a y \$b
<b>\$a - \$b</b>	Resta de \$a y \$b
<b>\$a * \$b</b>	Multiplicación de \$a y \$b
<b>\$a / \$b</b>	División de \$a entre \$b
<b>\$a % \$b</b>	Módulo de \$a dividido po \$b
<b>\$a ** \$b</b>	Elevar \$a a la potencia \$bésima

### Operadores de comparación

<b>\$a == \$b</b>	<b>TRUE</b> si \$a es igual a \$b
<b>\$a === \$b</b>	<b>TRUE</b> si \$a es igual a \$b, y son del mismo tipo
<b>\$a != \$b ó \$a &lt;&gt; \$b</b>	<b>TRUE</b> si \$a no es igual a \$b
<b>\$a !== \$b</b>	<b>TRUE</b> si \$a no es igual a \$b, o no son del mismo tipo
<b>\$a &lt; \$b</b>	<b>TRUE</b> si \$a es menor que \$b
<b>\$a &gt; \$b</b>	<b>TRUE</b> si \$a es mayor que \$b
<b>\$a &lt;= \$b</b>	<b>TRUE</b> si \$a es menor o igual que \$b
<b>\$a &gt;= \$b</b>	<b>TRUE</b> si \$a es mayor o igual que \$b

## Operadores para string

<b>\$a . \$b</b>	Concatenación de \$a y \$b
------------------	----------------------------

## Operadores de asignación

<b>\$a = valor</b>	\$a se define como <i>valor</i>
<b>\$a += número</b>	Equivalente a \$a = \$a + número
<b>\$a .= "texto"</b>	Equivalente a \$a = \$a . "Texto"


## Operadores para arrays

<b>\$a + \$b</b>	Unión de \$a y \$b
<b>\$a == \$b</b>	<b>TRUE</b> si \$a y \$b tienen las mismas parejas clave/valor
<b>\$a === \$b</b>	<b>TRUE</b> si \$a y \$b tienen las mismas parejas clave/valor en el mismo orden y de los mismos tipos
<b>\$a != \$b o \$a &lt;&gt; \$b</b>	<b>TRUE</b> si \$a no es igual a \$b
<b>\$a !== \$b</b>	<b>TRUE</b> si \$a no es idéntico a \$b

## Operador ternario

Otro operador condicional es el operador ternario (?).

```
<?php
    $nombre = isset($_POST['nombre']) ? $_POST['nombre'] : 'Pepe';
    echo $nombre;
?>
```



La expresión  $(expr1) ? (expr2) : (expr3)$  evalúa a  $expr2$  si  $expr1$  se evalúa como **TRUE** y a  $expr3$  si  $expr1$  se evalúa como **FALSE**. En el ejemplo anterior si existe  $$_POST['nombre']$ ,  $$nombre$  toma el valor  $$_POST['nombre']$ . Si no, valdrá 'Pepe'. Es equivalente a la siguiente sentencia:

```
<?php
    if (isset($_POST['nombre'])) {
        $nombre = $_POST['nombre'];
    } else {
        $nombre = 'Pepe';
    }
?>
```

A partir de PHP 5.3, es posible dejar de lado la parte media del operador ternario. La expresión  $expr1 ?: expr3$  retorna  $expr1$  si  $expr1$  se evalúa como **TRUE** y  $expr3$  si es de otra manera.

## Operadores nave espacial y fusión de null

A partir de PHP 7 existen dos operadores adicionales de comparación, el operador **nave espacial** y el operador **fusión de null**.

El operador **nave espacial** se emplea para comparar dos expresiones. Devuelve -1, 0 o 1 cuando \$a es respectivamente menor, igual, o mayor que \$b.

```
<?php
// Numeros enteros
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1


// Numeros decimales
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1

// Cadenas de caracteres
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
?>
```

Por su parte, el operador **fusión de null (??)** se ha añadido como aliciente sintáctico para el caso común de la necesidad de utilizar un operador ternario junto con **isset()**. Devuelve su primer operando si existe y no es NULL; de lo contrario devuelve su segundo operando.

```
<?php
    // Obtener el valor de $_GET['usuario'] y devolver 'nadie' si no existe.
    $nombre = $_GET['usuario'] ?? 'nadie';
    // Esto equivale a:
    $nombre = isset($_GET['usuario']) ? $_GET['usuario'] : 'nadie';

    // La fusión se puede encadenar: esto devolverá el primer
    // valor definido de $_GET['usuario'], $_POST['usuario'],
    // y 'nadie'.
    $nombre = $_GET['usuario'] ?? $_POST['usuario'] ?? 'nadie';
?>
```





# Estructuras de control

---

## Condicionales

### if

El condicional simple, se escribe con la expresión **if**. Podemos añadir un bloques **else** al condional.

```
<?php
    if ($a < 5) {
        # codigo...
    } else {
        # codigo...
    }
?>
```

También podemos usar **elseif** para añadir más condiciones:

```
<?php
    if ($a < 5) {
        # codigo...
    } elseif ($a < 10){
        # codigo...
    } else {
        # codigo...
    }
?>
```

## switch

Para el condicional múltiple, usaremos la expresión **switch**.

```
<?php
    switch ($a) {
        case 0:
            # codigo...
            break;
        case 1:
            # codigo...
            break;
        default:
            # codigo...
            break;
    }
?>
```

La lista de sentencias para un caso también puede estar vacías, lo que pasará el control a la lista de sentencias para el siguiente caso.

```
<?php
    switch ($a) {
        case 0:
        case 1:
        case 2:
            # codigo1...
            break;
        case 3:
            # codigo2...
            break;
        default:
            # codigo3...
            break;
    }
?>
```

En este caso, tanto si `$a` vale 0, 1 o 2 se ejecutará `codigo1`.

No hay que olvidarse de añadir **break** al final de cada opción. En caso contrario, PHP seguiría ejecutando las sentencias del caso siguiente.

Para añadir la opción por defecto, utilizaremos **default**.

## Bucles

### while

Los bucles **while** son el tipo de bucle más sencillo en PHP. Se comportan igual que su contrapartida en C. La forma básica de una sentencia **while** es:

```
<?php
    while ($a <= 10) {
        # codigo...
    }
?>
```

### do-while

Los bucles **do-while** son muy similares a los bucles **while**, excepto que la condición es verificada al final de cada iteración, en lugar de al principio. La principal diferencia es que en un bucle **do-while** se ejecutará la primera iteración siempre, mientras que un bucle **while** no tiene porque.

```
<?php
    do {
        # codigo...
    } while ($a <= 10)
?>
```

## for

Los bucles **for** son los más complejos en PHP. Se comportan como sus homólogos en C. La sintaxis de un bucle **for** es:

```
<?php
    for ($i=0; $i < 5; $i++) {
        # codigo...
    }
?>
```

PHP ofrece la posibilidad de usar expresiones vacías en los bucles **for**, lo que puede resultar útil en alguna ocasión.

```
<?php
    for ($i=0; ; $i++) {
        if ($i < 5) {
            break;
        }
        echo $i;
    }
?>
```

Para salir de cualquier bucle, se puede utilizar **break**, mientras que para continuar con la siguiente iteración podemos usar **continue**.



## foreach

El constructor **foreach** proporciona un modo sencillo de iterar sobre arrays. **foreach** funciona sólo sobre arrays y objetos, y emitirá un error al intentar usarlo con una variable de un tipo diferente de datos o una variable no inicializada. Existen dos sintaxis:

```
<?php
    foreach ($array1 as $value) {
        # codigo...
    }

    foreach ($array1 as $key => $value) {
        # codigo...
    }
?>
```

La primera forma recorre el array `$array1`. En cada iteración el valor del elemento actual se asigna a `$value` y el puntero interno del array avanza una posición.

La segunda forma además asigna la clave del elemento actual a la variable `$key` en cada iteración.

## Incluir archivos externos

---

En PHP podemos incluir varios archivos externos en nuestros scripts. Estos archivos pueden ser otros PHPs, HTMLs... Esta práctica es habitual a la hora de estructurar la programación y así poder reutilizar el código en diferentes scripts de nuestras webs.

Para incluir archivos externos podemos usar dos sentencias: **include** o **require**.

### Include y require

La sentencia **include** incluye y evalúa el archivo especificado.

Si el archivo no se encuentra en la ruta dada, **include** emitirá una advertencia. Si no se da ninguna ruta, PHP buscará primero en el *include\_path* especificado en el archivo *php.ini*, después en el propio directorio del script que hace la llamada y en el directorio de trabajo actual antes de fallar.

Usar la sentencia **include** equivale a cortar y pegar el contenido del archivo incluido en nuestro script.

Vamos a ver un ejemplo de como incluir un archivo dentro de otro.

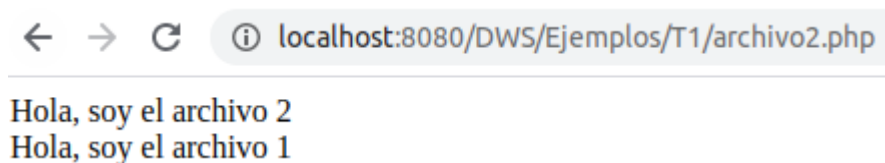
Crea un archivo llamado *archivo1.php* en el directorio *htdocs\_apache/DWS/Ejemplos/T1/Ej1* con el siguiente contenido:

```
<?php
    echo "Hola, soy el archivo 1";
?>
```

Crea otro archivo llamado *archivo2.php* en el mismo directorio y añade el siguiente código:

```
<?php
    echo "Hola, soy el archivo 2";
    include "archivo1.php";
?>
```

Ahora abre en el navegador <http://localhost/DWS/Ejemplos/T1/archivo2.php> y comprueba la salida.



Como ves, la sentencia **include** es muy útil para separar y estructurar nuestro código.



Una cosa a tener en cuenta es que si incluimos dos veces el mismo archivo PHP no comprobará si ya está incluido, y lo volverá a incluir.

Para que no vuelva a incluir un fichero ya incluido podemos usar la sentencia **include\_once**. Tiene un comportamiento similar al de la sentencia **include**, con la diferencia que si el código del fichero ya se ha incluido, no se volverá a incluir, e **include\_once** devolverá **TRUE**.

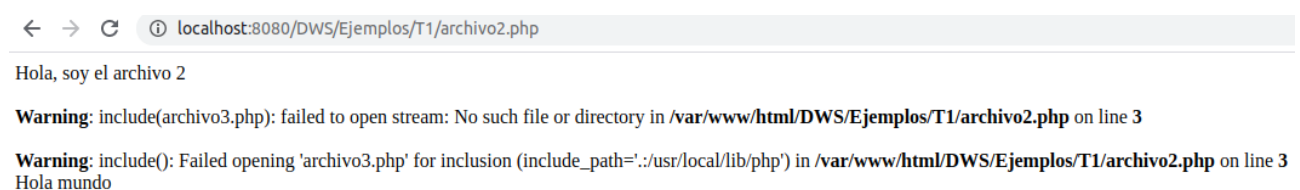


La sentencia **require** es exactamente igual que **include**, excepto que en caso de fallo producirá un error fatal de nivel **E\_COMPILE\_ERROR**. En otras palabras, éste detiene el script, mientras que **include** sólo emitirá una advertencia, lo cual permite continuar el script.

Para ver la diferencia, modifica *archivo2.html* y cambia la ruta de **include** por algún archivo que no exista. A continuación saca por pantalla la frase “Hola mundo”.

```
<?php
    echo "Hola, soy el archivo 2";
    include "archivo3.php";
    echo "Hola mundo";
?>
```

Si vemos la salida en el navegador observamos que aunque nos de una advertencia, el script sigue ejecutándose mostrando la frase “Hola mundo” al final.



← → ↻ ⓘ localhost:8080/DWS/Ejemplos/T1/archivo2.php

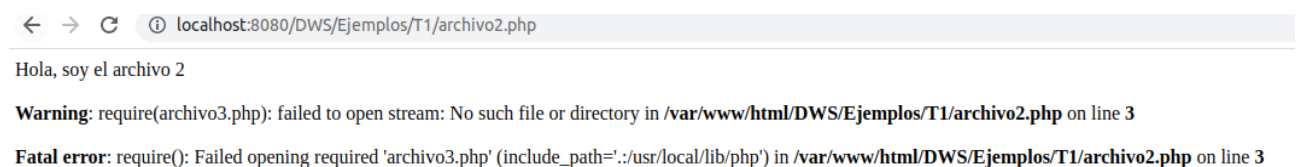
Hola, soy el archivo 2

**Warning:** include(archivo3.php): failed to open stream: No such file or directory in /var/www/html/DWS/Ejemplos/T1/archivo2.php on line 3

**Warning:** include(): Failed opening 'archivo3.php' for inclusion (include\_path=.:usr/local/lib/php) in /var/www/html/DWS/Ejemplos/T1/archivo2.php on line 3

Hola mundo

Ahora cambia **include** por **require** y comprueba la salida.



← → ↻ ⓘ localhost:8080/DWS/Ejemplos/T1/archivo2.php

Hola, soy el archivo 2

**Warning:** require(archivo3.php): failed to open stream: No such file or directory in /var/www/html/DWS/Ejemplos/T1/archivo2.php on line 3

**Fatal error:** require(): Failed opening required 'archivo3.php' (include\_path=.:usr/local/lib/php) in /var/www/html/DWS/Ejemplos/T1/archivo2.php on line 3

En esta ocasión, el intérprete PHP nos advierte del fallo y además nos da un **Fatal error**. Además, ya no saca la frase “Hola mundo”, ya que en cuanto genera el error detiene el script.

¿Cuándo utilizar **include** y cuando **require**? Cuando el código a insertar sea imprescindible para el funcionamiento de nuestro script, debemos utilizar **require**. En caso contrario, usaremos **include** para que nuestra web pueda seguir funcionando.

**NOTA:** Puede que veas alguna vez usar **include** o **require** como si fueran funciones: `require("archivo3.php");`

Aunque esa sintaxis funcione sin problemas, **include** o **require** no son funciones, si no sentencias que forman parte de la estructura de control, por eso se aconseja utilizar la primera forma.

Igual que **include\_once**, también existe **require\_once**, que verificará si el archivo ya se ha incluido.

**NOTA:** Tenemos que tener en cuenta que tanto **include\_once** como **require\_once** requieren más tiempo de procesamiento, ya que el analizador tiene que comprobar si los archivos ya están incluidos.

En cualquier caso, muchas veces nos interesa usar **include\_once** o **require\_once** para evitar redefinición de funciones, reasignación de valores a las variables, redefinición de clases...

## Excepciones

---

PHP tiene un modelo de excepciones similar al de otros lenguajes de programación. Se puede lanzar una excepción (**throw**), y atraparla (**catch**). El código puede estar dentro de un bloque **try** para facilitar la captura de excepciones potenciales. Cada bloque **try** debe tener al menos un bloque **catch** o **finally** correspondiente.

### **catch**

Se pueden usar múltiples bloques **catch** para atrapar diferentes clases de excepciones. La ejecución normal (cuando no es lanzada ninguna excepción dentro del bloque **try**) continuará después del último bloque **catch** definido en la secuencia. Las excepciones pueden ser lanzadas (**throw**) dentro de un bloque **catch**.

Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque **catch** coincidente. Si una excepción no es capturada, se emitirá un **Fatal Error** de PHP con un mensaje “**Uncaught Exception...**” a menos que se haya definido un manejador con **set\_exception\_handler()**.

## finally

En PHP, se puede utilizar un bloque **finally** después o en lugar de bloques **catch**. El código de dentro del bloque **finally** siempre se ejecutará después de los bloques **try** y **catch**, independientemente de que se haya lanzado una excepción o no, y antes de que la ejecución normal continúe.

```
<?php
    try {
        //codigo...
    } catch (Exception $e) {
        //codigo...
    } finally {
        //codigo...
    }
?>
```

## Bibliografía

---

<https://www.php.net/manual/es/index.php>