# Tutorial básico de

# PHP

# 1 Introducción

PHP es un lenguaje de programación de código abierto del lado del servidor que se utiliza principalmente para **crear páginas web dinámicas**. La abreviatura nació originariamente de "Personal Home Page Tools", aunque hoy en día se ha convertido en el acrónimo recursivo para "**PHP:HypertextPreprocessor**".

Mientras que los lenguajes del lado del cliente como HTML, CSS o JavaScript son interpretados primero por el navegador web en el momento de abrir una página, el código PHP **se ejecuta en el servidor web**. Allí, los scripts de PHP generan el código HTML que se envía después al navegador. Este no recibe el código real (el script de PHP), sino el resultado de la ejecución del mismo.

El ámbito de aplicación principal de PHP es la **programación del lado del servidor,** sobre todo de páginas dinámicas y aplicaciones. Otras áreas de implementación son la creación de aplicaciones de escritorio o la programación de líneas de comandos. A pesar de tener una sintaxis sencilla para principiantes, PHP ofrece una cantidad remarcable de funciones. Este lenguaje de programación se distingue por su amplio soporte a bases de datos, puede utilizarse en todo tipo de plataformas y está cubierto por una **licencia PHPespecial** que permite su libre utilización y modificación del código fuente, una combinación más que convincente.

Cabe destacar que cuatro de los **sistemas de gestión de contenidos** más populares, es decir, WordPress, TYPO3, Joomla y Drupal, se basan en PHP. Un análisis de mercado de W3Techs establece que este lenguaje de scripts se emplea en el 82,4por ciento de todas las páginas de la World Wide Web (datos del 01 de febrero de 2017), lo que indica que PHP es, con diferencia, el **lenguaje de programación del lado del servidor más popular** en el marco del desarrollo web. Esto se convierte en motivo suficiente para que cada vez más usuarios se familiaricen con las posibilidades de PHP: HypertextPreprocessor.

La manera más rápida de **aprender PHP** es entendiendo los ejemplos y adaptándolos a las necesidades de cada proyecto web. Todo lo que se necesita para la programación con PHP es un **servidor web con un intérprete de PHP**, un **editor de textos** (por ejemplo, Notepad++ o Vim) y un **navegador web**. Como servidor para una primera inclusión es recomendable utilizar el **entorno de prueba local XAMPP,** puesto a disposición por Apache Friends para los sistemas operativos Windows, Linux y macOS de forma gratuita.

El lenguaje PHP es un lenguajeinterpretado con una sintaxis similar a la de C++ o JAVA. Aunque el lenguaje se puede usar para realizar cualquier tipo de programa, es en la generación dinámica de páginas web donde ha alcanzado su máxima popularidad. En concreto, suele incluirse incrustado en páginas HTML (o XHTML), siendo el servidor web el encargado de ejecutarlo.

Algunas de las características de su enorme popularidad son:

- Es un lenguaje libre. Puede descargarse de http://www.php.net.
- Está disponible para muchos sistemas (GNU/Linux, Windows, UNIX, etc).
- Tiene una extensa documentación oficial en varios idiomas (disponible libremente en http://www.php.net).
- Existen multitud de extensiones: para conectar con bases de datos, para manejo de sockets,para generar documentos PDF, para generar dinámicamente páginas en Flash, etc.

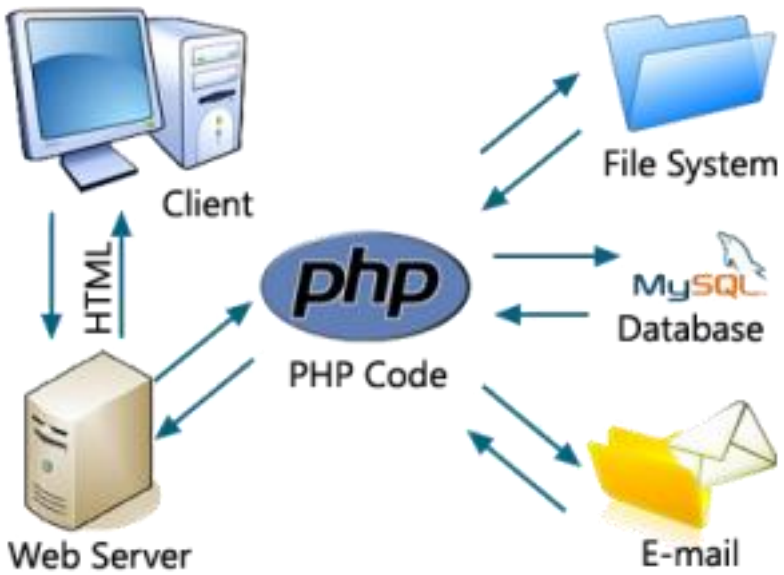
- Al ejecutarse en el servidor, los programas PHP lo pueden usar todo tipo de máquinas con todo tipo de sistemas operativos.
- En caso de que un cliente falle (por error hardware, virus, etc) se puede seguir usando el sistema desde otro cualquiera que tenga un navegador web con conexión al servidor.

Para realizar programas en PHP son necesarios algunas nociones de HTML (o XHTML), pero se puede ir aprendiendo sobre la marcha con los ejemplos.

## 2 Inclusión de código PHP en una página HTML

Para incluir código PHP basta con precederlo de la etiqueta <?php, y cerrarlo con ?>. Si el servidor web está correctamente configurado, detectará código PHP y, en vez de proporcionarle el contenidode la página directamente al cliente (lo que significaría que recibiría el código fuente del programa), ejecuta el programa y devuelve su resultado al navegador.

Así pues, el esquema de una petición sería como sigue:



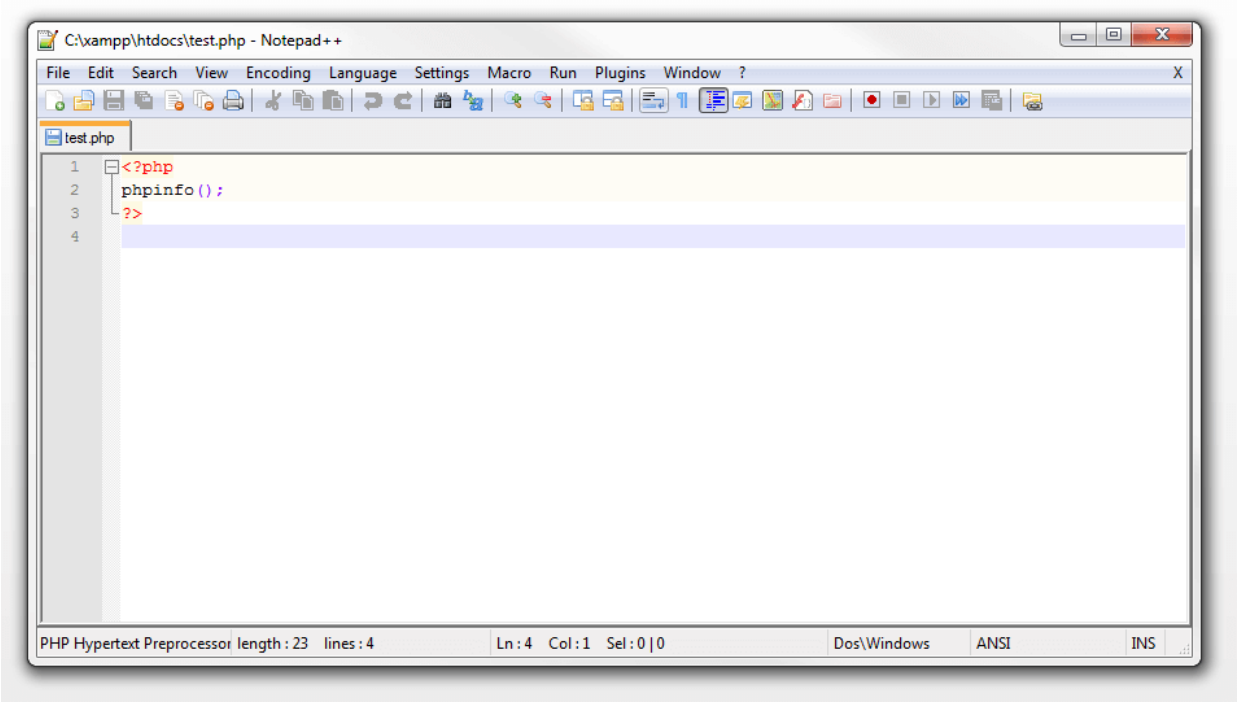
# 4. La sintaxis del lenguaje de scripts

Una vez hayas configurado tu servidor web local (por ejemplo, con ayuda de XAMPP), tienes que comprobar que **PHP esté instalado correctamente** y pueda ejecutar scripts.

Un **script** es un programa informático en general de pequeñas dimensiones que no se compila en código binario previamente. Los scripts se escriben en un lenguaje de programación como *PHP*, *Perl* o *JavaScript* y son ejecutados por un intérprete en el servidor web (del lado del servidor) o por un motor de renderizado en el navegador web (del lado del cliente).

Abre tu **editor de textos** preferido e introduce el siguiente script de PHP:

```
<?php
phpinfo();
?>
```




Un editor de textos como Notepad++ ayuda a programar poniendo de relieve la sintaxis.

Los scripts de PHP siempre se basan en el mismo esquema. La **etiqueta PHP de apertura** *<?php* señaliza que se va a iniciar un **entorno de scripts**. A esto le sigue el propio código PHP en forma de órdenes o instrucciones. En el ejemplo se trata de la llamada a la **función***phpinfo()*.La mayoría de funciones requieren uno o varios parámetros situados entre paréntesis. En el caso de *phpinfo(),* estos son opcionales: *phpinfo( INFO_ALL )*. Cada función termina con un **punto y coma** (*;*) y para cerrar el script entra en juego la **etiqueta PHP de cierre**, es decir, *?>*.

Las **funciones** son subprogramas que permiten externalizar partes del código de programa. Para evitar la redundancia, se definen las tareas recurrentes como funciones y se hace un llamamiento a las mismas con ayuda de un nombre de función. Los desarrolladores utilizan para ello funciones de PHP predefinidaso crean sus propios subprogramas.

Guarda el archivo de texto con el nombre *test* en formato *.php* (script PHP) y abre el servidor web. Siempre y cuando utilices el entorno de prueba de XAMPP, coloca *test.php* en el directorio de XAMPP en ***htdocs*** (*C:\xampp\htdocs*).

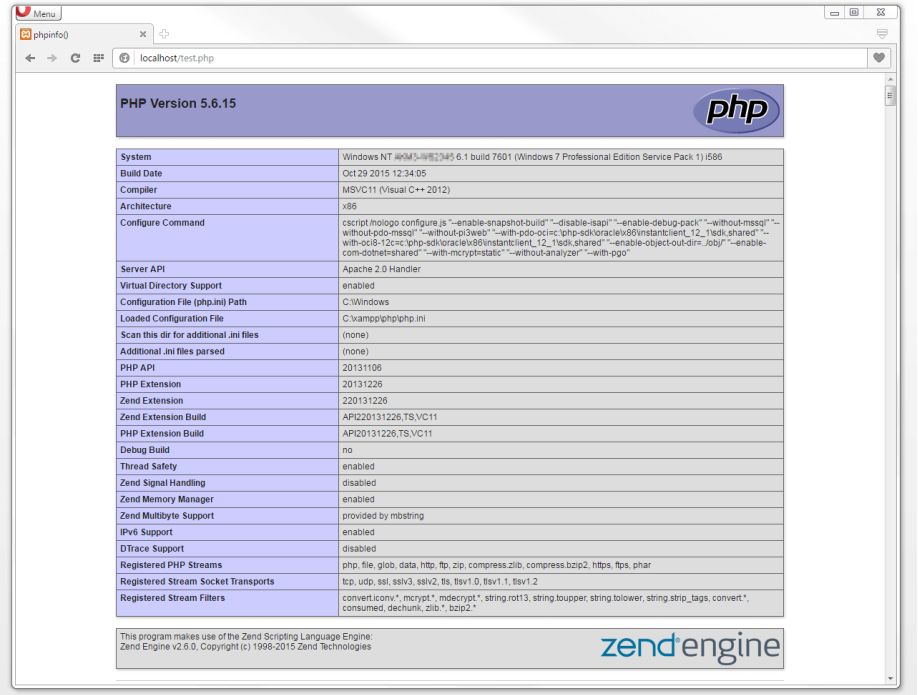
Se puede acceder al archivo del ejemplo introduciendo el siguiente URL en el navegador web: *http:// localhost/test.php*. En caso de utilizar otro servidor web o la configuración personalizada del software de XAMPP, **selecciona el URL de la ruta del archivo correspondiente**.

**Atención**

Debes tener en cuenta que mientras que en las direcciones de Internet se utiliza la barra diagonal (/) como símbolo delimitador de directorios, el explorador de Windows utiliza la barra invertida (\). Los navegadores web modernos, sin embargo, transforman la barra invertida en una diagonal en la barra de navegación de forma automática.

Al introducir el URL *http:// localhost/ test.php* se está indicando al navegador web que solicite el archivo *test.php* al servidor web. El servidor Apache HTTP u otro software de servidor web abre el archivo en el directorio correspondiente. La terminación *.php* informa de que el archivo contiene código PHP. Ahora se pone en marcha el intérprete de PHP integrado en el servidor web, el cual hace un recorrido por el documento hasta dar con la etiqueta PHP de apertura *<?php,* la cual señala el comienzo del código PHP. Tras ello, el intérprete ya tiene la capacidad de ejecutar el código PHP y de generar una salida en HTML que se envía al navegador desde el servidor web.

Si PHP se ha instalado correctamente, aparecerá la página web siguiente como **resultado de la ejecución del script**:




Si se ejecuta un script con la función phpinfo(), el navegador mostrará información sobre la configuración de PHP

La función *phpinfo()* consiste en la abreviatura del valor estándar *phpinfo( INFO_ALL )*, el cual ofrece información pormenorizada sobre la configuración de PHP en el servidor web. Si no se puede localizar ninguna versión de PHP, el navegador mostrará un mensaje de error o entregará el código PHP sin interpretar al navegador.

## "HelloWorld!" – Así puedes crear texto con echo

Si la instalación de PHP se lleva a cabo sin ningún tipo de errores, ya se pueden escribir los primeros scripts propios. Para ello es apropiada la instrucción *echo*. A diferencia de *phpinfo(),echo* no representa una función, sino que se trata, más bien, de una **construcción del lenguaje** que permite distribuir el siguiente string como texto.

**Definición**

Las **palabras reservadas** son instrucciones que se utilizan en PHP para controlar la ejecución de los programas. Entre las palabras reservadas se encuentran, además de *echo,* instrucciones como *if, for, do, include, return, exit* o *die*. A diferencia de las funciones, en este caso no se necesitan los paréntesis.

Crea un nuevo archivo PHP para tu primer script propio y escribe el siguiente código:

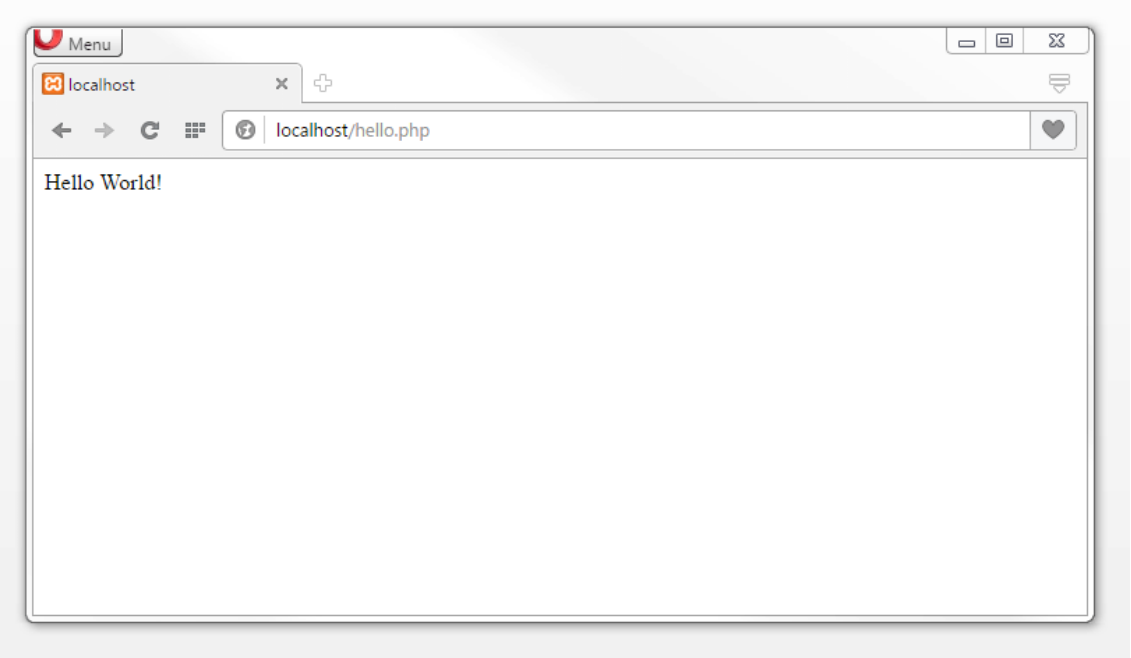
```
<?php

echo 'Hello World!';

?>
```

La etiqueta de apertura *<?php* da comienzo a un entorno de script. A esta le sigue tanto la palabra reservada ***echo*** como el string *HelloWorld!* **entre comillas** simples. Con la etiqueta *?>* se cierra el script. En este sentido hay que prestar atención al punto y coma que va detrás de la instrucción. En lugar de *HelloWorld!* se puede utilizar cualquier otro texto.

**Definición**

En informática se entiende por **string** una secuencia de caracteres de longitud variable o, en otras palabras, una cadena de consonantes, números y símbolos. En el ámbito de la programación, los strings están considerados como tipos de datos independientes y guardan diferencias con otros tipos de datos como los *integers* (números enteros) o los *floats* (números de punto flotante).

Guarda el script con el nombre de *hello.php* en la carpeta *htdocs* de tu servidor web y accede al archivo a través del URL *localhost/hello.php* en el navegador. Si el código se ha transmitido correctamente, ahora debería mostrarse en la ventana del navegador la secuencia de caracteres que has usado:

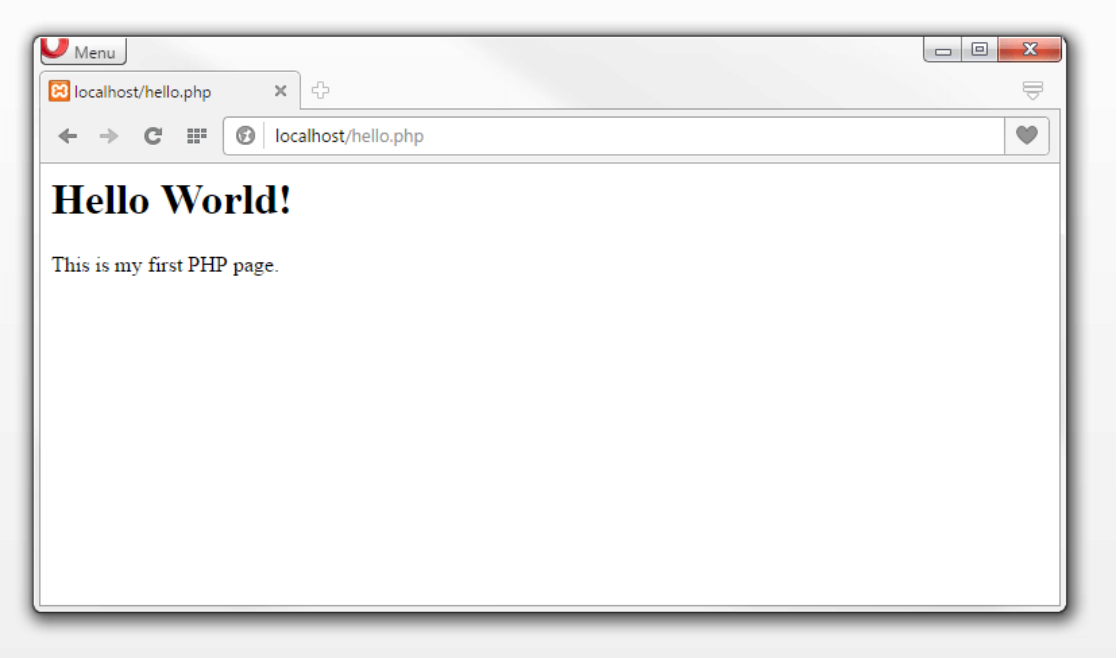



La instrucción del lenguaje **echo** instruye al servidor web para que emita la secuencia de caracteres HelloWorld!

Cada texto que se emite con *echo* puede estar formado por **etiquetas HTML** en caso de ser necesario, las cuales son interpretadas por parte del navegador web tras la correspondiente especificación de HTML. Puedes comprobarlo tú mismo, por ejemplo, con el siguiente script:

```
<?php
echo '<h1>Hello World!</h1>
<p>This is my first PHP page.</p>';
?>
```

Al hacer la solicitud al navegador web, se puede ver el **resultado de la ejecución del script** de la siguiente manera:




Si el resultado de la ejecución de script incluye etiquetas HTML, el navegador las interpreta automáticamente

La secuencia de caracteres incluida en las etiquetas del *<h1>*, es decir, *HelloWorld!*, es interpretada por el navegador como título de primer nivel y a esta le siguen un salto de línea automático y la etiqueta de párrafo *<p>*.

En función de las necesidades, *echo* se puede emplear tanto con **comillas sencillas(')** como con **comillas dobles(")**. Si se quiere emitir texto, no hay diferencia alguna con respecto al tipo de comillas que se escojan, pero esto cambia en cuanto las **variables** entran en acción.

## Variables

La palabra reservada *echo* resulta de una utilidad mayor que la propia emisión de texto, la cual puede implementarse también sin PHP y tomando HTML como base. La verdadera plusvalía de *echo* está basada en el hecho de que la instrucción permite generar **textos de manera dinámica con ayuda de variables**.

Los usuarios de PHP se pueden encontrar con variables que tienen, por ejemplo, la siguiente forma:

*$ejemplo*

Cada variable consta del **símbolo del dólar($),** seguido del **nombre de la variable**. Las variables se utilizan en los scripts de PHP para integrar datos externos en páginas web. En este sentido se puede hablar de valores muy variados que van desde números simples y cadenas de caracteres hasta textos completos o estructuras de documentos HTML.

PHP diferencia entre siete tipos de variables:

| Tipos de variables | Descripción |
|---|---|
| String | Un string es una secuencia de caracteres, que puede tratarse de una palabra, de una frase, de un texto o de la totalidad del código HTML de una página web. |
| Integer | Un integer es un número entero y sin decimales que puede ser positivo o negativo. |
| Float | Un float es un número de punto flotante, es decir, un valor numérico con decimales. En los lenguajes de programación, la coma se escribe con un punto (.). PHP permite colocar hasta 14 caracteres detrás de la coma. |
| Boolean | Las variables booleanas son el resultado de una operación lógica y solo comprenden dos constantes: TRUE (verdadero) y FALSE (falso). Este tipo de variables se aplica cuando se trabaja con condiciones. |
| Array | Un array es una variable que puede albergar varios elementos. Se trata de una agrupación de diversos datos estructurados formando una matriz. |
| Object | La variable object permite a los programadores definir tipos de datos propios y se aplica en la programación orientada a objetos. Las variables del tipo object no se incluyen en nuestro tutorial de PHP. |
| NULL | NULL representa una variable sin valor. Para las variables del tipo NULL, este es el único valor. |

El tipo de una variable no se suele especificar. Se decide en tiempo de ejecución en función del contexto y puede variar.

Algunas funciones de interés:

- La función **gettype**() devuelve el tipo de una variable.
- Las funciones is_type comprueban si una variable es de un tipo dado: **is_array**(), **is_bool**(), **is_float**(), **is_integer**(), **is_null**(), **is_numeric**(), **is_object**(), **is_resource**(), **is_scalar**(), **is_string**().
- La función **var_dump**() muestra el tipo y el valor de una variable. Es especialmente interesante con los arrays.

La administración central de los contenidos tiene lugar, en general, con ayuda de **sistemas de bases de datos**. Los valores para las variables pueden definirse directamente en el script. Este tipo de clasificación se realiza según el esquema siguiente:

*$ejemplo = "Valor";*

El característico símbolo del dólar va seguido del nombre de la variable (en este caso *ejemplo*), el cual se une al valor entrecomillado con el **símbolo de igualdad(=)**.

**Atención**

Los valores para las variables del tipo integer y float no se escriben entre comillas (p. ej., *$ejemplo = 24;* o *$ejemplo = 2.7;*)

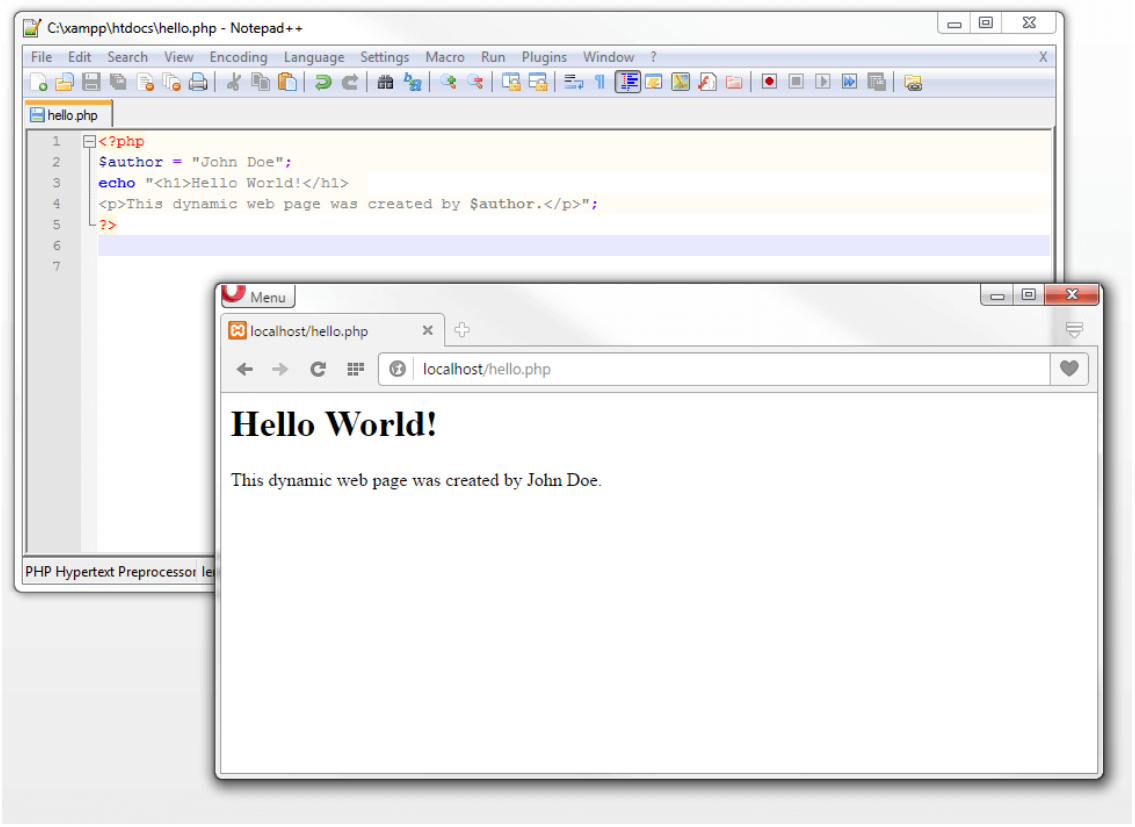
PHP te da la libertad de designar variables según tu arbitrio, pero surgen aquí ciertas limitaciones:

- Cada variable comienza con el símbolo del dólar.
- El nombre de las variables es una secuencia de caracteres formada por letras, números y guiones (p.ej., *$ejemplo_1*).
- Un nombre de variable válido siempre comienza con una letra o con un guion bajo (*$ejemplo1* o *$_ejemplo*), pero nunca con un número (incorrecto: *$1ejemplo*).
- PHP distingue entre mayúsculas y minúsculas *($ejemplo ≠ $Ejemplo*).
- Los nombres de las variables no pueden contener espacios o saltos de línea (incorrecto: *$ejemplo 1*)
- El usuario no puede hacer uso libre de las secuencias de caracteres reservadas por PHP para otros propósitos (p. ej., *$this*)
- Variables predefinidas: $GLOBALS, $_SERVER, $_GET, $_POST, $_COOKIES, $_FILES, $_ENV, $_REQUEST, $_SESSION.
- Ámbito: globales al fichero (excepto funciones) o locales a una función.

Veámoslo en un ejemplo:

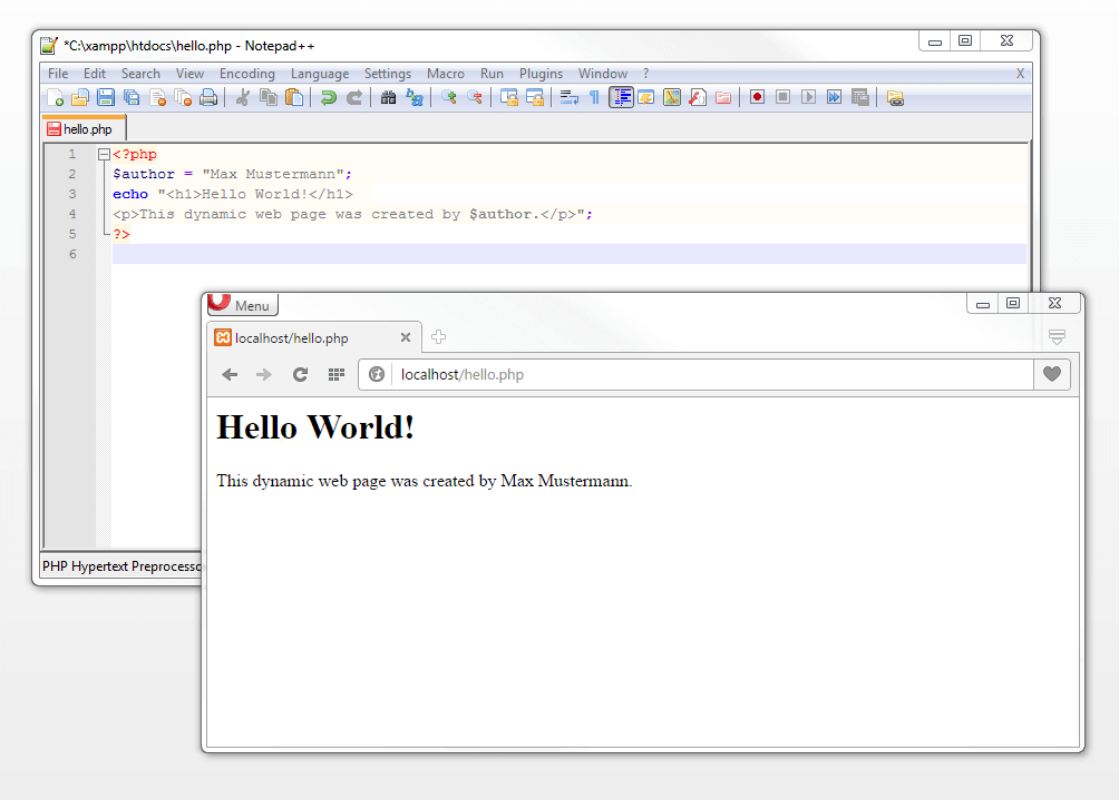
```
<?php
$author = "John Doe";
echo "<h1>Hello World!</h1>
<p>This dynamic web page was created by $author.</p>";
?>
```

La etiqueta de apertura de PHP va seguida de la **definición de la variable**: en el caso de *$author*se utilizaría el valor *John Doe*. A la hora de ejecutar el script, la variable *$author* se sustituye por el valor *John Doe* cada vez que se haga mención a ella en el entorno del script. El siguiente gráfico muestra cómo se refleja esto en el navegador web.




En el resultado de la ejecución del script, a la variable $author se le asigna el valor John Doe

Si ocurre un error y la página web no procede de John Doe, sino de su colega alemán Max Mustermann, la variable con el nombre *$author* tiene que adaptarse para subsanar el fallo.




Para la variable $author se utiliza el valor Max Mustermann

Esto resulta especialmente eficiente cuando una variable aparece varias veces en un script. En este caso solo se tiene que corregir una parte de la misma, es decir, aquella en donde se haya definido el valor de la variable.

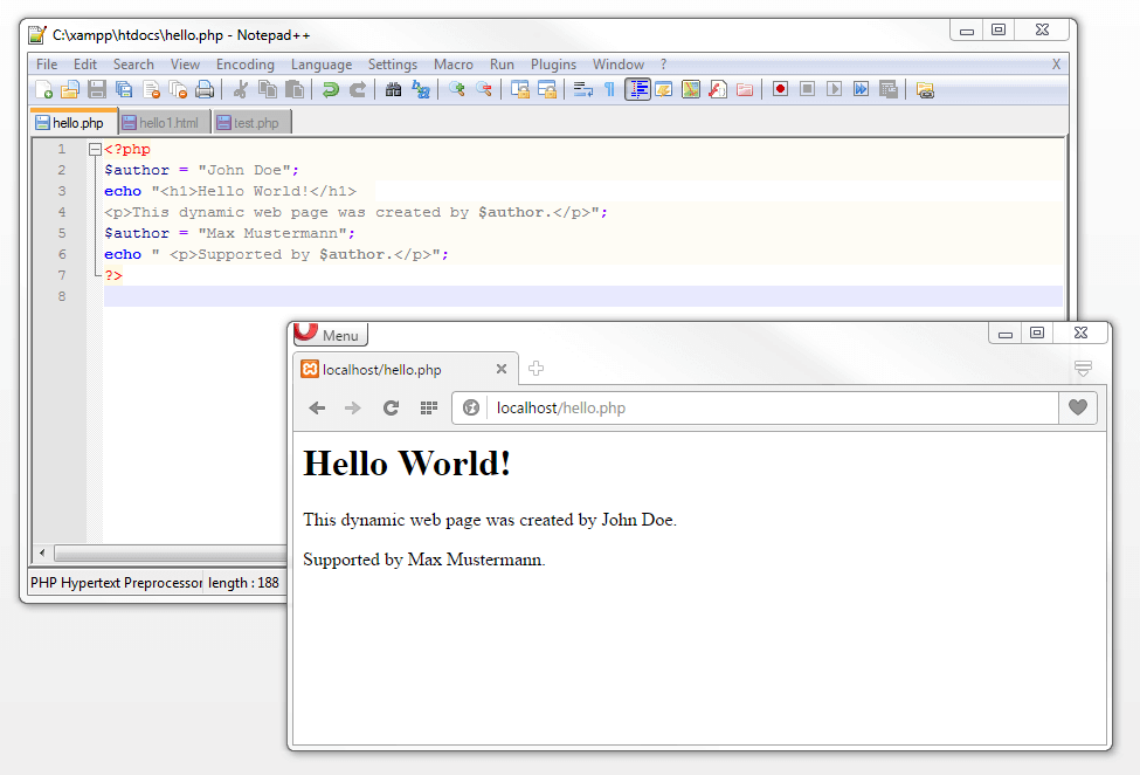
Aquí queda patente el punto fuerte de PHP: **los contenidos pueden integrarse como variables**, lo que se constituye como la base del desarrollo web dinámico. A diferencia de las páginas web estáticas, que se presentan como páginas HTML extraditables, las **páginas web dinámicas** se generan en el momento en que se abren. Para ello, el intérprete de PHP descarga cada uno de los elementos de la página web solicitada con ayuda de las variables de las diferentes bases de datos y los integra en una página HTML ajustada a la solicitud.

Las ventajas de este concepto de diseño son evidentes: si se corrigen los elementos de la página web (p. ej., en el pie de página), no es necesario realizar los ajustes de forma manual en cada una de las subpáginas del proyecto web. En lugar de eso, es suficiente con **actualizar la entrada en la base de datos.** De esta manera se asume la revisión de manera automática para todas las páginas web que enlazan los datos correspondientes como variables.

Si **una variable se define varias veces en un script**, la nueva definición sobreescribe a la anterior. El siguiente *echo* siempre ofrece el valor actual de una variable.

```
<?php
$author = "John Doe";
echo "<h1>Hello World!</h1>
<p>This dynamic web page was created by $author.</p>";
$author = "Max Mustermann";
echo " <p>Supported by $author.</p>";
```

```
?>
```




El valor John Doe se sobreescribe con el valor Max Mustermann

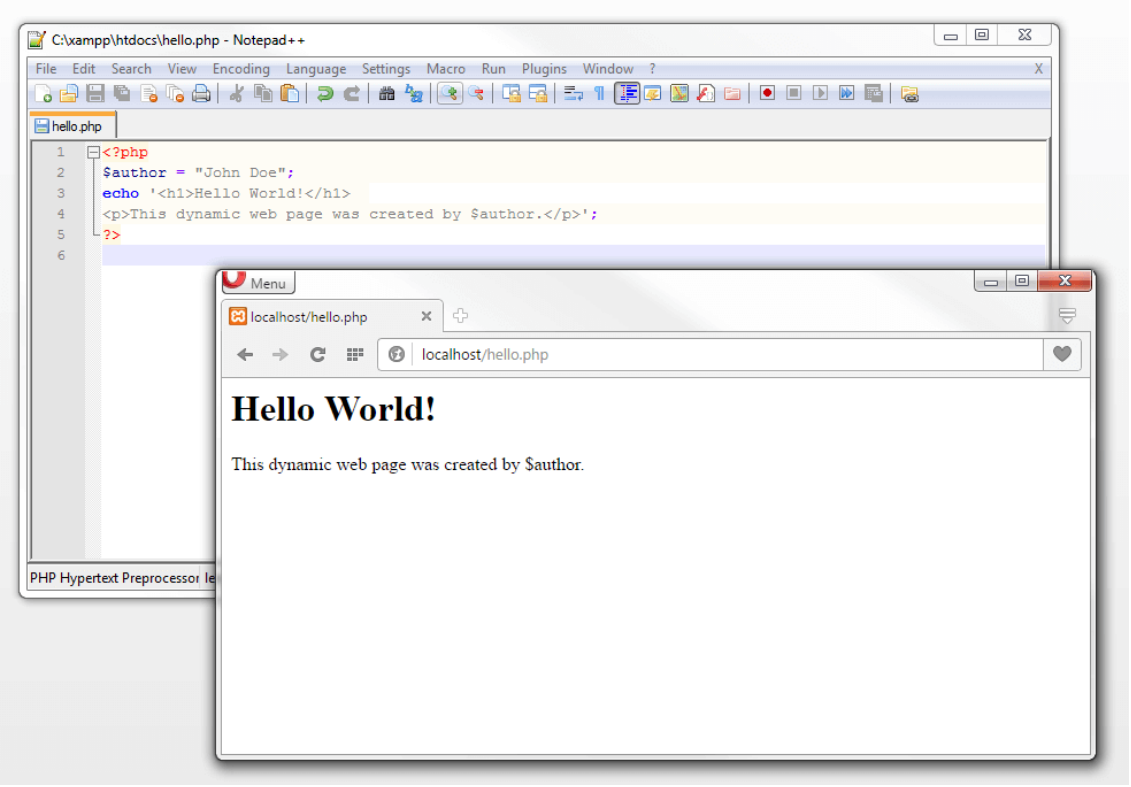
En el ejemplo de código se otorga el valor *John Doe* a la variable *$author* y, posteriormente, se sustituye por el valor *Max Mustermann*.

En lo que respecta al **uso de las comillas**, a diferencia de los strings, no es necesario que las variables se introduzcan entrecomilladas:

```
<?php
$author = "John Doe";
echo $author;
?>
```

A no ser que la variable se tenga que utilizar dentro de un string, en cuyo caso habrá que recurrir a las comillas dobles (”). Estas indican al intérprete de PHP que tiene que buscar variables en el string para sustituirlas con los valores asociados a ellas. Las secuencias de caracteres expresadas entre comillas simples (‘) serán interpretadas y reproducidas como información puramente textual incluso cuando son variables. Lo puedes comprobar tú mismo con la siguiente secuencia de código:

```
<?php
$author = "John Doe";
echo '<h1>Hello World!</h1>
<p>This dynamic web page was created by $author.</p>';
?>
```




Las comillas simples dan lugar a texto sin formato. Las variables no se interpretan

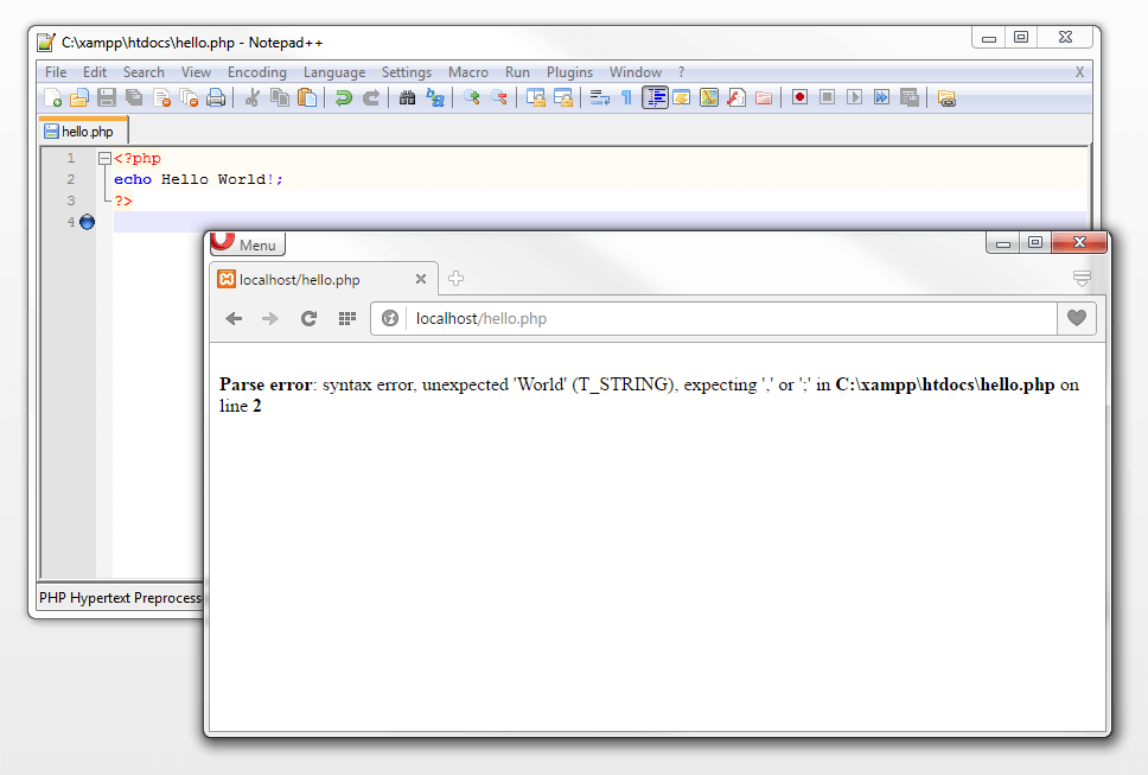
Puede que a este respecto surja la pregunta de lo que pasaría si se eliminaran las comillas completamente. En este caso PHP informaría de un error de sintaxis.

## Mensajes de error y enmarascaramiento

En caso de que haya **errores sintácticos**, el código PHP no es válido y el intérprete de PHP emite un mensaje de error. También puede ocurrir lo mismo cuando, por ejemplo, se utiliza la instrucción *echo* con un string sin comillas:

```
<?php
echo HelloWorld!;
?>
```

Los mensajes de error contienen, en la mayoría de los casos, información acerca de la ubicación de los errores, ofreciendo, así, datos importantes para su erradicación.

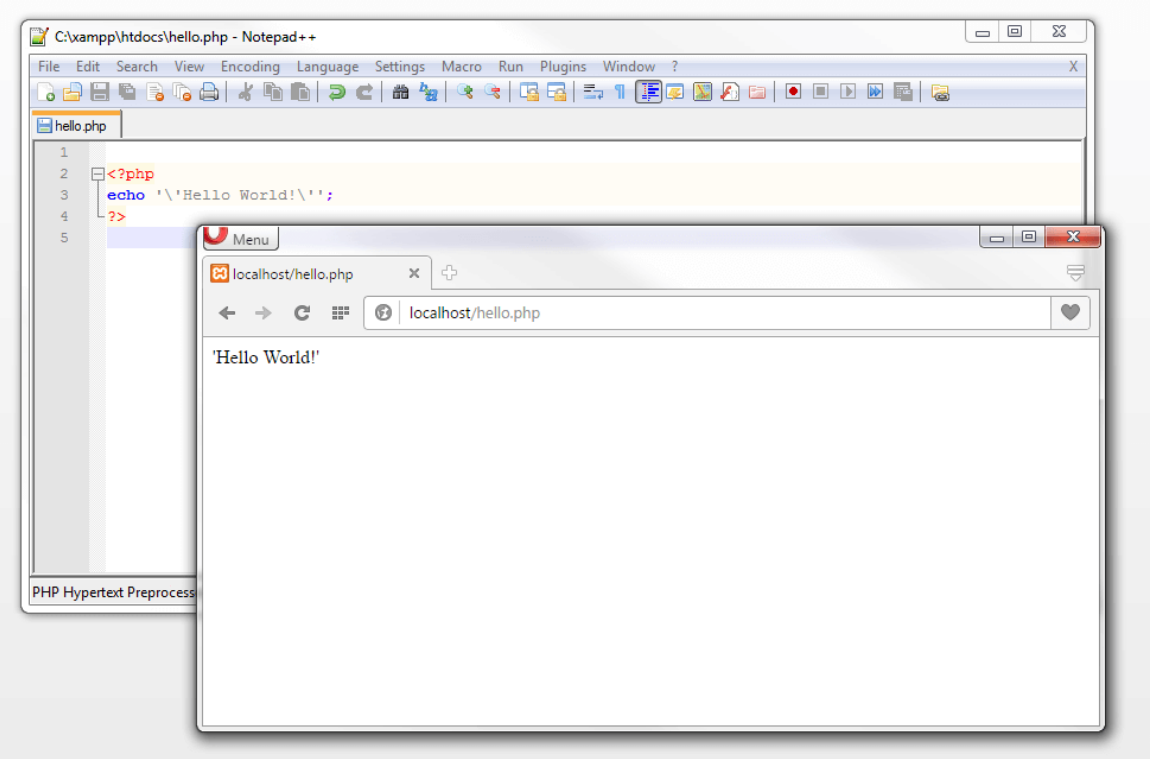



Un mensaje de error informa de la presencia de errores sintácticos en el código del programa

En dicho ejemplo es de suponer que hay un error en la línea 2 del código de programa, pues es exactamente ahí donde hemos eliminado las comillas para demostrarlo.

Los **errores sintácticos** también surgen cuando se quieren utilizar caracteres como texto y estos están asociados con una determinada tarea. Un ejemplo sería la comilla sencilla (‘). Los símbolos como este solo aparecen en PHP en calidad de texto cuando se hace entender al intérprete que el símbolo no estaría relacionado con su verdadero objetivo. En el caso de las comillas simples, existen dos posibilidades: por un lado se puede enmarcar un string con comillas simples entre comillas dobles o, por otro, se pueden enmascarar las comillas por medio de una barra invertida (\):

```
<?php
echo '\'Hello World!\' ';
?>
```




Si hay símbolos enmascarados con una barra invertida, estos serán exonerados de su función en la sintaxis de PHP.

También es posible la combinación de comillas sencillas y dobles:

```
<?php
echo " 'Hello World!' ";
?>
```

Sin embargo, esta grafía resultaría incorrecta:

```
<?php
echo ' 'Hello World!' ';
?>
```

En los ejemplos aparecen espacios entre las comillas para mejorar la legibilidad de los mismos.

## Instrucción printf

La instrucción printf ofrece más posibilidades que echo. Sintaxis:

```
<?php
printf(cadena formato, variable1, variable2, etc.);
?>
```

Donde se permiten las siguientes posibilidades para el formato en los casos más habituales:

- %s: Cadena de caracteres
- %d: Número sin decimales
- %f: Número con decimales
- %c: Carácter ASCII

Se muestra un ejemplo que incluye todas las posibilidades de printf:

- %b = 11101011011110011010000101 01
- %c = 2
- %d = 123456789
- %d = -123456789
- %e = 1.234568e+8
- %E = 1.234568E+8
- %u = 123456789
- %u = 18446744073586094827
- %f = 123456789.000000
- %F = 123456789.000000
- %g = 1.23457e+8
- %G = 1.23457E+8
- %o = 726746425
- %s = 123456789
- %x = 75bcd15
- %X = 75BCD15
- %+d = +123456789
- %+d = -123456789

Obteniéndose:

- %b = 11101011011110011010000101 01
- %c = 2
- %d = 123456789
- %d = -123456789
- %e = 1.234568e+8
- %E = 1.234568E+8
- %u = 123456789
- %u = 4171510507
- %f = 123456789.000000
- %F = 123456789.000000
- %g = 1.23457e+8
- %G = 1.23457E+8
- %o = 726746425
- %s = 123456789

- %x = 75bcd15
- %X = 75BCD15
- %+d = +123456789
- %+d = -123456789

Ejemplo de printf:

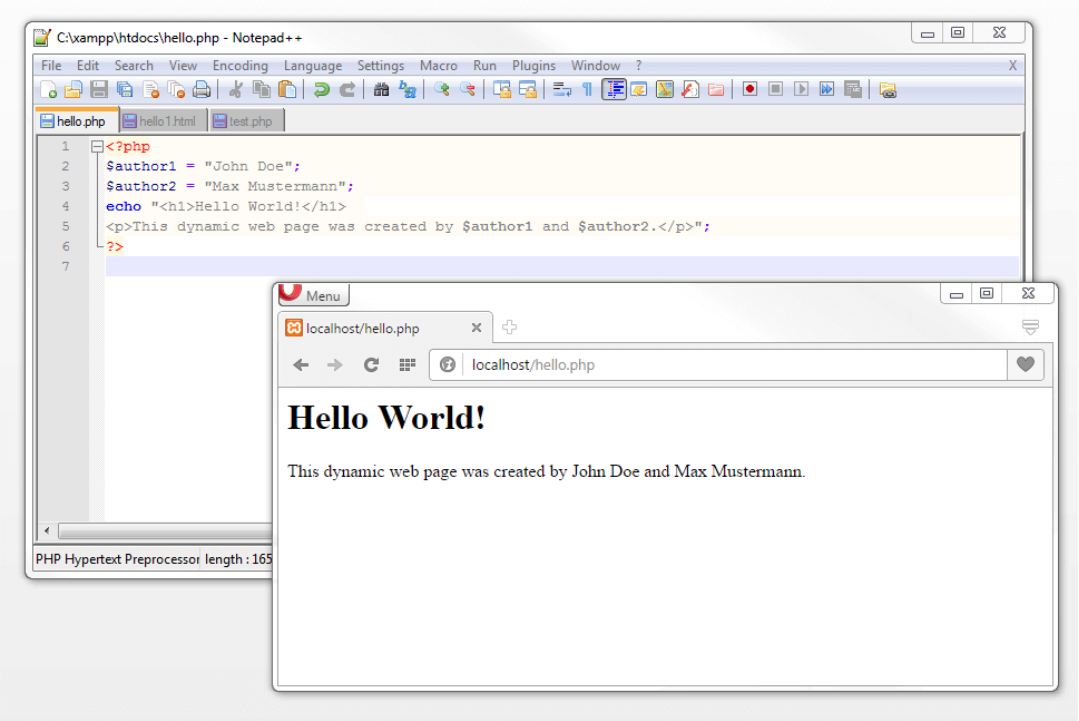
```
<html>
<head>
<title>Ejemplo de printf en PHP</title>
</head>
<body>
<?php
	$var = "texto"; $num = 3;
	printf("Puede fácilmente intercalar <b>%s</b> con números
	<b>%d</b> <br>",$var,$num);
	printf("<table border=1 cellpadding=10>");
	for ($i=0; $i<10; $i++)
	{
		printf("<tr><td>%10.d</td></tr>",$i);
	}
	printf("</table>");
?>
</body>
</html>
```

## Operadores de concatenación

Si se quiere incluir más de una variable en un script de PHP de manera simultánea, se puede hacer uso de lo aprendido hasta ahora:

```
<?php
$author1 = "John Doe";
$author2 = "Max Mustermann";
echo "<h1>Hello World!</h1>
<p>This dynamic web page was created by $author1 and $author2.</p>";
?>
```

Ambas variables se escriben con el resto del texto que se va a visualizar en el string señalado con las comillas dobles. PHP reconoce automáticamente las variables mediante el símbolo del dólar (*$*) y coloca tras ellos los valores correspondientes.



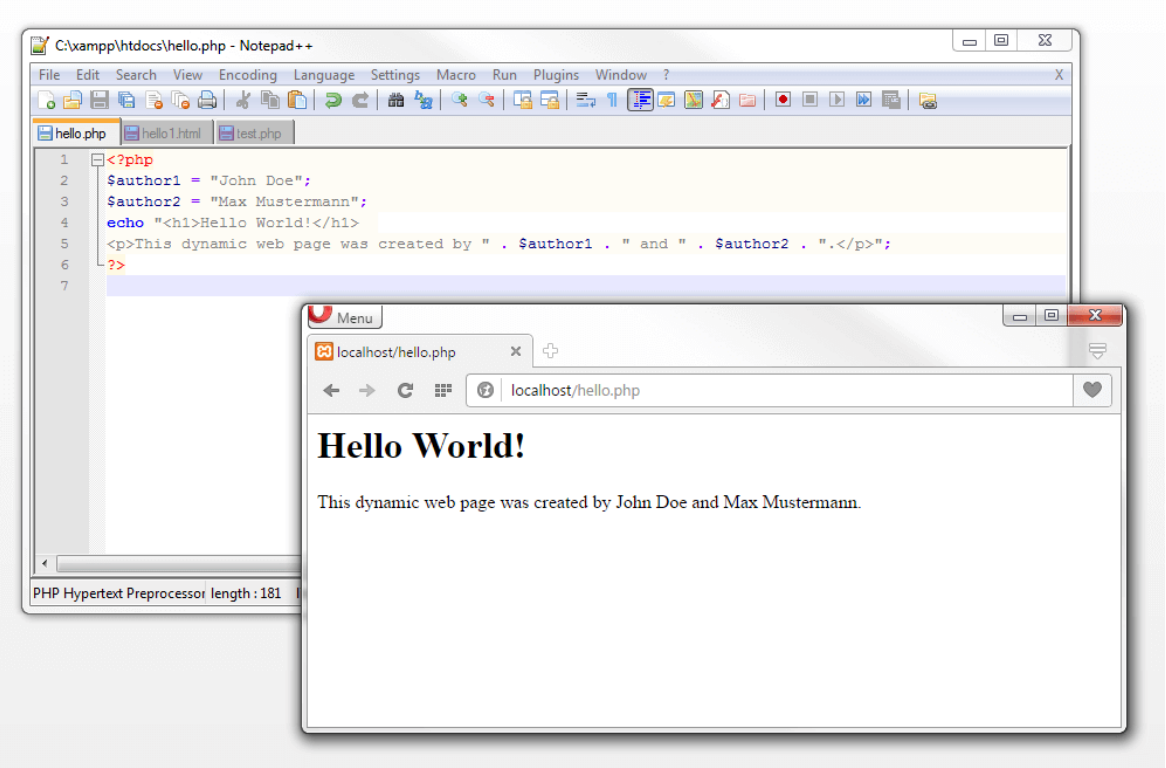

PHP solo acepta las variables en el string cuando el script no incluya ninguna función

Entre los programadores, seguir este procedimiento es considerado un trabajo sucio. En este sentido existe el dogma de que **las variables no deberían formar parte del string** y una de las razones para ello es que gran parte de los lenguajes de programación exigen dicha separación. Lo más importante es, sin embargo, que PHP también reclama la separación de string y variable cuando se trabaja con llamadas a funciones o variables complejas. En este caso es recomendable realizar esta separación incluso en el caso del texto sin formato aun cuando no fuera estrictamente necesario.

Al trabajar con variables siempre entran en juego varios elementos que deben estar interconectados entre sí. Para ello entra en juego el **operador de concatenación** (.).

Si se programa un código “limpio”, este debería tener el siguiente aspecto para el ejemplo mencionado anteriormente:

```
<?php
$author1 = "John Doe";
$author2 = "Max Mustermann";
echo '<h1>Hello World!</h1>
<p>This dynamic web page was created by ' . $author1 .' and ' .
$author2 .'.</p>';
?>
```




Los operadores de concatenación conectan strings y variables

Nos encontramos aquí ante tres strings y dos variables que están encadenados formando una secuencia de caracteres.

| **String1** | **Variable1** | **String2** | **Variable2** | **String3** |
|---|---|---|---|---|
| '<h1>Hello World!</h1> This dynamic web page was created by ' | . $author1 . | ' and ' | . $author2 . | '. ' |

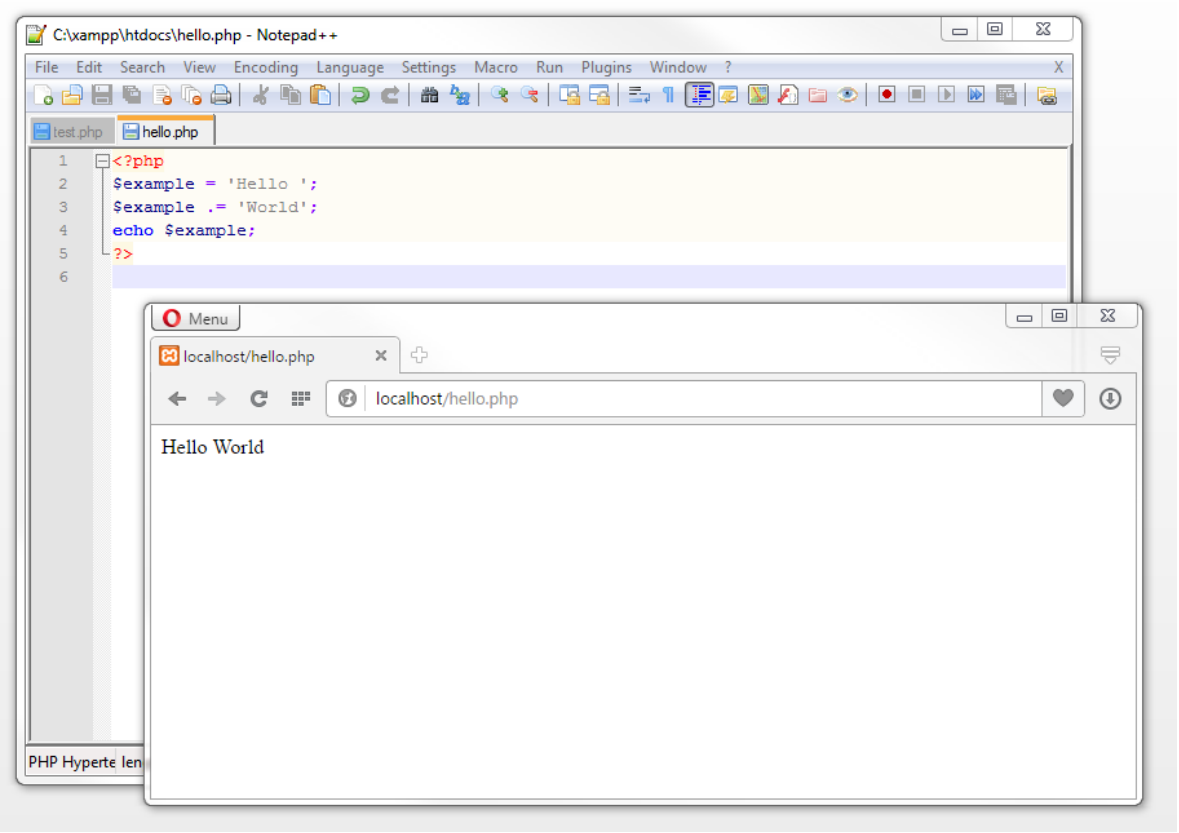
Es importante señalar que un operador de concatenación une strings o variables sin espacios. Si se quiere incluir un espacio, este tiene que escribirse, al igual que en el ejemplo, en el string y con comillas.

Los programadores no solo hacen uso de los operadores de concatenación para vincular strings y variables, sino también para prolongar las variables. Veamos cómo funciona con el siguiente ejemplo:

```
<?php
$example = 'Hello ';
$example .= 'World';
echo $example;
?>
```

Para prolongar el valor de una variable es necesario definirlo de nuevo, pero colocando el operador de concatenación *punto(.)* delante del signo de igualdad. Se trataría de la abreviación habitual para *$example = $example . 'World'* (*$ejemplo = $ejemplo . 'Mundo'*).

PHP agrega el nuevo valor al ya previamente definido. Si se quiere incluir un espacio entre ambos valores, este debe ir al final del primer string, tal y como se muestra en el ejemplo.




El string Hello se alarga y se convierte en Hello world.

## Cómo enlazar PHP en HTML

El intérprete de PHP solo se interesa, en principio, por el código colocado entre una etiqueta de PHP de apertura y otra de cierre:

```
<?php [Esta área será analizada por el intérprete de PHP] ?>
```

Este ignora el resto de fragmentos del documento y los transmite al servidor web tal y como aparecen. De esta forma, el código PHP puede integrarse en **cualquier documento HTML** para, por ejemplo, crear una plantilla para un sistema de gestión de contenidos. En este caso hay que tener en cuenta que los documentos HTML que contienen código PHP se han de guardar como archivos PHP. De lo contrario, el intérprete de PHP no podría procesar el documento, sino que este le sería entregado directamente al navegador, lo que tendría como consecuencia que el código del programa aparecería como texto plano en la página web.

Se puede imaginar al intérprete de PHP como el colega perezoso del servidor web que solo trabaja cuando se le exige explícitamente por medio de, por ejemplo, una etiqueta de apertura de PHP.

Si quieres **combinar HTML y PHP**, escribe la página HTML en su forma habitual conforme a la estructura clásica del documento y guárdala con la terminación *.php.*

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
<meta charset="utf-8">
<title>My first PHP page</title>
</head>
<body>
<h1>Hello World</h1>
<p>What is the current time and date?</p>
</body>
</html>
```

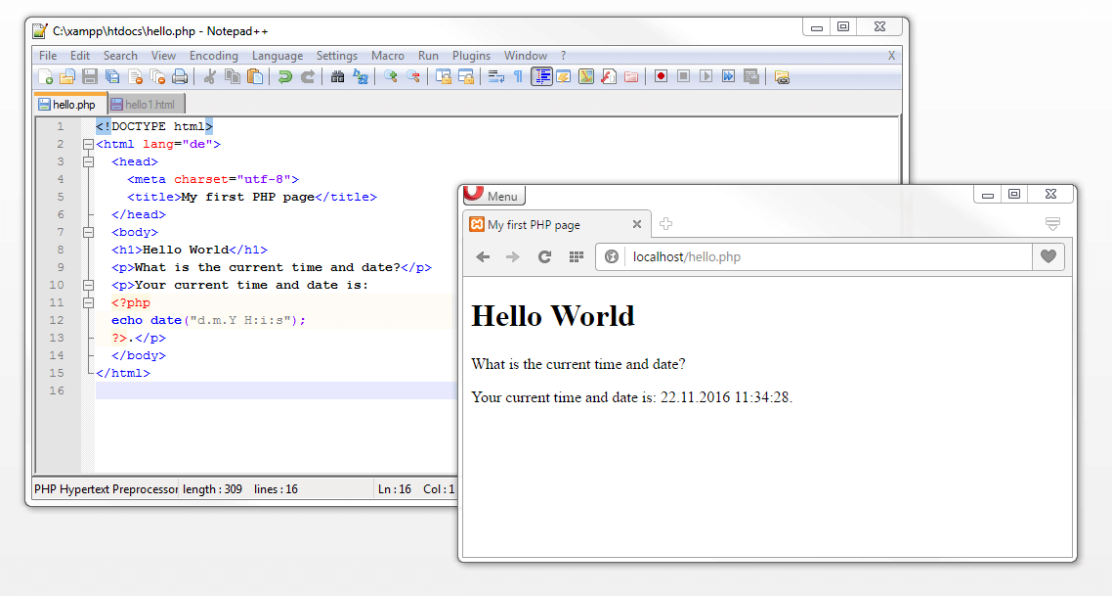
Complementa tu documento HTML para el script PHP, pero recuerda que la totalidad del código del programa ha de aparecer entre las etiquetas de PHP.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>My first PHP page</title>
</head>
<body>
<h1>Hello World</h1>
<p>What is the current time and date?</p>
<p>Your current time and date is:
<?php
echo date("d.m.Y H:i:s");
?>.</p>
</body>
</html>
```

En el ejemplo se ha combinado la construcción del lenguaje *echo* con la **función *date()*** para visualizar del lado del servidor la fecha y el horario actuales como texto. El parámetro de la función muestra el formato deseado en forma de string:

*d.m.Y H:i:s = dia.mes.año hora:minuto:segundo*

Si el navegador web solicita un archivo de tales características, el intérprete de PHP ejecuta el script y escribe el **horario actual en forma de texto** en el documento HTML, el cual será entregado por el servidor web y será representado como página web.




El documento HTML contiene un script de PHP integrado, el cual muestra la fecha y el horario actual.

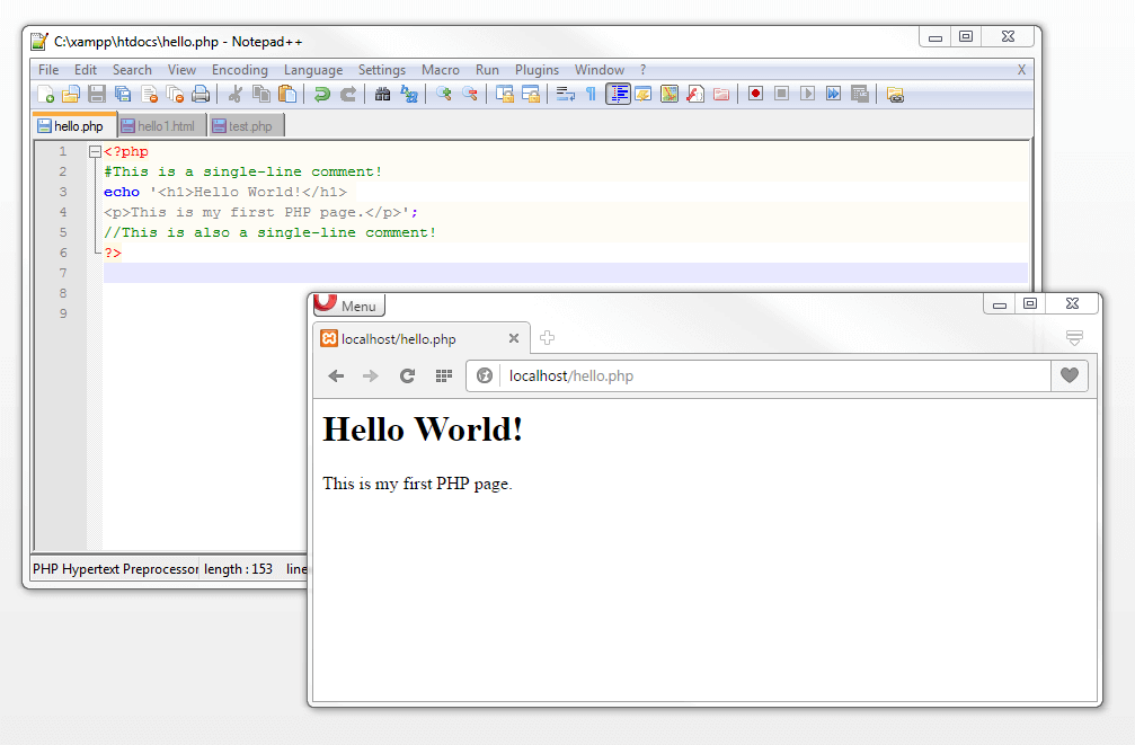
## Comentarios

Al igual que el código HTML, PHP también da la posibilidad de incluir comentarios. El intérprete de PHP ignora los **comentarios en el código fuente** siempre y cuando estos tengan unas características adecuadas para la sintaxis. Para ello, PHP pone a disposición tres alternativas diferentes.

Si se quiere destacar toda una línea como comentario y excluirla de la interpretación, se utilizará la **almohadilla (#)** o las **dos barras(//)**. En el siguiente ejemplo de código se emplean ambas posibilidades:

```
<?php
#This is a single-line comment!
echo '<h1>Hello World!</h1>
<p>This is my first PHP page.</p>';
//This is also a single-line comment!
?>
```

El editor de textos Notepad++ resalta los comentarios en color verde. Los pasajes de texto señalados como comentarios en el entorno del script no llegan siquiera al navegador web, a diferencia de los comentarios en HTML, puesto que el intérprete de PHP los ignora ya a la hora de ejecutar el script.

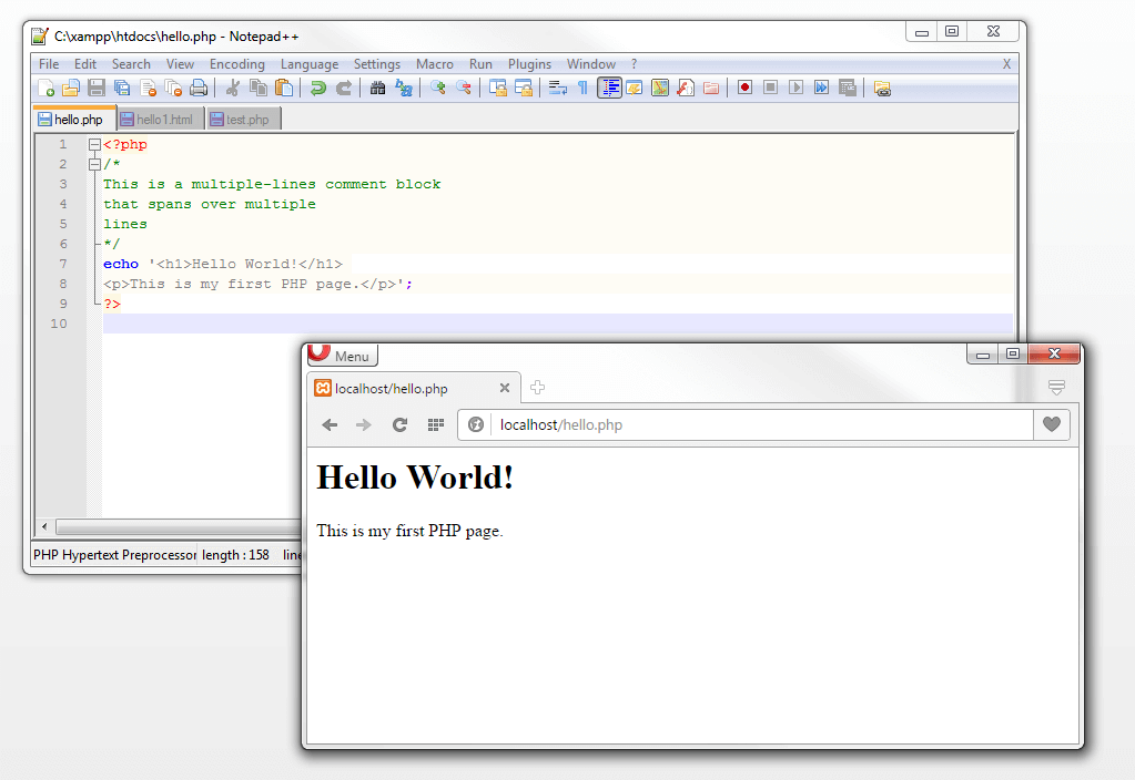



Las líneas marcadas como comentarios no se visualizan en la pantalla en la que se muestra el texto resultante

Asimismo, también se pueden insertar comentarios que comprenden varias líneas. Para ello, se puede marcar el inicio de una sección con comentarios con una **barra seguida de un asterisco(/*)** y el final con un **asterisco seguido de una barra (*/).**

```
<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
echo '<h1>Hello World!</h1>
<p>This is my first PHP page.</p>';
?>
```

Este tipo de comentarios destacados no son analizados sintácticamente y, por lo tanto, no aparecen en la página web.




El comentario de varias líneas no se visualiza en el texto publicado

Los programadores utilizan los comentarios para estructurar el código fuente de los scripts, para hacer indicaciones para su posterior edición o para añadir datos internos como por ejemplo el autor o la fecha.

La inclusión de comentarios es opcional y **no es recomendable abusar de su uso** para garantizarla buena legibilidad del código fuente.

## Realización de cálculos con variables

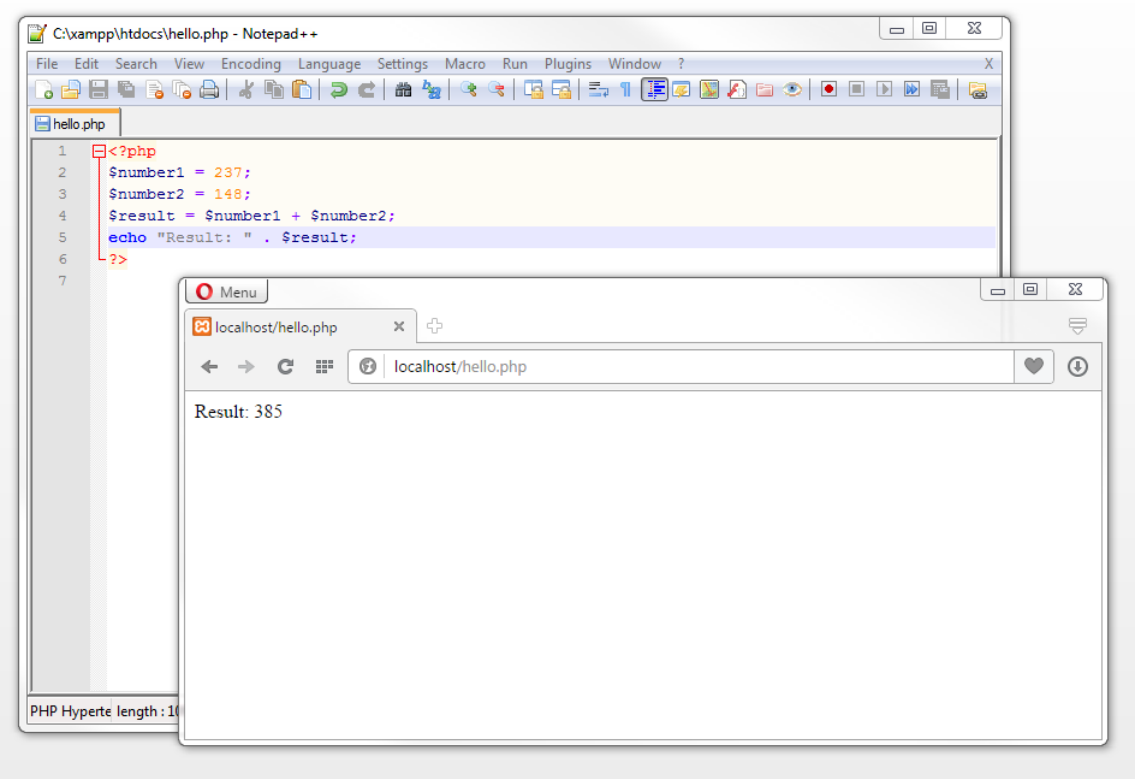
En nuestro tutorial para aprender a utilizar PHP ya has podido familiarizarte con las variables, a las que se han asignado en primera instancia valores string (cadenas de caracteres). Ahora pasaremos a ocuparnos de variables que representan **números enteros** (integers) o **números de punto flotante** (floats).

Si las variables almacenan valores numéricos, PHP ofrece la posibilidad de realizar cálculos con ellas. Veamos a continuación la traducción de una suma sencilla de dos números enteros:

```
<?php
$numero1 = 237;
$numero2 = 148;
$resultado = $numero1 + $numero2;
echo "Resultado: " . $resultado;
?>
```

En primer lugar, a las variables *$numero1* y *$numero2* se les asigna los números enteros 237 y 148 y, a continuación, se define una variable del tipo *$resultado*, la cual alberga la suma de las variables *$numero1* y *$numero2*. Para ello se emplea el **operador aritmético +** (más). Por último se emite el

resultado de la suma con ayuda del constructor *echo* como texto. Hay que tener en cuenta en este sentido que en la asignación de valores numéricos a las variables no son necesarias las comillas.



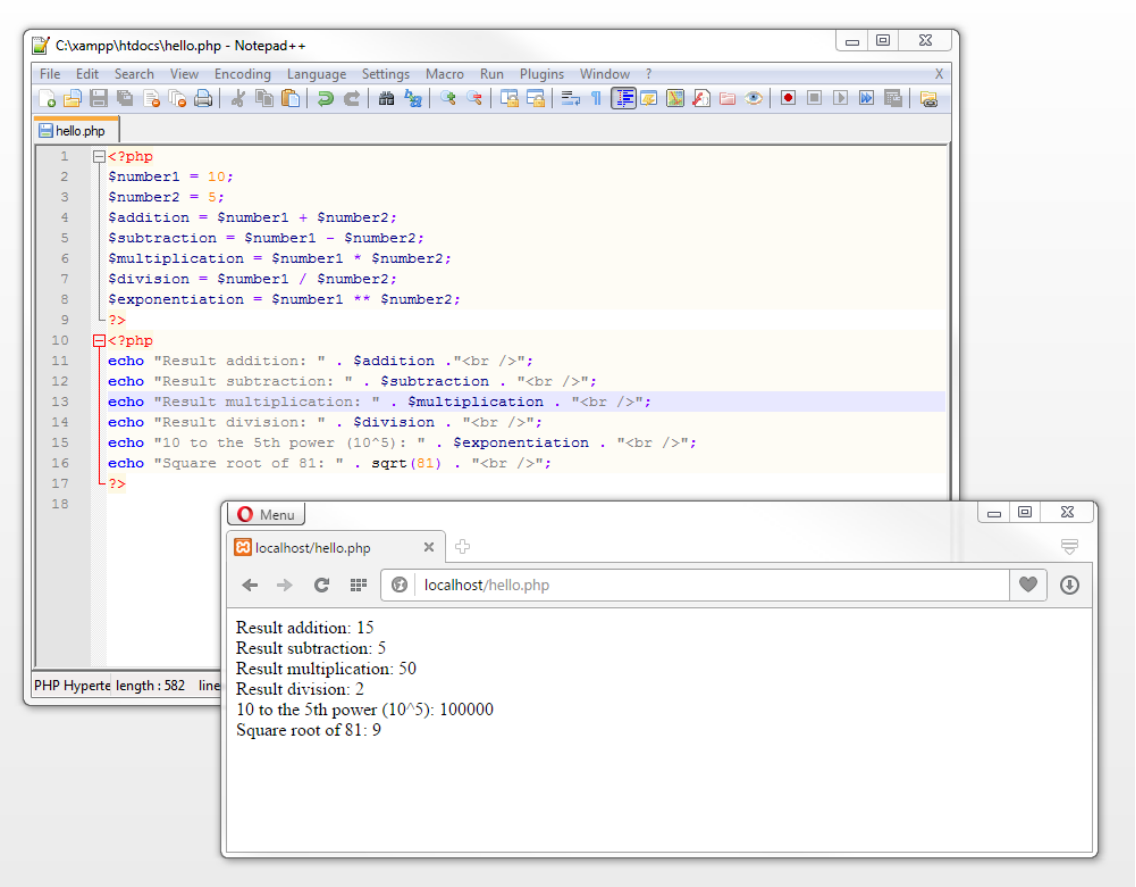

El resultado de la suma se muestra en forma de texto en el navegador web

El siguiente ejemplo de código, del que mostramos la versión en español de la subsiguiente captura de pantalla, muestra una **selección de cálculos matemáticos** que se pueden llevar a cabo con PHP del lado del servidor. Los operadores empleados se corresponden con gran parte de los operadores aritméticos estandarizados de las matemáticas.

| **Operador aritmético** | **Operación** | **Resultado** |
|---|---|---|
| $numero1 + $numero2 | Adición | Suma de $numero1 y $numero2 |
| $numero1 - $numero2 | Sustracción | Diferencia de $numero1 y $numero2 |
| $numero1 * $numero2 | Multiplicación | Producto de $numero1 y $numero2 |
| $numero1 / $numero2 | División | Cociente de $numero1 y $numero2 |
| $numero1 ** $numero2 | Exponenciación | Resultado de elevar $numero2 a la potencia $numero1 |

```
<?php
$numero1 = 10;
$numero2 = 5;
$adicion = $numero1 + $numero2;
$sustraccion = $numero1 - $numero2;
$multiplicacion = $numero1 * $numero2;
```

```
$division = $numero1 / $numero2;
$exponenciacion = $numero1 ** $numero2;
?>
<?php
echo "Resultado de la adición: " . $adicion ."<br />";
echo "Resultado de la sustracción: " . $sustraccion . "<br />";
echo "Resultado de la multiplicación: " . $multiplicacion . "<br />";
echo "Resultado de la división: " . $division . "<br />";
echo "10 elevado a la quinta potencia (10^5): " . $exponenciacion .
"<br />";
echo "Raíz de 81: " . sqrt(81) . "<br />";
?>
```




Resumen de operaciones matemáticas en inglés

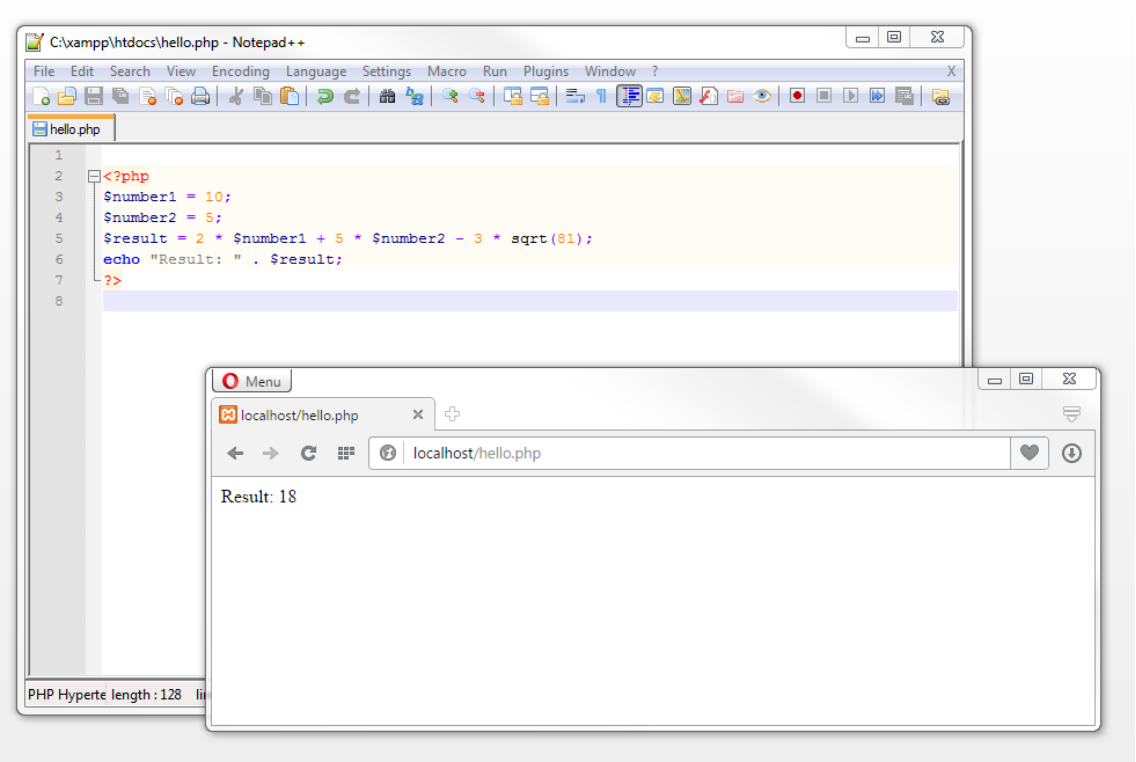
Para los cálculos más complejos se pueden combinar las diferentes operaciones aritméticas en un mismo script. Veamos cómo se plasmaría esto en español:

```
<?php
$numero1 = 10;
$numero2 = 5;
$resultado = 2 * $numero1 + 5 * $numero2 - 3 * sqrt(81);
echo "Resultado: " . $resultado;
?>
```

El intérprete PHP determina los valores de las variables y calcula:

*2 * 10 + 5 * 5 - 3 * √81 = 20 + 25 – 27 = 18*

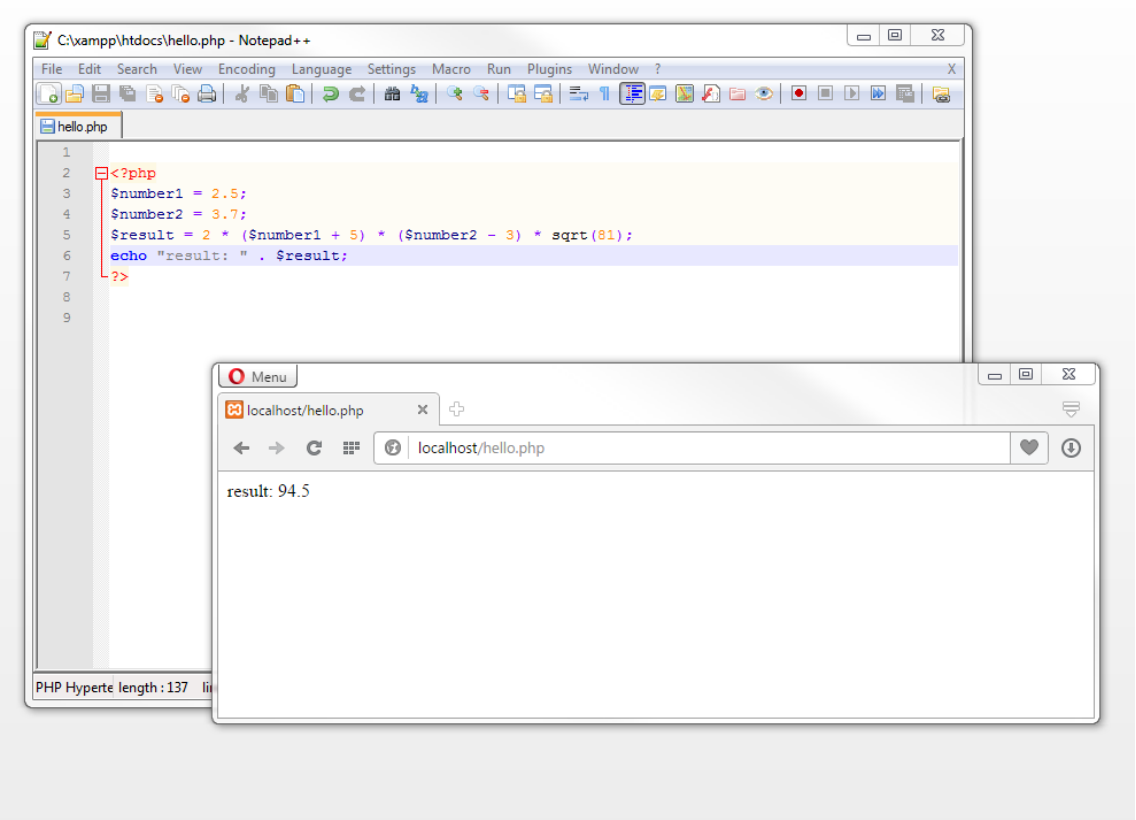
La función *sqrt()* calcula la raíz cuadrada del parámetro entre paréntesis. En este caso se aplica la clásica **jerarquía de operadores** de las matemáticas: los operadores de multiplicación y división tienen precedencia sobre los de adición y sustracción. La instrucción *echo* ofrece el resultado como string para el navegador web.




PHP se atiene a la regla de que los operadores de multiplicación y división tienen **precedencia** sobre los de adición y sustracción. **Los términos entre paréntesis** son los que primero se evalúan en PHP.

Ahora nos centraremos en los números de punto flotante:

```
<?php
$numero1 = 2.5;
$numero2 = 3.7;
$resultado = 2 * ($numero1 + 5) * ($numero2 - 3) * sqrt(81);
echo "Resultado: " . $resultado;
?>
```




PHP calcula 2*(2.5+5)*(3.7-3)*√81 y emite el resultado 94.5

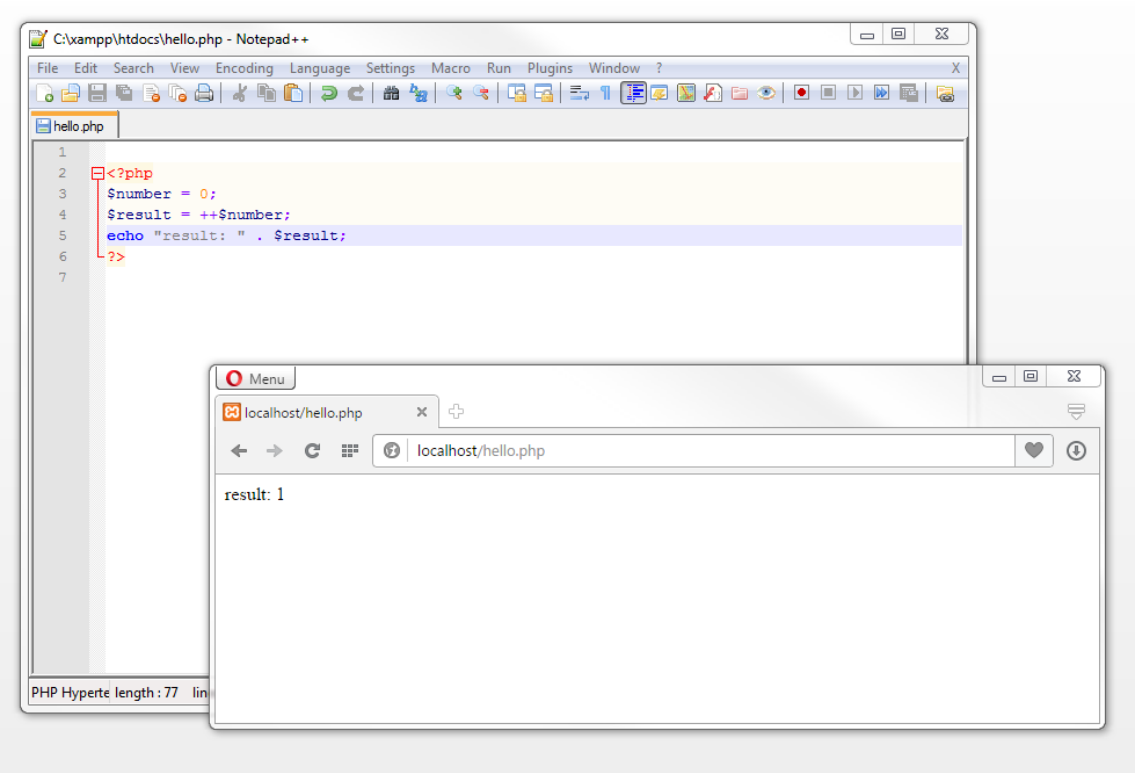
Como los lenguajes de programación más habituales, PHP también soporta operadores para **aumentar o disminuir en el valor 1 los valores numéricos**. En este sentido, se puede diferenciar entre el operador de preincremento, el de predecremento, el de postincremento y el de postdecremento.

| Operación | Operador | Resultado |
|---|---|---|
| Preincremento | ++$numero | El operador ++ incrementa el valor de la variable $numero, de forma que el valor aumenta en 1. El resultado será devuelto como el nuevo valor para $numero. |
| Predecremento | --$numero | El operador -- disminuye el valor de la variable $numero, de forma que el valor disminuye en 1. El resultado se convertirá en el nuevo valor para $numero. |
| Postincremento | $numero++ | Se devuelve el valor actual de $numero y se incrementa el valor en 1. |
| Postdecremento | $numero-- | Se devuelve el valor actual de $numero y se decrementa el valor en 1. |

En el ejemplo del preincremento se demuestran las operaciones aritméticas con operadores de incremento y decremento. El siguiente script en español incrementa el valor de la variable *$numero* en 1, guarda el valor nuevo en la variable *$resultado* y emite su valor como string:

```
<?php
```

```
$numero = 0;
$resultado = ++$numero;
echo "Resultado: " . $resultado;
?>
```

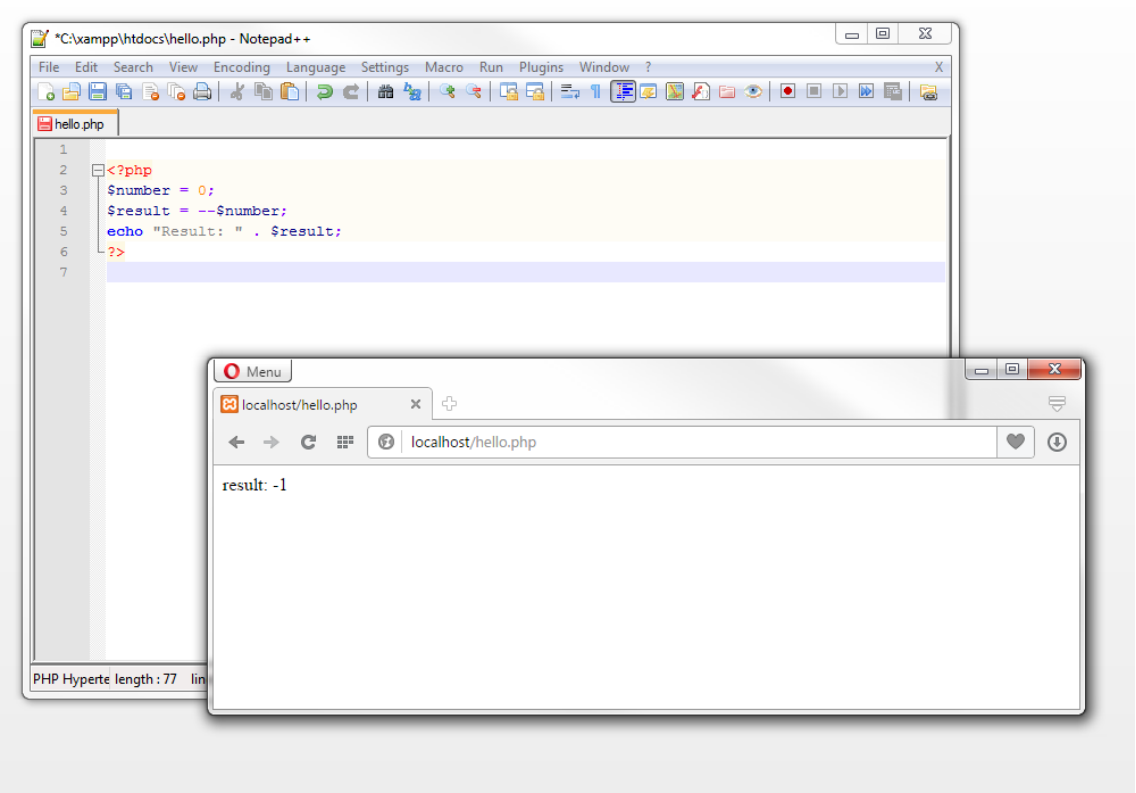



El operador ++ incrementa 0 en 1

Para calcular el predecrementode la variable *$number ($numero)* se recurre a los mismos scripts, pero cambiando el operador de preincremento I (*++*) por el operador de decremento (--):

```
<?php
$numero = 0;
$resultado = --$numero;
echo "Resultado: " . $resultado;
?>
```

Al reducir el valor 0 de la variable *$number ($numero),* se obtiene el resultado -1.

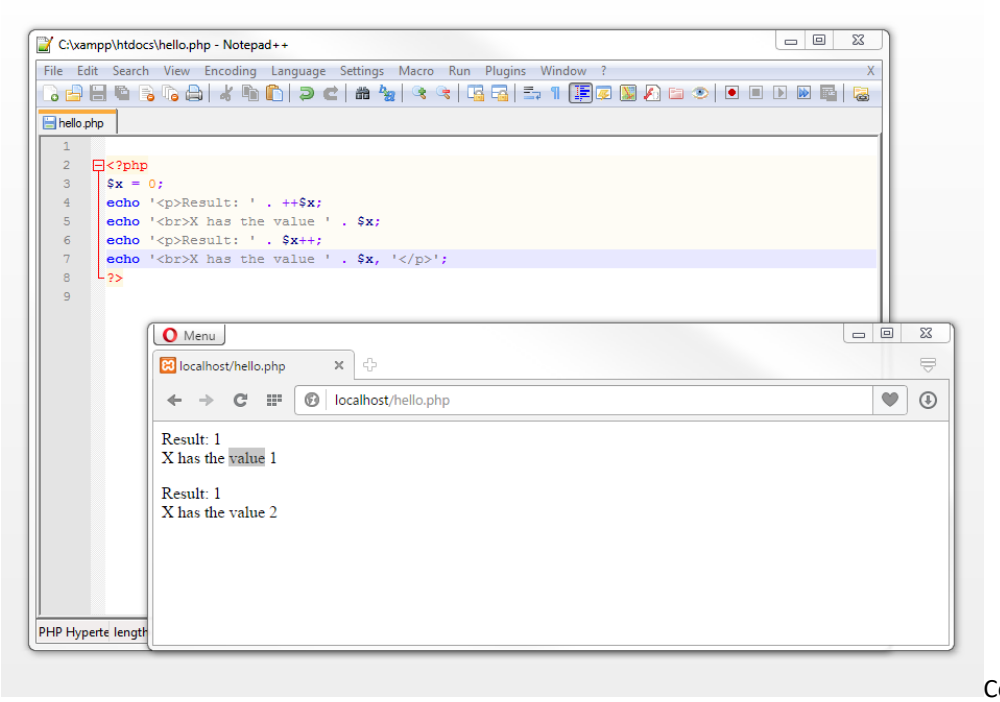



El operador -- hace disminuir el número 0 en 1

El incremento anterior o posterior (pre o post) de un valor puede ponerse de relieve en el siguiente script. Veámoslo con más claridad en la traducción de la captura de pantalla en cuestión:

```
<?php
$x = 0;
echo '<p>Resultado: ' . ++$x;
echo '<br>x tiene el valor ' . $x;
echo '<p>Resultado: ' . $x++;
echo '<br>x tiene el valor ' . $x, '</p>';
?>
```

En ambos casos se obtiene el mismo resultado. En el preincremento se incrementa el valor *x* antes de la edición en la línea 3 y en el postincremento esto ocurre en la línea 5.




Comparativa entre pre y postincremento

**Otros operadores en PHP**

- Operador de asignación:
  - =
- Operadores combinados: .=, +=, etc
  - $a = 3; $a += 5;  // a vale 8
  - $b = “hola ”; $b .= “mundo”;  //b vale “hola mundo”
  - Equivale a $b = $b . “mundo”;
- Operadores de comparación:
  - ==, !=, <, >, <=, >= y otros
- Operadores lógicos:
  - and (&&), or (||), !, xor
  - and/&& y or/|| tienen diferentes prioridades
- Operadores de cadena:
  - concatenación: . (punto)
  - asignación con concatenación: .=

Precedencia de operadores (de mayor a menor):

- ++, --
- *, /, %
- +,-
- <, <=, >, >=
- ==, !=
- &&
- ||
- and
- or

## Variables superglobales en PHP 7

Algunas variables predefinidas en PHP son "superglobales", lo que significa que están disponibles en todos los ámbitos a lo largo del script. No es necesario emplear **global $variable;** para acceder a ellas dentro de las funciones o métodos.

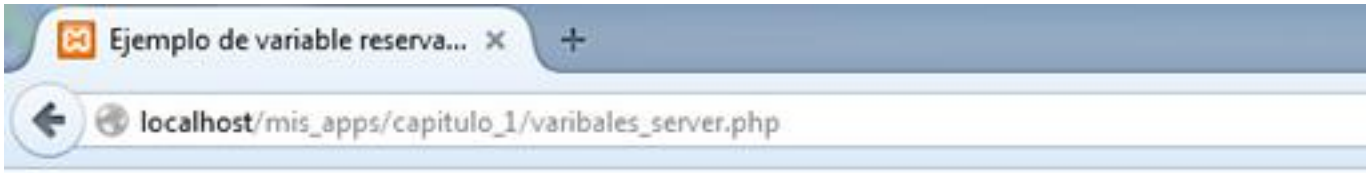
A continuación veremos una tabla descriptiva y después, ya en detalle cada una:

| Variable | Valor |
|---|---|
| $_SERVER | Información del entorno del servidor y de ejecución. |
| $_GET | Variables en el encabezado HTTP GET |
| $_POST | Variables recibidas en el encabezado HTTP GET |
| $_COOKIE | Variable con la cual podemos crear, accesar, editar o destruir Cookies. |
| $_FILES | Variables que llegan al servidor con archivos mediante carga. |
| $_REQUEST | Es una variable de array asociativo que por defecto contiene el contenido de $_GET, $_POST y $_COOKIE. |
| $_SESSION | Variables de sesión. |

## $_SERVER

La varibale **$_SERVER** nos devolverá en forma de array (matriz) información de servidor, rutas, conexiones, información del cliente y distintos *headers* recibidos. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_server.php** con el siguiente código:

```
 <html>
<head>
   <title>Ejemplo de variable reservada en PHP 7</title>
 </head>

 <body>

 <pre>

 <?php
 print_r($_SERVER);
  ?>
  </pre >

</body>
 </html>
```



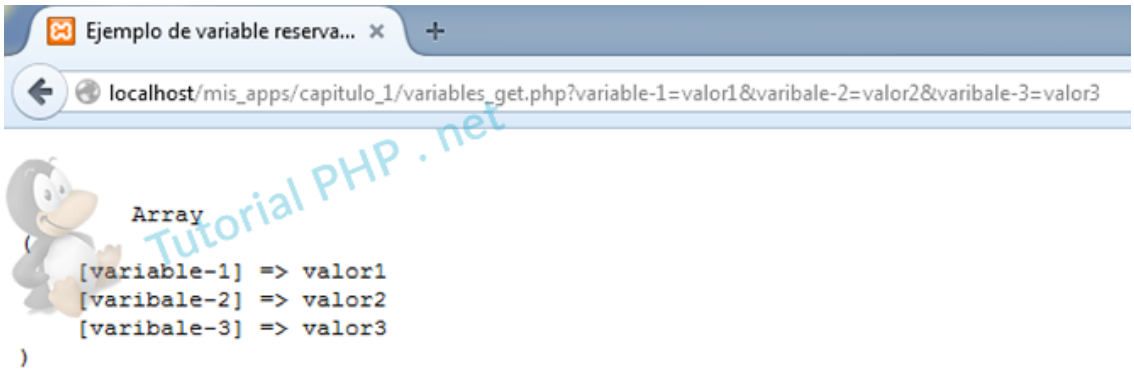

```
Array
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => \xampp\mysql\bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => \xampp\php
    [PHPRC] => \xampp\php
    [TMP] => \xampp\tmp
    [HTTP_HOST] => localhost
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64; rv:39.0) Ge
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0
    [HTTP_ACCEPT_LANGUAGE] => es-MX,es-ES;q=0.9,es;q=0.7,es-AR;q=0.6,es-
    [HTTP_ACCEPT_ENCODING] => gzip, deflate
    [HTTP_CONNECTION] => keep-alive
    [PATH] => C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:
    [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
    [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
    [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] =>
Apache/2.4.12 (Win32) OpenSSL/1.0.11 PHP/5.6.8 Server at localhost Port


    [SERVER_SOFTWARE] => Apache/2.4.12 (Win32) OpenSSL/1.0.11 PHP/5.6.8
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
    [REQUEST_SCHEME] => http
    [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
    [SERVER_ADMIN] => postmaster@localhost
    [SCRIPT_FILENAME] => C:/xampp/htdocs/mis_apps/capitulo_1/varibales_s
    [REMOTE_PORT] => 49504
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /mis_apps/capitulo_1/varibales_server.php
    [SCRIPT_NAME] => /mis_apps/capitulo_1/varibales_server.php
    [PHP_SELF] => /mis_apps/capitulo_1/varibales_server.php
    [REQUEST_TIME_FLOAT] => 1438834878.384
    [REQUEST_TIME] => 1438834878
)
```

## $_GET

La variable **$_GET** nos devolverá en forma de array (matriz) información de variables enviadas a través del parámetro HTTP GET, es decir, en la dirección de solicitud, variables y asignaciones con & = *(archivo.php?variable-1=valor1&varibale-2=valor2&varibale-3=valor3)*. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_get.php** con el siguiente código y lo ejecutaremos con los siguientes parámetros:

http://localhost/mis_apps/capitulo_1/variables_get.php?variable-1=valor1&varibale-2=valor2&varibale-3=valor3

```
 <html>
<head>
   <title>Ejemplo de variable reservada en PHP 7</title>
 </head>

 <body>

 <pre>

 <?php

print_r($_GET);

?>

 </pre >

</body>
 </html>
```

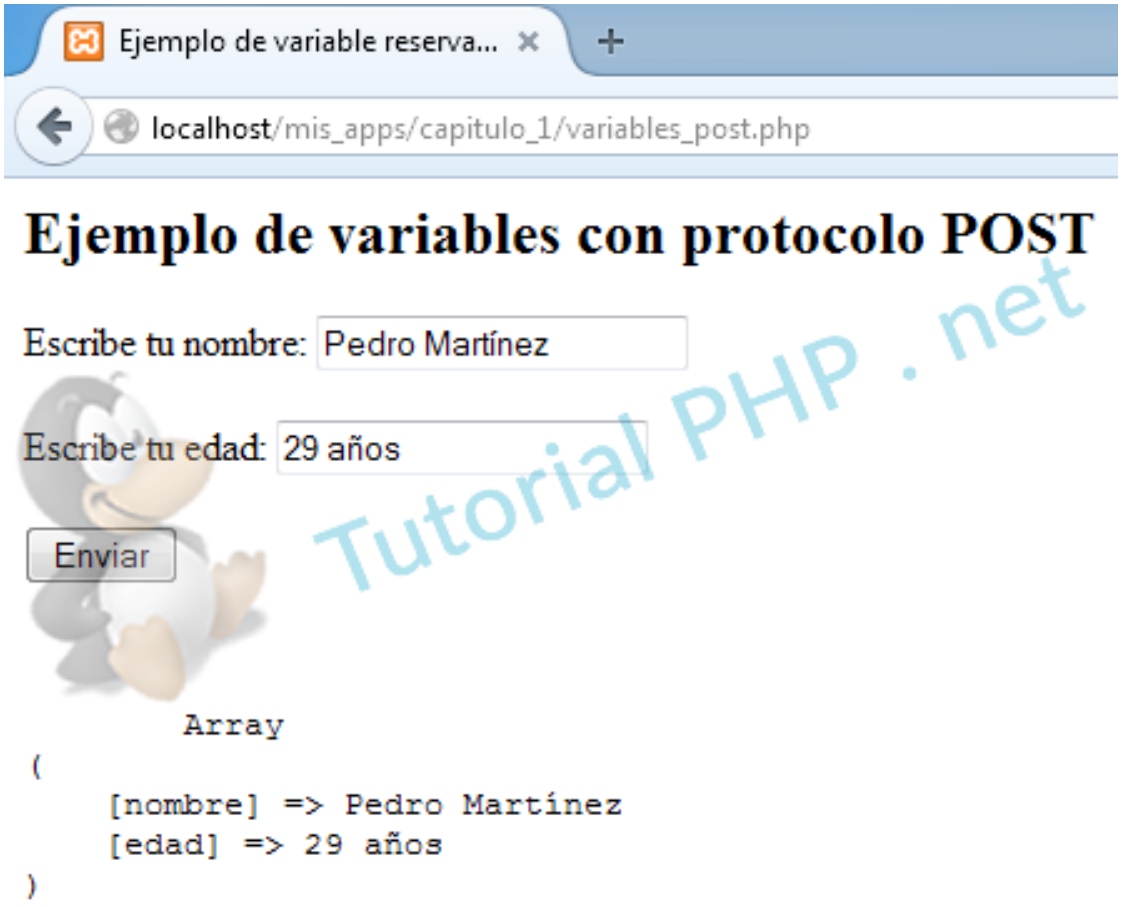



## $_POST

La variable **$_POST** nos devolverá en forma de array (matriz) información de variables enviadas a través del parámetro HTTP POST. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_post.php** con el siguiente código:

```
 <html>
<head>
   <title>Ejemplo de variable reservada en PHP 7</title>
```

```
  </head>

  <body>

  <h2>Ejemplo de variables con protocolo POST</h2>

  <form method="post">

  Escribe tu nombre: <input type="text" name="nombre"
value="<?=@$_POST['nombre'];?>"> <br> <br>

  Escribe tu edad: <input type="text" name="edad"
value="<?=@$_POST['edad'];?>"> <br> <br>

  <input type="submit" value="Enviar">

  </form>

 <pre>

  <?php

 if($_POST)
  {
   print_r($_POST);
  }

 ?>

  </pre >

 </body>
  </html>
```

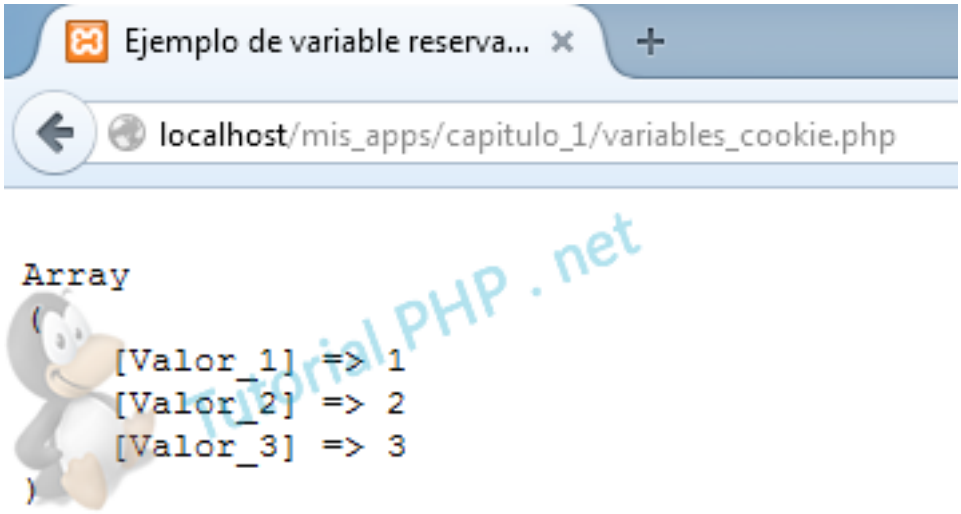



## $_COOKIE

La variable **$_COOKIE** nos devolverá en forma de array (matriz) información de cookies guardadas en nuestro cliente. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_cookie.php** con el siguiente código:

```
<?php
// Establecemos los valores de las Cookies
setcookie("Valor_1", "1");
setcookie("Valor_2", "2");
setcookie("Valor_3", "3");
?>
<html>
<head>
 <title>Ejemplo de variable reservada en PHP 7</title>
</head>
<body>
```

```
 <pre>

<?php

  print_r($_COOKIE);

 ?>

 </pre >

</body>
 </html>
```




## $_FILES

La variable **$_FILES** nos devolverá en forma de array (matriz) información de archivos que hayan llegado al servidor a través del protocolo HTTP POST. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_file.php** con el siguiente código:

```
 <html>

 <head>
  <title>Ejemplo de variable reservada en PHP 7</title>
 </head>

 <body>

 <h2>Ejemplo de variables FILES POST</h2>

 <form method="post" enctype="multipart/form-data">

 Archivo: <input type="file" name="archivo"> <br> <br>

<input type="submit" value="Enviar">

 </form>

<pre>
```

```
<?php

if($_FILES)
{
  print_r($_FILES);
}

?>

 </pre >

</body>
 </html>
```


## $_REQUEST

La variable **$_REQUEST** nos devolverá en forma de array (matriz) información en array asociativo con el contenido de las varibles $_GET, $_POST y $_COOKIE. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_request.php** con el siguiente código:

```
  <html>
```

```
 <head>
  <title>Ejemplo de variable reservada en PHP 7</title>
 </head>

 <body>

 <pre>

<?php

  print_r($_REQUEST);

?>

 </pre >

</body>
 </html>
```

## $_SESSION

La variable **$_SESSION** nos devolverá en forma de array (matriz) información de variables establecida en sesión, estando accesibles en cualquier parte de nuestra aplicación sin necesidad de requerir las definiciones. Veamos un ejemplo, dentro de nuestra carpeta **"mis_apps\capitulo_1"** crearemos un archivo llamado **variables_session.php** con el siguiente código:

```
 <?php

  // Siempre que utilicemos sesiones deberemos iniciar con
session_start.

  session_start();

   $_SESSION['variable_de_sesion_1'] = "Algún valor definido";

  $_SESSION['variable_de_sesion_2'] = "Algún otro valor definido";
 ?>

 <html>

 <head>
  <title>Ejemplo de variable reservada en PHP 7</title>
 </head>

 <body>

 <pre>

<?php

  print_r($_SESSION);

?>

 </pre >

</body>
 </html>
```

## Las variables superglobales $_GET y $_POST

Una vez has conocido los fundamentos de PHP y ya te sabes manejar con las variables, puedes concatenarlas y realizar cálculos. En este apartado te contamos por qué las variables son un factor esencial a la hora de programar scripts.

Una función importante de los lenguajes de programación es que estos ofrecen la posibilidad de analizar las entradas de los usuarios y de transferir los valores a otros scripts. Para ello, PHP se basa en las **variables superglobales *$_GET* y *$_POST,* variables de sistema** predefinidas que están disponibles en todos los ámbitos de validez. Como arrays asociativos (campos de datos), *$_GET* y *$_POST* almacenan un conjunto de variables en forma de strings en una variable.

Los **arrays** pueden imaginarse como si fueran un armario con varios cajones, cada uno de los cuales ofrece la posibilidad de archivar datos. Para poder saber posteriormente lo que alberga cada uno de dichos cajones, estos reciben un nombre de variable, que, en función del tipo de array, se puede tratar de un **index** o de una **key (llave)**. Mientras que en el caso de los **arrays indexados** se le otorga a cada cajón un índice en forma de número, a los cajones de un **array asociativo** se les asigna una key en forma de string (secuencia de caracteres).

Las variables superglobales *$_GET* y *$_POST* contienen una serie de variables en forma de llaves que permiten llegar a los valores vinculados a dichas llaves. Hablaremos en detalle de este tema cuando nos centremos en las variables superglobales *$_GET* y *$_POST.*

## Transferencia de datos vía $_GET

La **variable superglobal *$_GET*** representa un array de variables que se transfiere a un script PHP con ayuda de un URL.

Si visitas **weblogs**, **tiendas online** o **foros de Internet**, es posible que te hayan llamado la atención los peculiares URL que aparecen en ellos. Suelen generarse siguiendo el esquema siguiente:

*nombredeequipo/carpeta/nombredearchivo.php*variable

En un weblog, el esquema puede tener la siguiente apariencia:

*www.ejemplo-blog.es/index.php*

Un URL de este tipo puede desglosarse de manera muy sencilla: en un servidor web con el dominio *ejemplo-blog.es* existe un archivo con el nombre *index.php,* que sirve para crear una página web dinámica. Por lo general, este contiene código HTML y PHP, así como enlaces a archivos de plantillas y a hojas de estilo externas, es decir, todo lo necesario para representar una página web. El indicador que delata que se trata de una página web dinámica es el código que sigue al signo de interrogación (*?*): *id=1.* Este recibe la denominación de **HTTP querystring** o cadena de consulta de HTTP y está formando por una variable (*id)* y un valor (*1*), ambos unidos por el signo igual. Los parámetros URL de este tipo se utilizan, por ejemplo, para generar páginas web dinámicas, para cargar contenidos de las bases de datos o para solicitar la plantilla adecuada.

Las páginas web dinámicas permiten la **separación entre contenido y presentación**. El elemento *index.php*contieneprácticamentetoda la información sobre la estructura de la página web, pero no alberga los contenidos. Estos se depositan normalmente en una **base de datos** y se puede acceder a ellos a través de los parámetros en el HTTP querystring. En nuestro ejemplo, el URL entrega al *index.php*el parámetro *id=1,* el cual determina qué contenidos de la base de datos se tienen que leer y

cargarse en el *index.php.* En el caso de los weblogs se trata, generalmente, del identificador de un artículo determinado. En los foros esto permite visitar una entrada o, en las tiendas online, ver un producto determinado.

Si un URL contiene más de un parámetro, estos se unen entre sí con el **símbolo at (&)**.

*www.ejemplo-blog.es/index.php?page=article&id=1*

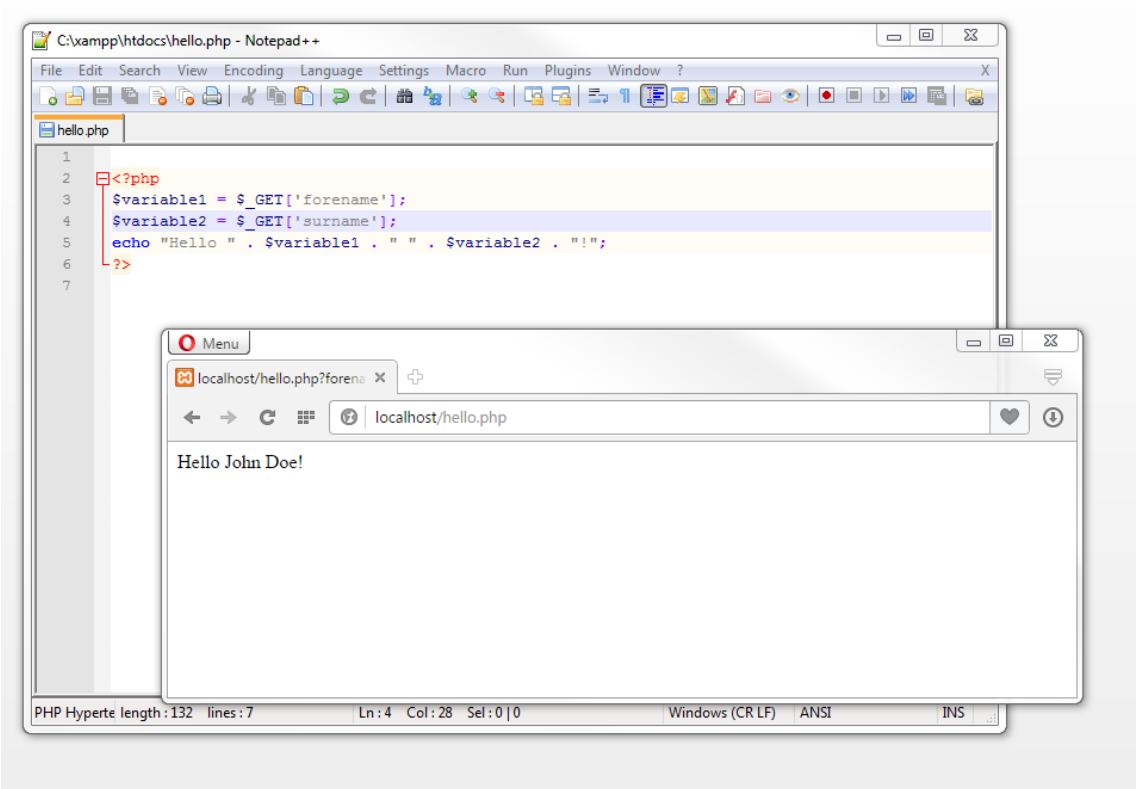
No es necesario recurrir a una base de datos para entender cómo se emplea *$_GET* en un ejemplo de código. En el siguiente script se utiliza la superglobal*$_GET* para interpretar los valores de las variables *nombre* y *apellido (forename y surname)* de una cadena de consulta HTTP y para escribirlos en las variables PHP *$variable1* y *$variable2*:

```
<?php
$variable1 = $_GET['forename'];
$variable2 = $_GET['surname'];
echo "Hello " . $variable1 . " " . $variable2 . "!";
?>
```

Para abrir el script usamos el siguiente URL:

*localhost/hello.php?forename=John&surname=Doe*

Con ello se entregan los parámetros *forename=John* y *surname=Doe*. La emisión de los valores tiene lugar, como hasta ahora, con ayuda de la construcción del lenguaje *echo*.




Los parámetros de URL entregan al script de PHP el par de valores de las variables forename=John y surname=Doe

La transmisión de datos vía *$_GET* da lugar inevitablemente a que los **datos transferidos se puedan visualizar en la línea de direcciones**. En todo momento puede comprobarse cuáles son los parámetros

que se van a entregar, lo que tiene la ventaja de que las variables pueden guardarse en hiperenlaces. Además, los usuarios de Internet también tienen la posibilidad de depositar los URL e incluso el querystring de HTTP como marcador en el navegador.

El hecho de que el parámetro GET se ejecute en el URL como texto sin codificar, descalifica, sin embargo, a este método para la entrega de datos sensibles como los que albergan los formularios online. Asimismo, el volumen de datos que se puede entregar por medio de la variable superglobal *$_GET* está limitado a la longitud máxima de los URL, limitaciones que pueden evitarse con el método POST de HTTP. Los datos transferidos con él se encuentran en la variable superglobal ***$_POST.***

## Transferencia de datos vía $_POST

Mientras que los datos que se transfieren mediante el método GET se entregan como parámetros URL, la transferencia de datos vía *$_POST* se realiza en el cuerpo de una petición HTTP. Esto permite a los desarrolladores transferir grandes cantidades de datos de un script a otro.

Un campo de aplicación esencial del método HTTP-POST es la **transmisión de datos de formularios HTML.** Veámoslo con el ejemplo de una suscripción a una newsletter.

Para ello, crea un nuevo archivo PHP con el nombre *page1.php* e incluye el siguiente bloque de código en español (para ver el código en inglés, consulta la captura de pantalla que aparece a continuación):

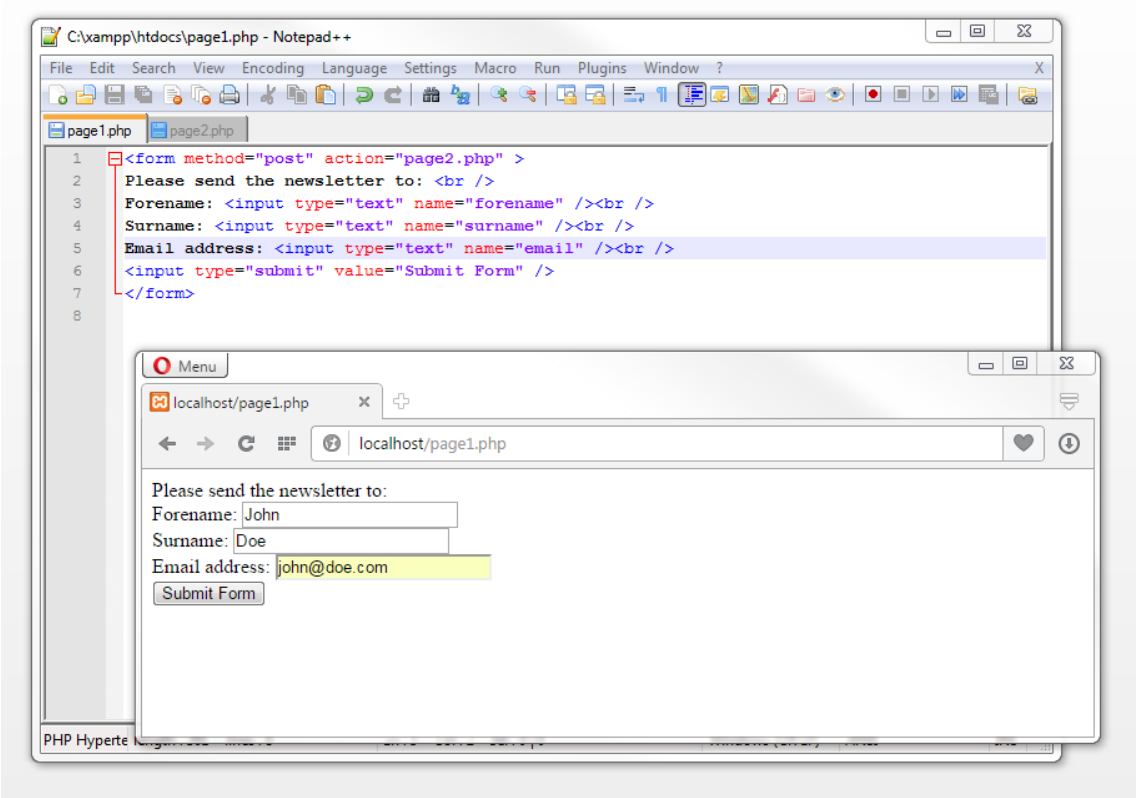
```
<form method="post" action="page2.php" >
Por favor, envía la newsletter a: <br />
Tu nombre: <input type="text" name="nombre" /><br />
Tu apellido: <input type="text" name="apellido" /><br />
Tu correo electrónico: <input type="text" name="email" /><br />
<input type="submit" value="Enviarformulario" />
</form>
```

En la creación de formularios entra en juego el elemento HTML *<form>*. Este incluye en la etiqueta de inicio dos atributos: *method*y *action.* Con el primero se pueden definir los métodos de transmisión, en este caso el método POST de HTTP. En el atributo *action* se deposita el URL de un script que recibe todos los datos registrados a través de los siguientes campos de entrada. El ejemplo muestra un **formulario HTML con tres elementos de entrada** (*input type="text"*) y un **botón de envío** (*input type="submit"*). Como receptor de los datos, se define el archivo *page2.php*.

Para entender la transferencia de datos por medio de *$_POST*, recurrimos a un script sencillo para el análisis de los datos del formulario que almacena los valores introducidos como variables PHP en formato de texto. Crea para ello un archivo *page2.php* e inserta el siguiente código de programación:

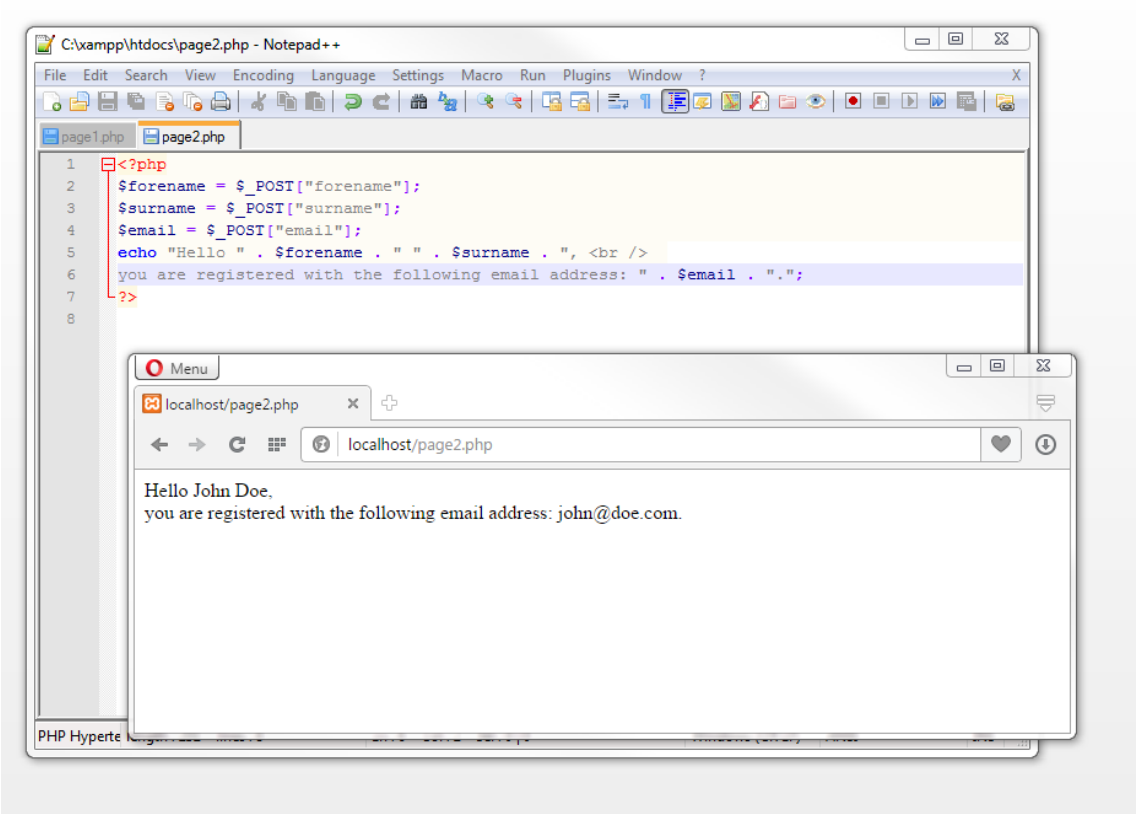
```
<?php
$nombre = $_POST["nombre"];
$apellido = $_POST["apellido"];
$email = $_POST["email"];
echo "Hola " . $nombre . " " . $apellido . ", <br />
Te has registrado con el siguiente correo electrónico: " . $email .
".";
?>
```

Guarda los dos archivos PHP en la carpeta *htdocs* de tu servidor de prueba y abre *page1.php* con el siguiente URL en el navegador web: *http:// localhost/ page1.php*. El navegador muestra entonces la interfaz web interactiva de tu formulario HTML.




El formulario HTML pagina1.php incluye datos de usuario y los envía al script con la dirección page2.php

Introduce cualquier dato de registro y haz clic en el botón de enviar para transferir variables de un script a otro. Tan pronto como confirmes los datos en *page1.php,* se te redirige directamente a *page2.php.* La ventana del navegador muestra el resultado de la ejecución del script en base a los datos transmitidos.




Transmisión de datos con el método POST de HTTP y emisión vía echo

La *page2.php* solicita los datos de usuario registrados a través de los campos de entrada de la *page1.php* por medio del siguiente esquema:

```
$_POST["Nombre del campo de entrada"]
```

Así es como la línea *$forename = $_POST["forename"]* solicita el *nombre de pila* en el campo de entrada y lo guarda en la variable *$forename,* que, a su vez, se emite como string gracias a *echo.*

## La sentencia if y los operadores de comparación de PHP

Hasta ahora hemos definido las variables, pero también hemos mostrado cómo se transmiten estas de un script a otro y se emiten en forma de strings. A continuación te mostramos cómo se puede vincular la ejecución de los fragmentos de código con unas **condiciones** determinadas.

La **sentencia *if*** brinda la posibilidad de escribir scripts de tal manera que las instrucciones surten efecto cuando el usuario cumple una de las condiciones previstas, como puede ser, por ejemplo, la introducción de una contraseña correcta.

Las condiciones se pueden definir en PHP según la siguiente estructura básica:

```
<?php
if(expression)
    {
statement;
    }
```

```
?>
```

Esta se lee de la siguiente manera: solo en aquellos casos en los que se cumpla la condición en *expression* se podrá ejecutar el *statement.* Una condición se ha cumplido cuando el constructor *if* da como resultado *TRUE* (verdadero). De no ser así, este se interpreta como *FALSE* (falso), en cuyo caso se omitirá la instrucción.

Por regla general, la instrucción *if* comprueba si el valor de una variable se corresponde con lo definido en la condición. Esta **estructura de control** tiene lugar normalmente **en base a los operadores de comparación**.

## Operadores de comparación

Los operadores de comparación se utilizan a la hora de formular condiciones, con el objetivo de poner dos argumentos en una **relación lógica** que se puede evaluar como verdadera (*TRUE*) o falsa (*FALSE*). Si se emplean los operadores de comparación en las estructuras de control de PHP, estos pueden implementarse con dos variables en la *expression* de una sentencia *if:*

```
if ($a == $b)
   {
statement;
   }
```

Expresemos la estructura de control con palabras: las condiciones definidas se llevan a cabo en caso de que la variable *$a* sea equivalente a la variable *$b*.

Los **operadores de comparación de PHP** se basan en el lenguaje de programación C y se diferencian considerablemente de los símbolos matemáticos clásicos en su escritura. En la tabla siguiente te ofrecemos una lista detallada de los mismos:

| Operador de comparación | Descripción | Condiciones |
|---|---|---|
| == | Igual | La condición se cumple si $a y $b muestran el mismo valor. |
| === | Idéntico | La condición se cumple si $a y $b muestran el mismo valor y pertenecen al mismo tipo de datos. Esto se puede visualizar en un ejemplo en el que se compara un íntegro (1) con un string ("1"): 1 == "1" //TRUE 1 === "1" //FALSE. Lo más recomendable es utilizar siempre el operador de comparación === (idéntico) para aquellas condiciones que requieren dos variables. |
| != | Diferente | La condición se cumple si $a y $b tienen valores diferentes. |
| !== | No idéntico | La condición se cumple si $a y $b tienen valores que no son iguales o que pertenecen a tipos de datos diferentes. |
| < | Menor que | La condición se cumple si el valor de $a es menor que el de $b. |
| > | Mayor que | La condición se cumple si el valor de $a es mayor que el de $b. |
| <= | Menor o igual que | La condición se cumple si el valor de $a es menor que el valor de $b o si $a y $b tienen el mismo valor. |

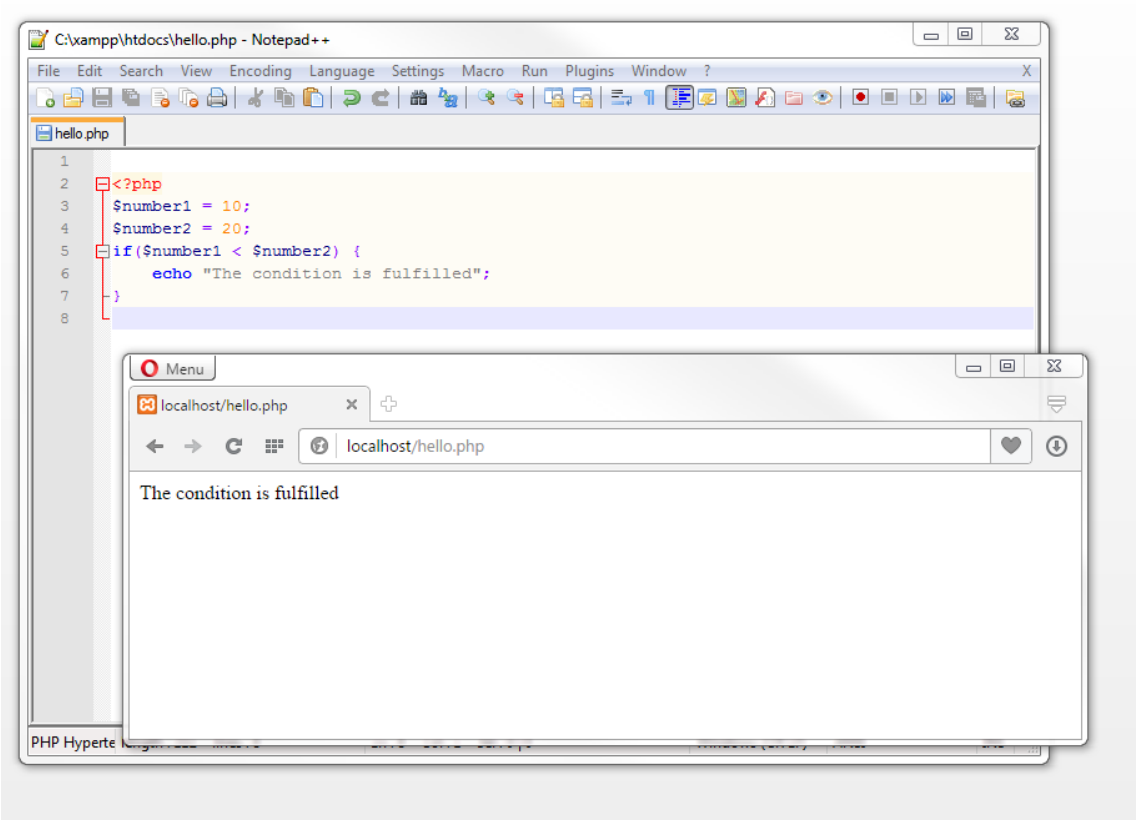
| Operador de comparación | Descripción | Condiciones |
|---|---|---|
| >= | Mayor o igual que | La condición se cumple si el valor de $a es mayor que el valor de $b o si $a y $b tienen el mismo valor. |

El siguiente script debe arrojar luz a esta estructura de control, en la que se comparan dos números enteros. El operador de comparación que se utiliza es <(menor que):

```
<?php
$numero1 = 10;
$numero2 = 20;
if($numero1 < $numero2) {
echo "Se cumple la condición";
}
```

En este caso, se definen las variables *$numero1* y *$numero2* y se les asignan los valores *10* y *20*. A continuación se establece una condición: si *$numero1* es menor que *$numero2*, se emitirá el string mencionado en la condición de *echo*.

El resultado de la ejecución del script contiene la respuesta: 10 es menor que 20. El constructor *if* devuelve el resultado *TRUE*. Tras ello, se puede decir que la condición se ha cumplido (*The condition is fulfilled*).




Si se cumple la condición, se ejecuta la instrucción.

Si quieres definir instrucciones que se ejecuten en el caso de que una condición no se cumpla puedes complementar la sentencia *if* con la **sentencia *else*** según el esquema siguiente:

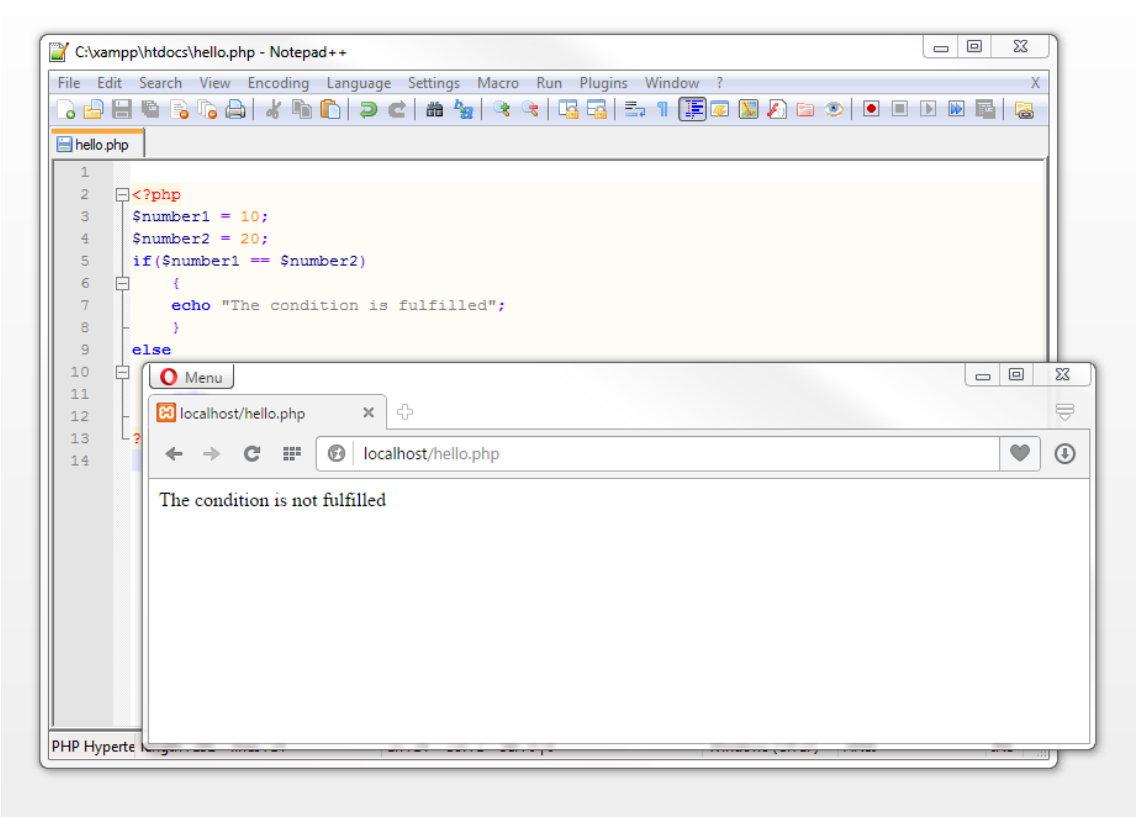
```
<?php
if(condición a)
    {
      instrucción b;
    }
else
    {
      instrucción c
    }
?>
```

Este script también comprueba si la *condición a* devuelve el resultado *TRUE* o *FALSE*. Si se cumple la *condición a* (*TRUE*) se ejecutará la *instrucción b*. Si no se cumple la condición *a(FALSE)*, se saltará la *instrucción b* y, en su lugar, se ejecutará la *instrucción c*.

A continuación, ampliamos nuestro script con el constructor *else* y cambiamos el operador de comparación < (menor que) por == (igual):

```
<?php
$numero1 = 10;
$numero2 = 20;
if($numero1 == $numero2)
    {
echo "La condición se cumple";
    }
else
    {
echo "La condición no se cumple";
    }
?>
```

En este caso, la sentencia *if* ofrece como resultado *FALSE.* El valor de la variable *$numero1* no es igual que el valor de la variable *$numero2*. La condición no se cumple (*The condition is not fulfilled*). Por lo tanto, no se ejecuta bajo la instrucción *if,* sino bajo la definida por *else*.




Si no se cumple la condición, se ejecuta la instrucción definida por else.

**Atención**

Si la ejecución de un fragmento de código está sujeta a la igualdad de dos valores, se utiliza el signo igual dos veces (==). Se utiliza un único signo de igualdad (=) en el proceso de adjudicación de valores a variables.

La **negación de las condiciones** tiene lugar mediante un signo de exclamación (*!*) dentro de la *expression*.

```
<?php
$numero1 = 10;
$numero2 = 20;
if ($numero1 == $numero2)
    {
      echo "Los números son iguales.";
    }
if (!($numero1 == $numero2))
    {
      echo "Los números no son iguales.";
    }
?>
```

El ejemplo muestra la condición $numero1 == $numero2 y su negación. !($numero1 == $numero2) equivale a ($numero1 != $numero2).

Una aplicación práctica de if y else es, por ejemplo, la **solicitud de contraseña** en un formulario HTML. A continuación, veamos una simulación con ayuda de los ficheros PHP page1.php y page2.php.

Abre el fichero page1.php e introduce el siguiente código de formulario:

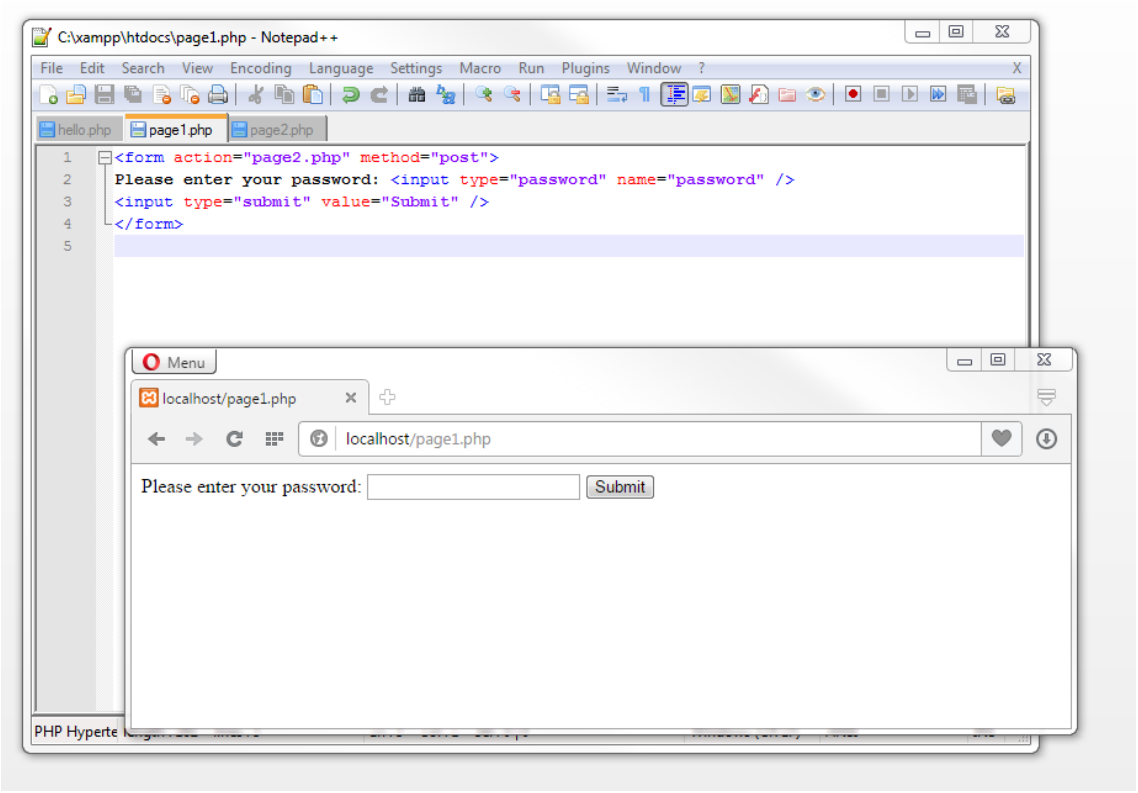
```
<form action="page2.php" method="post">
Por favor, introduce tu contraseña: <input type="password"
name="contraseña" />
<input type="submit" value="Enviar" />
</form>
```

La estructura se corresponde con el formulario ya creado. En este caso solo es necesario un campo de entrada: la solicitud de la contraseña. Como se ha descrito anteriormente, las entradas del usuario se comunican al script *page2.php*.Este puede adaptarse con el siguiente código de manera que la introducción de la contraseña **se compare con una contraseña ya facilitada**:

```
<?php
$contrasena = $_POST["contrasena"];
if($contrasena=="qwertz123")
   {
echo "La contraseña es correcta";
   }
else
   {
echo "La contraseña es incorrecta";
   }
?>
```

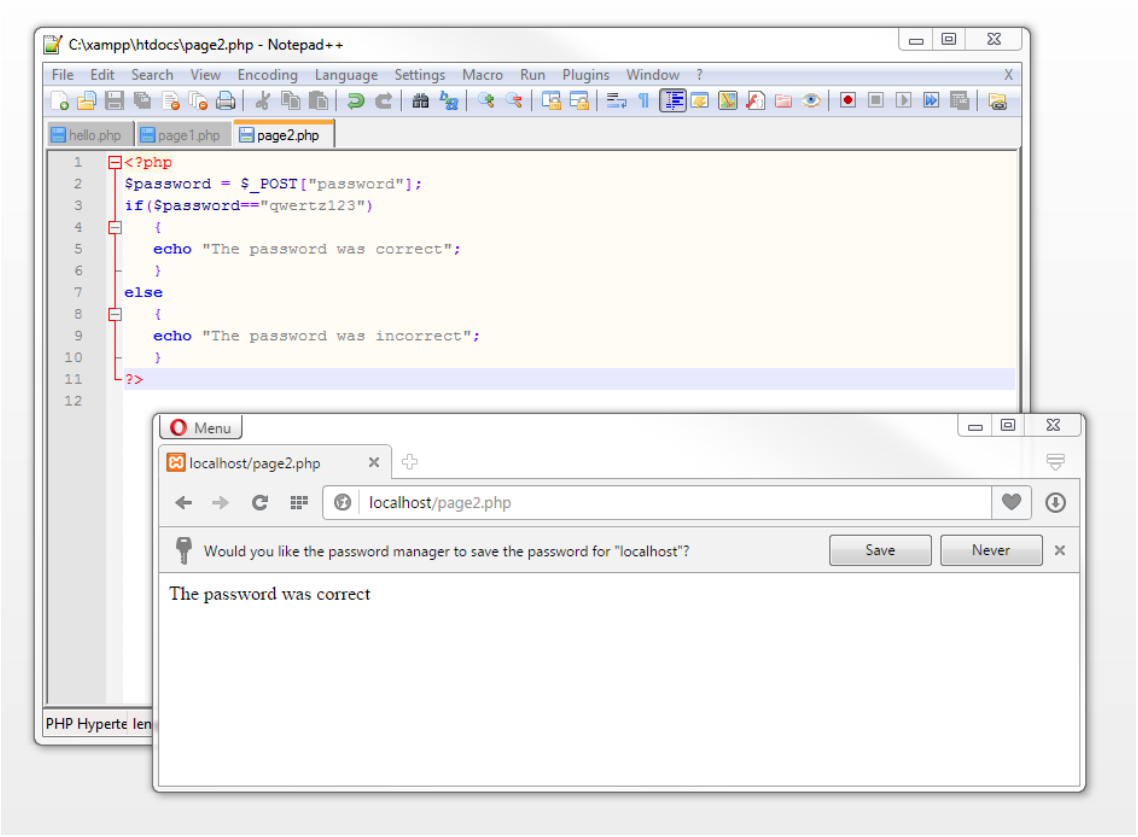
El código puede leerse de la siguiente manera: en primer lugar le asignamos un valor a la variable *$contrasena* en la línea 2, al que se accede a través del método POST de HTTP. A continuación, se define la siguiente estructura de control: la sentencia *if* de la línea 3 debe verificar si el valor de la variable *$contrasena* coincide con el string*qwertz123*. En caso de ser así, se obtiene el string *The password was correct* (*La contraseña es correcta).* Si la sentencia *if* da como resultado *FALSE,* entra en juego en la línea 7 la instrucción *else* y se emite el string *The password was incorrect* (*La contraseña es incorrecta)*.

Se puede acceder al script *page1.php* a través del URL *http:// localhost/ page1.php.*




El formulario HTML exige la introducción de una contraseña.

El navegador presenta la vista web del formulario HTML para la solicitud de la contraseña. Aquí se responde a la solicitud, se introduce la contraseña *quertz123* definida en el script *page2.php* y, a continuación, se hace clic en el botón de enviar.




El script confirma que se ha introducido la contraseña correcta.

El navegador redirige automáticamente a la *page2.php*. En un segundo plano, el constructor *if* compara la contraseña introducida con la guardada y concluye que "*qwertz123 == qwertz123* is *TRUE*". Seguidamente emite el string *The password was correct* (*La contraseña es correcta)*.

Comprueba tú mismo lo que ocurre al introducir una contraseña diferente en el campo de entrada.

## Operadores lógicos

Las condiciones que se definen con ayuda de los operadores de comparación en la *expression* del constructor *if* pueden enlazarse, en caso necesario, con otras condiciones en la misma *expression.* PHP se basa, para ello, en los operadores lógicos **AND** y **OR.**

**Operadores lógicos:**

| **Vínculo estrecho** | **Vínculo débil** | **Descripción** |
|---|---|---|
| && | AND | Las dos condiciones vinculadas con el operador deben estar definidas por el valor TRUE. |
| // | OR | Una de las condiciones vinculadas por el operador tiene que ser TRUE. |

Para enlazar condiciones, PHP cuenta con **operadores lógicos que establecen vínculos estrechos o débiles**. El hecho de decidirse por una u otra opción no plantea diferencias en la práctica. Si estas se combinan, sin embargo, es fácil darse cuenta que *OR* y // establecen un vínculo más estrecho que *AND* y *OR.* Además, el vínculo creado por *AND* y *&&*es más estrecho que *OR* y //. Esto es comparable con la jerarquía de operadores, tal como se conoce por los operadores matemáticos (por ejemplo, los

operadores de multiplicación y división tienen precedencia sobre los de adición y sustracción: *** establece un vínculo más estrecho que *+*).

Un ejemplo práctico de esto es el que ofrece la **solicitud de contraseña**. Por lo general, los datos de registro se componen de una contraseña secreta y de un nombre de usuario. Solo se podrá llevar a cabo el registro en aquellos casos en los que tanto un dato como el otro coincidan con los facilitados en el sistema.

En este caso, volvemos a abrir el formulario para la solicitud de la contraseña en la *page1.php* y añadimos un campo de entrada para el nombre de usuario:

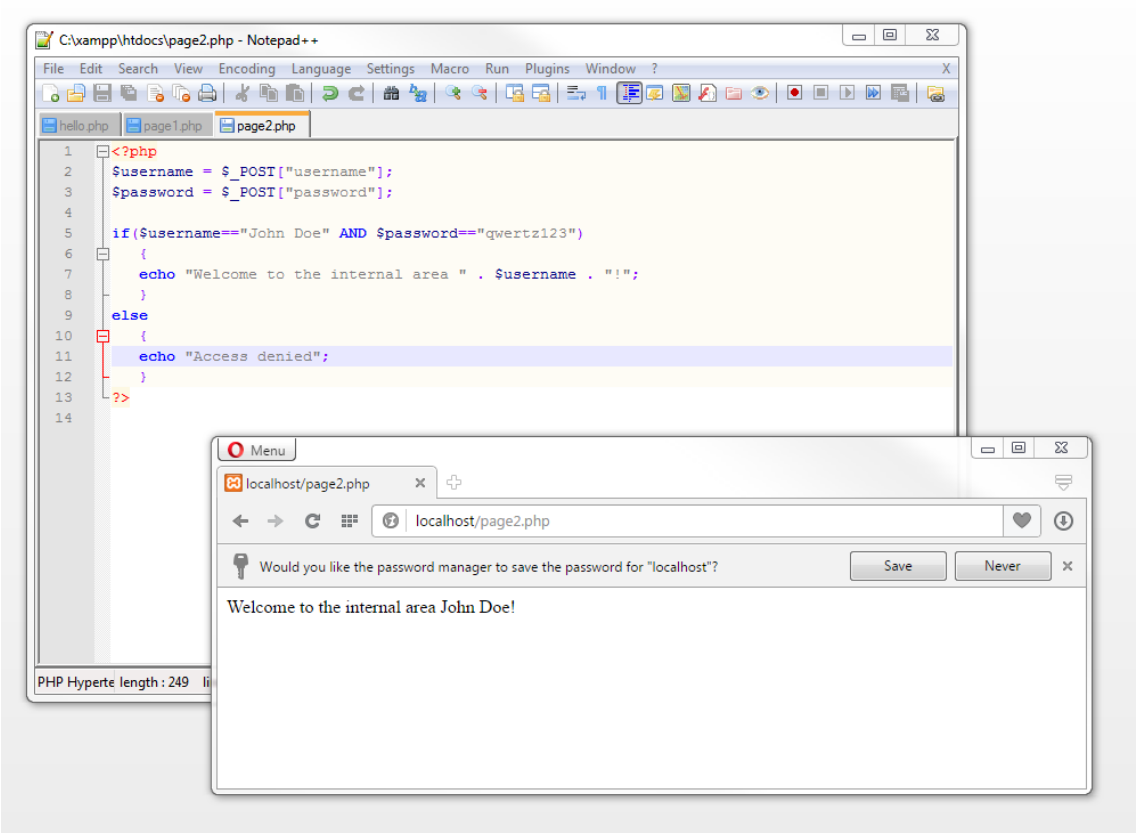
```
<form action="page2.php" method="post">
Nombre de usuario: <input type="text" name="username" /><br />
Contraseña: <input type="password" name="password" /><br />
<input type="submit" value="Enviar" />
</form>
```

En el paso siguiente hay que adaptar la estructura de control de la sentencia *if*. Para ello es necesario recurrir al operador lógico AND y enlazar, así, la condición para la solicitud de la contraseña con una condición para la solicitud del nombre de usuario.

```
<?php
$username = $_POST["username"];
$password = $_POST["password"];

if($username=="John Doe" AND $password=="qwertz123")
{
      echo "Bienvenido al área interna " . $username . "!";
}
else
{
      echo "Acceso denegado";
}
?>
```

El script *page2.php* recibe los valores para *nombre de usuario* y *contraseña* y los almacena en las variables *$username* y *$password.* La *expression* de la sentencia *if* contiene **dos condiciones que están vinculadas con el operador lógico *AND***. Solo cuando se hayan cumplido ambas condiciones (*username=="John Doe"* y *$password=="qwertz123"*), la sentencia *if* ofrecerá como resultado *TRUE*.

Debido a que el nombre de usuario se obtiene a través del campo de entrada *username*, este puede utilizarse directamente para publicar el texto vía *echo*: a *Welcome to the internal área (Bienvenido al área interna)* le sigue el valor de la variable *$username*. Si no se cumple una de las dos condiciones, se recibirá el texto *Access denied (Acceso denegado)*.




El script solo confirmará la entrada de la contraseña cuando se cumplan ambas condiciones.

Los operadores lógicos pueden combinarse libremente. En este sentido se puede considerar que *AND* tiene una jerarquía de operadores superior a *OR*. Del mismo modo que ocurre con las ecuaciones matemáticas, en PHP también se pueden utilizar paréntesis para influir en la jerarquía.

## Bucles (while, for)

A veces es necesario que un script recorra un determinado segmento de código varias veces antes de que se ejecute el resto del código del programa. Para ello, los lenguajes de programación utilizan el concepto de bucle. Aquí se pueden **diferenciar tres tipos**:

- bucles *while*
- bucles *do-while*
- bucles*for*

### Bucles while

Los bucles *while* son los bucles más sencillos de PHP. Su **estructura base** obedece al siguiente esquema:

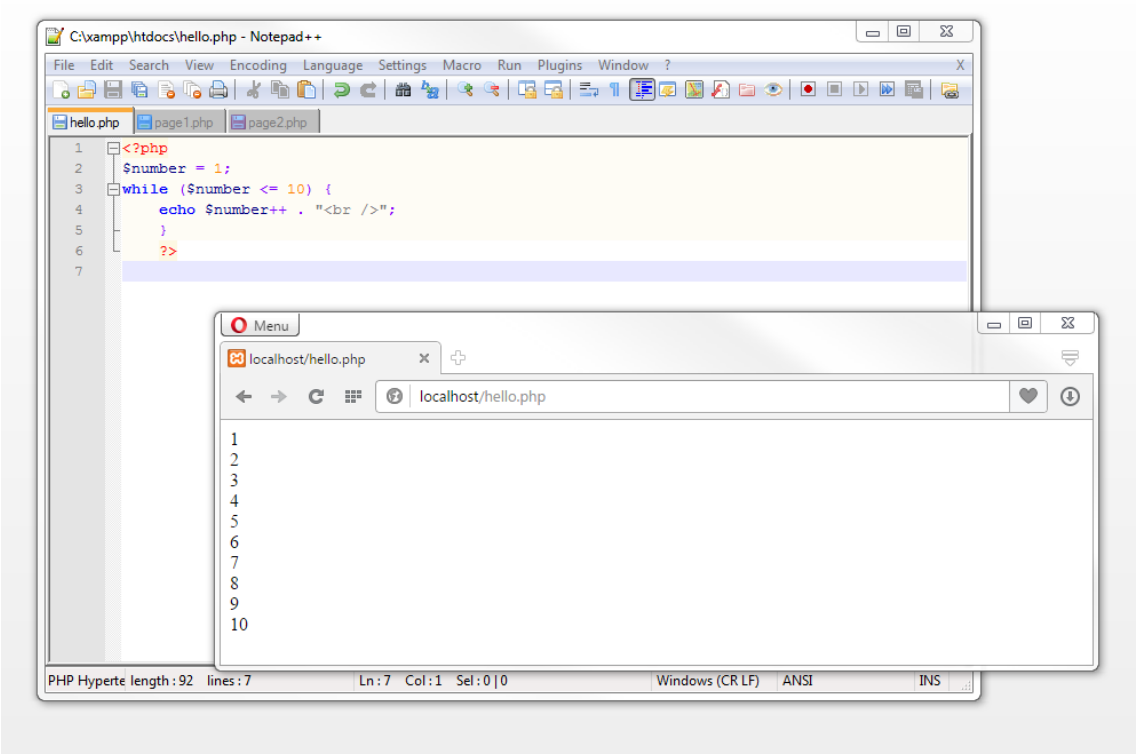
```
while (condición)
{
    Paso del bucle y otras instrucciones
}
```

Los bucles *while* instan a PHP a ejecutar instrucciones subordinadas en tanto que la **condición while** se cumpla. Para ello, el intérprete de PHP comprueba la condición al principio de cada vuelta del bucle. La ejecución del código subordinado se para cuando la condición *while* ya no se cumple.

Se puede visualizar este principio con un sencillo script con números. A continuación, la versión en español:

```
<?php
$numero = 1;
while ($numero <= 10) {
      echo $numero++ . "<br />";
}
?>
```

En el apartado “Realización de cálculos con variables” anterior ya se introdujo el principio del incremento. En el siguiente script recurrimos a dicho principio, pero utilizamos un operador de postincremento para aumentar en 1 el valor del integer de la variable *$number ($numero)* cada vez que el bucle dé una vuelta tras la publicación del texto vía *echo*. Como condición para el bucle *while* definimos que *$number* sea menor/igual a 10. La instrucción *echo* se repetirá tantas veces como sea necesario para que *$number* reciba un valor superior a 10.




El script incrementa la variable $number hasta alcanzar un valor mayor que 10. Entonces el bucle deja de ejecutarse

El resultado de la ejecución del script es un string que, cada vez que el bucle da una vuelta, emite el valor de la variable *$number* antes de que este se incremente. El script cuenta de 1 a 10 y finaliza la ejecución del código en cuanto no se cumple la condición *while*.

## Bucles do-while

La construcción de los bucles *do-while*se asemeja a la de los bucles *while*. La diferencia solo radica en que la condición no se comprueba al principio de cada vuelta del bucle, sino al final. El **esquema básico de un bucle*do-while***se corresponde con el siguiente ejemplo:

```
do {
    Paso del bucle y otras instrucciones
    }
while (Condición)
```

Cuando se programa un bucle *do-while*, el script resultante tiene el siguiente aspecto:

```
<?php
$numero = 1;
do {
echo $numero++ . "<br />";
    }
while ($numero <= 10);
  ?>
```

En este caso, el resultado es el mismo. Lo peculiar en el bucle *do-while* es que este se ejecuta al menos una vez, aun cuando la condición no se cumpla en ninguna ejecución del bucle.

### Bucles for

Básicamente, los bucles *for* tienen la misma funcionalidad en un script de PHP que los bucles *while*. A diferencia de estos, sin embargo, **el valor inicial, la condición y la instrucciónse anotarán dentro de una línea** y no se distribuirán a lo largo de tres o más líneas.

La estructura base del bucle *for* tiene el siguiente esquema:

```
for (valor inicial; condición; paso del bucle)
   instrucciones
```

Este ejemplo en español puede entenderse como bucle *for* en forma compacta:

```
<?php
for($numero = 1; $numero <= 10; $numero++) {
      echo $numero . "<br /> ";
}
?>
```

En primer lugar, se define el valor 1 para la variable *$numero*. Posteriormente, PHP comprueba si se cumple la condición *$numero <= 10.* Si es correcto, el bucle continúa y se ejecutan las instrucciones por debajo del bucle (en este caso, la instrucción *echo*). Solo entonces se ejecuta el bucle, en cuyo caso no tiene importancia si se ha optado por el preincremento o por el postincremento, puesto que esta instrucción se realiza con anterioridad a la emisión. Si el paso del bucle ha concluido, dará comienzo la siguiente vuelta.

El valor inicial, la condición o el paso del bucle son **elementos opcionales** de un bucle *for*. En teoría, son posibles incluso los bucles vacíos, aunque serían redundantes.

Básicamente, tomar una decisión acerca de si los scripts PHP deben incluir bucles *for* o *while* depende de cada uno. Existe, sin embargo, un **argumento que demuestra que los bucles *for* ganan la partida**: si se aplican bucles *for,* se pueden controlar mejor los datos fundamentales del bucle. Esto contribuye a

prevenir el peligro de escribir un bucle que funcione hasta que la memoria del intérprete esté llena. Esto ocurre en referencia al ejemplo mencionado si se olvida incrementar el valor de la variable *$numero.*

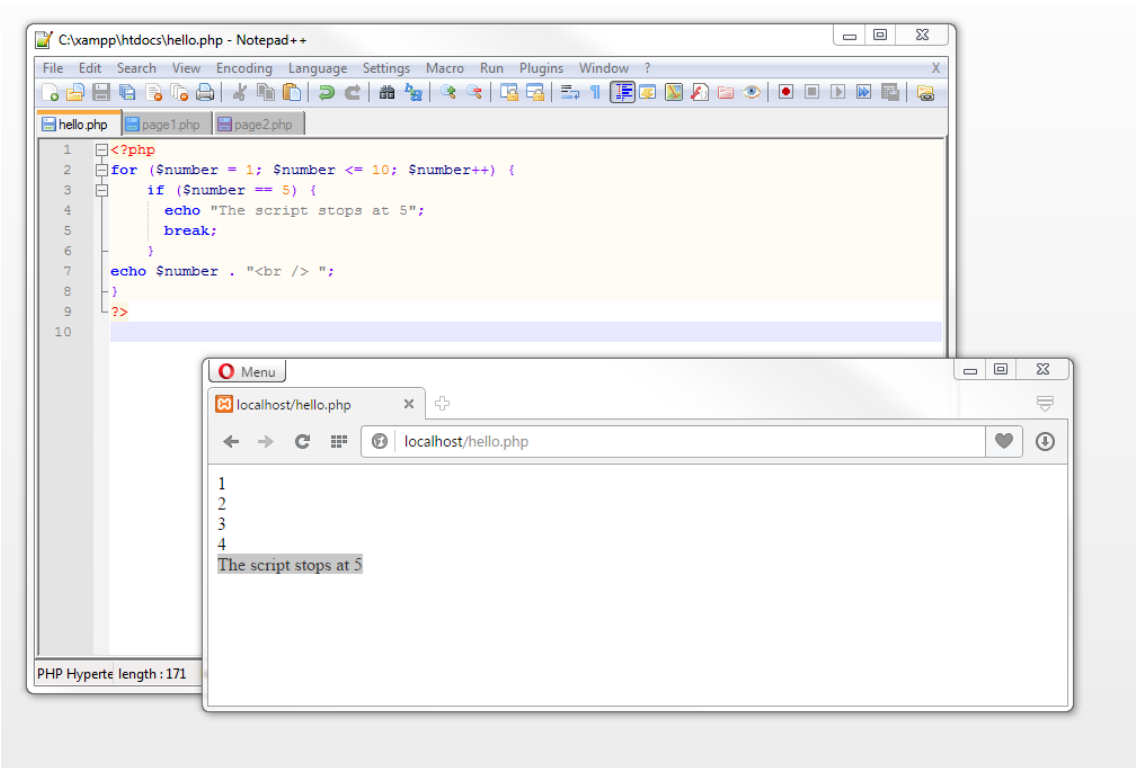
En caso de que el bucle tenga que funcionar al menos una vez independientemente de la condición, el bucle *do-while* es el bucle que habría que elegir.

### break y continue

Se puede influir en el transcurso de un bucle *while-*, *do-while* o *for* por medio de las instrucciones ***break*** y ***continue***. Se usa *break* para interrumpir el transcurso de un bucle en cualquier lugar y *continue* para omitir una vuelta del bucle. Tanto una instrucción como la otra están conectadas a una condición por medio de la instrucción *if*. En el ejemplo siguiente, tanto el original como la traducción muestran un script numérico con un *break*:

```
<?php
for ($numero = 1; $numero <= 10; $numero++) {
if ($numero== 5) {
echo "El script se para al llegar a 5";
break;
    }
echo $numero . "<br /> ";
}
?>
```

En el bucle *for* hemos definido que el valor de la variable *$numero ($number)* se incremente en 1 en cada vuelta hasta que la variable haya alcanzado el valor 10. Ahora, con la instrucción *break,* **el bucle se puede interrumpir con anticipación** en cuanto *$numero ($number)* haya alcanzado el valor 5. El constructor *echo* solo ofrece los números comprendidos entre el 1 y el 4.

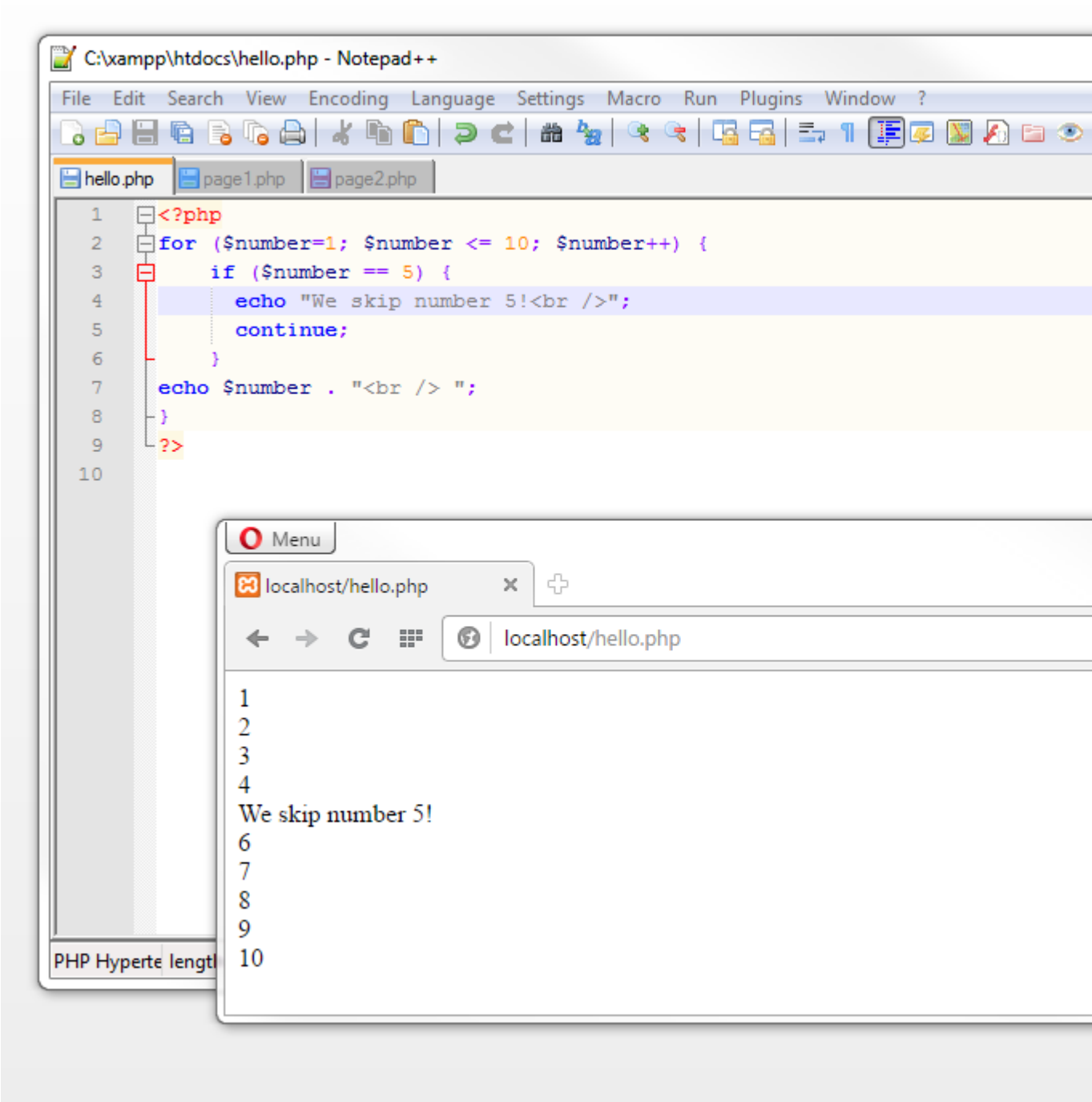



La instrucción break interrumpe el bucle en cuanto se cumple la condición if.

Para **omitir la edición de la quinta vuelta**, pero no interrumpir para ello todo el transcurso del bucle, se reemplaza la instrucción *break* por *continue*:

```
<?php
for ($numero=1; $numero <= 10; $numero++) {
if ($numero == 5) {
echo "¡Se omite la 5!<br />";
continue;
    }
echo $numero . "<br /> ";
}
?>
```

En lugar del número 5, PHP ofrece el string textual definido bajo *if ¡Se omite la 5!* (*We skip number 5!*).




El bucle for omite una vuelta por medio de la instrucción continue.

# Funciones

Una función puede ser definida empleando una sintaxis como la siguiente:

```
function suma ($x, $y)
{
      $s = $x + $y;
      return $s;
}
$a=1;
$b=2;
$c=suma ($a, $b);
print $c;
```

Cualquier código PHP válido puede aparecer dentro de una función, incluso otras funciones y definiciones de clases.

Los nombres de las funciones siguen las mismas reglas que las demás etiquetas de PHP. Un nombre de función válido comienza con una letra o guión bajo, seguido de cualquier número de letras, números o guiones bajos. Como expresión regular se expresaría así: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*.

## Valores por referencia

Por defecto, los argumentos de las funciones son pasados por valor (así, si el valor del argumento dentro de la función cambia, este no cambia fuera de la función). Para permitir a una función modificar sus argumentos, éstos deben pasarse por referencia.

Para hacer que un argumento a una función sea siempre pasado por referencia hay que anteponer al nombre del argumento el signo 'et' (&) en la definición de la función:

```
function incrementa (&$a)
{
      $a = $a + 1;
}
$a=1;
incrementa ($a);
print $a; // Muestra un 2
```

## Argumentos por defecto

Se permite el uso de argumentos por defecto o predeterminados, es decir, que se define un argumento para la función y al mismo tiempo se le asigna un valor que se usará por defecto en caso de no incluir un valor para ese parámetro en el uso de la función.

```
function muestranombre ($titulo = "Sr.")
{
      print "Estimado $titulo:\n";
}
muestranombre ();
muestranombre ("Prof.");
```

En este ejemplo hemos creado una función que acepta un parámetro llamado $título, en caso de no especificar ningún valor para $titulo se pondrá por defecto “Sr.”. esta ejecución daría como salida:

```
Estimado Sr.:
Estimado Prof.:
```

Los argumentos con valores por defecto deben ser siempre los últimos:

```
function muestranombre ($nombre, $titulo= "Sr.")
{
      print "Estimado $titulo $nombre:\n";
}
muestranombre (“Fernández”);
muestranombre (“Fernández”, "Prof.");
```

Esto daría como salida:

```
Estimado Sr. Fernández:
Estimado Prof. Fernández:
```

## Arrays

Un array en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia *valores* con *claves*. Este tipo se optimiza para varios usos diferentes; se puede emplear como un array, lista (vector), tabla asociativa (tabla hash - una implementación de un mapa), diccionario, colección, pila, cola, y posiblemente más. Ya que los valores de un array pueden ser otros arrays, también son posibles árboles y arrays multidimensionales.

### Construcción

Un array puede ser creado con el constructor del lenguaje array(). Éste toma cualquier número de parejas clave => valor como argumentos.

```
array(
clave  => valor,
clave2 => valor2,
clave3 => valor3,
...
)
```

La coma después del último elemento del array es opcional, pudiéndose omitir. Esto normalmente se hace para arrays de una única línea, es decir, es preferible array(1, 2) que array(1, 2, ). Por otra parte, para arrays multilínea, la coma final se usa frecuentemente, ya que permite una adición más sencilla de nuevos elementos al final.

A partir de PHP 5.4 también se puede usar la sintaxis de array corta, la cual reemplaza array() con [].

```
Ejemplo #1 Un array simple
<?php
```

```
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// a partir de PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

## Acceso

La clave puede ser un integer o un string, de hecho, los arrays de PHP pueden contener claves integer y string al mismo tiempo ya que PHP no distingue entre arrays indexados y asociativos. El valor puede ser de cualquier tipo.

Ejemplos:

```
$color = array ('rojo'=>101, 'verde'=>51, 'azul'=>255);
$medidas = array (10, 25, 15);
```

Acceso:

```
$color['rojo'] // No olvidar las comillas
$medidas[0]
```

El primer elemento es el 0

```
Ejemplo #3 Claves mixtas integer y string
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

La clave es opcional. Si no se especifica, PHP usará el incremento de la clave de tipo integer mayor utilizada anteriormente.

```
Ejemplo #4 Arrays indexados sin clave
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hello"
  [3]=>
  string(5) "world"
}
```

Es posible especificar la clave sólo para algunos elementos y excluir a los demás:

```
Ejemplo #5 Claves no en todos los elementos
<?php
$array = array(
         "a",
         "b",
     6 => "c",
         "d",
);
var_dump($array);
?>
```

El resultado del ejemplo sería:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Como se puede ver, al último valor "d" se le asignó la clave 7. Esto es debido a que la mayor clave integer anterior era 6.

### Acceso a elementos de array con la sintaxis de corchete.

Los elementos de array se pueden acceder utilizando la sintaxis array[key].

```
Ejemplo #6 Acceso a elementos de un array
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

El resultado del ejemplo sería:

```
string(3) "bar"
int(24)
string(3) "foo"
```

### Creación/modificación con la sintaxis de corchete

Un array existente puede ser modificado estableciendo explícitamente valores en él.

Esto se realiza asignando valores al array, especificando la clave entre corchetes. Esta también se puede omitir, resultando en un par de corchetes vacíos ([]).

```
    $arr[clave] = valor;
    $arr[] = valor;
    // clave puede ser un integer o un string
    // valor puede ser cualquier valor de cualquier tipo
```

Si $arr aún no existe, se creará, siendo también esta forma una alternativa de crear un array. Sin embargo, se desaconsejada esta práctica porque que si $arr ya contiene algún valor (p.ej. un string de una variable de petición), este estará en su lugar y [] puede significar realmente el operador de acceso a cadenas. Siempre es mejor inicializar variables mediante una asignación directa.

Para cambiar un valor determinado se debe asignar un nuevo valor a ese elemento empleando su clave. Para quitar una pareja clave/valor, se debe llamar a la función unset() con éste.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // Esto es lo mismo que $arr[13] = 56;
                // en este punto de el script

$arr["x"] = 42; // Esto agrega un nuevo elemento a
                // el array con la clave "x"

unset($arr[5]); // Esto elimina el elemento del array

unset($arr);    // Esto elimina el array completo
?>
```

Como se mencionó anteriormente, si no se especifica una clave, se toma el máximo de los índices integer existentes, y la nueva clave será ese valor máximo más 1. Si todavía no existen índices integer, la clave será 0 (cero).

Tenga en cuenta que la clave integer máxima utilizada para éste no es necesario que actualmente exista en el array. Ésta sólo debe haber existido en el array en algún momento desde la última vez que el array fué re-indexado. El siguiente ejemplo ilustra este comportamiento:

```
<?php
// Crear un array simple.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Ahora elimina cada elemento, pero deja el mismo array intacto:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Agregar un elemento (note que la nueva clave es 5, en lugar de 0).
$array[] = 6;
print_r($array);

// Re-indexar:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
```

```
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

### Acceso a elementos de un array multidimensional

Primero se coloca la fila dentro de corchetes y luego la columna dentro de corchetes.

Ejemplo:

```
<?php
$libros = array();
$libros[0] = array('titulo'=>'El mundo y sus demonios', 'autor'=>'Carl
Sagan');
$libros[1]  =  array('titulo'=>'Comer  Sin  Miedo',  'autor'=>'J.  M.
Mulet');
echo $libros[1]['autor'];
// devuelve el valor J. M. Mulet, ya que se especifica la fila numero
1 y la columna autor
?>
```

### Añadir elementos a un array

Para agregar un elemento **al final** de un array se utiliza la función **array_push().**

Ejemplo:

```
<?php
$figuras = array('cuadrado', 'triángulo', 'circulo');
array_push($figuras, 'pentágono');
// quedando ('cuadrado', 'triángulo', 'circulo', 'pentágono')
?>
```

Para insertar **al principio** del array se emplea la función **array_unshift().**

Ejemplo:

```
<?php
$figuras = array('cuadrado', 'triángulo', 'circulo');
array_unshift($figuras ,'pentágono');
// resultando ('pentágono', 'cuadrado', 'triángulo', 'circulo')
?>
```

Se pueden **eliminar el primero y el último** de los valores del array con las funciones **array_pop()** y **array_shift()** respectivamente. Ejemplo:

```
<?php
$posiciones = array('Primera', 'Segunda', 'Tercera', 'Ultima');
array_pop($posiciones);
// el array queda así array('Primera', 'Segunda', 'Tercera')
array_shift($posiciones);
// el array queda así array('Segunda', 'Tercera')
?>
```

### Recorrer un array secuencialmente

Se pueden recorrer todos los elementos de un array para leerlos o editarlos. El bucle **foreach()** funciona como un while o un for, para los array. Hay **dos formas de uso**:

Asignar el valor actual del array a una variable que se puede acceder solo dentro del bucle, **foreach( $array as $elemento)**. Ejemplo:

```
<?php
$colores = array('rojo', 'amarillo', 'azul');
foreach( $colores as $color){
      echo 'Color actual '. $color ;
}
// Esto imprimirá lo siguiente
// Color actual rojo
// Color actual amarillo
// Color actual azul
?>
```

La segunda forma es para los **arrays asociativos**, permite obtener en una variable el valor actual y en otra la clave del valor, **foreach ($array as $clave => $elemento).** Ejemplo

```
<?php
$persona = array('nombre'=>'Hipatia',
            'apellido'=>'Pi',
            'direccion'=>'calle ciencia',
            'nacionalidad'=>'egipcia');

foreach($persona as $clave => $elemento){
      echo 'Clave - '. $clave;
      echo 'Elemento - '. $elemento;
}
// Esto imprimirá lo siguiente
// Clave - nombre
// Elemento - Hipatia
// Clave - apellido
// Elemento - Pi
// Clave - dirección
// Elemento - calle ciencia
// Clave - nacionalidad
```

```
// Elemento - griega
?>
```

Ejemplo de lectura de elementos de un array, mediante **for**

```
<?php
//Ejemplo arrays
$array[0] = "Uno";
$array[1] = "Dos";
$array[2] = "Tres";
$array[3] = "Cuatro";
$array[4] = "Cinco";
$array[5] = "Seis";
$array[6] = "Siete";
$array[7] = "Ocho";
for($i=0;$i<count($array);$i++) {
      echo $array[$i].'<br />';
}
?>
```

### Otras funciones para arrays

- **sort():** Sirve para ordenar un array no asociativo, recibe dos parámetros. El primero es el array que se va a modificar y el segundo es el tipo de algoritmo que se el va aplicar (SORT_NUMERIC, SORT_STRING, SORT_REGULAR, estos son los más utilizados)
- **ksort():** Es lo mismo que sort pero funciona para ordenar arrays asociativos por la clave
- **asort():** Es lo mismo que sort pero funciona para ordenar arrays asociativos por el valor
- **print_r():** Imprime todos los elementos del array que recibe como parámetro
- **in_array():** Busca un valor(primer parámetro) en un array (segundo parametro) y devuelve True si lo consigue o False si no
- **array_keys():** Recibe como parámetro un array asociativo y retorna un array solo con las claves
- **array_search():** Busca un valor (primer parámetro) en un array (segundo parámetro) y devuelve la posición si lo consigue
- **array_count_values(): C**alcula la frecuencia de cada uno de los elementos de un array
- **count():** Cuenta los elementos de un array

# Algunas funciones interesantes

## Operaciones con archivos

Las páginas web dinámicas se basan en la separación entre contenido y presentación, de ahí que los lenguajes de programación como PHP ofrezcan diferentes funcionalidades que permiten cargar contenidos de fuentes de datos externas en archivos de plantillas centrales. En la práctica, estas fuentes de datos son bases de datos que se administran con ayuda de sistemas de gestión de contenidos como MySQL.

Además, existe la posibilidad de integrar **datos procedentes de archivos**. A continuación te mostramos cómo se pueden leer archivos como string en un script PHP y cómo se pueden guardar los textos de tus scripts en archivos.

## Lectura de archivos

Para leer el contenido de un archivo, PHP ofrece diferentes funciones, de las cuales ***file()***y ***file_get_contents()*** son las más apropiadas para nuestros objetivos. Mientras que la función *file_get_contents()* sirve para leer la totalidad del contenido de un archivo en un string, la función *file()* guarda el contenido en forma de array. Cada elemento del array se corresponde con una línea del archivo. Mediante *file()* es más sencillo emitir cada línea por separado.

A continuación, demostramos las operaciones con archivos PHP con el archivo example.txt que alojamos en el directorio *htdocs*de nuestro servidor de prueba. El contenido del archivo lo componen cuatro líneas del texto “Loremipsum”:

*Lorem ipsum dolor sitamet, consectetueradipiscingelit.*

*Aenean commodo ligulaeget dolor. Aeneanmassa.*

*Cum sociisnatoquepenatibus et magnisdisparturient montes, nasceturridiculus mus.*

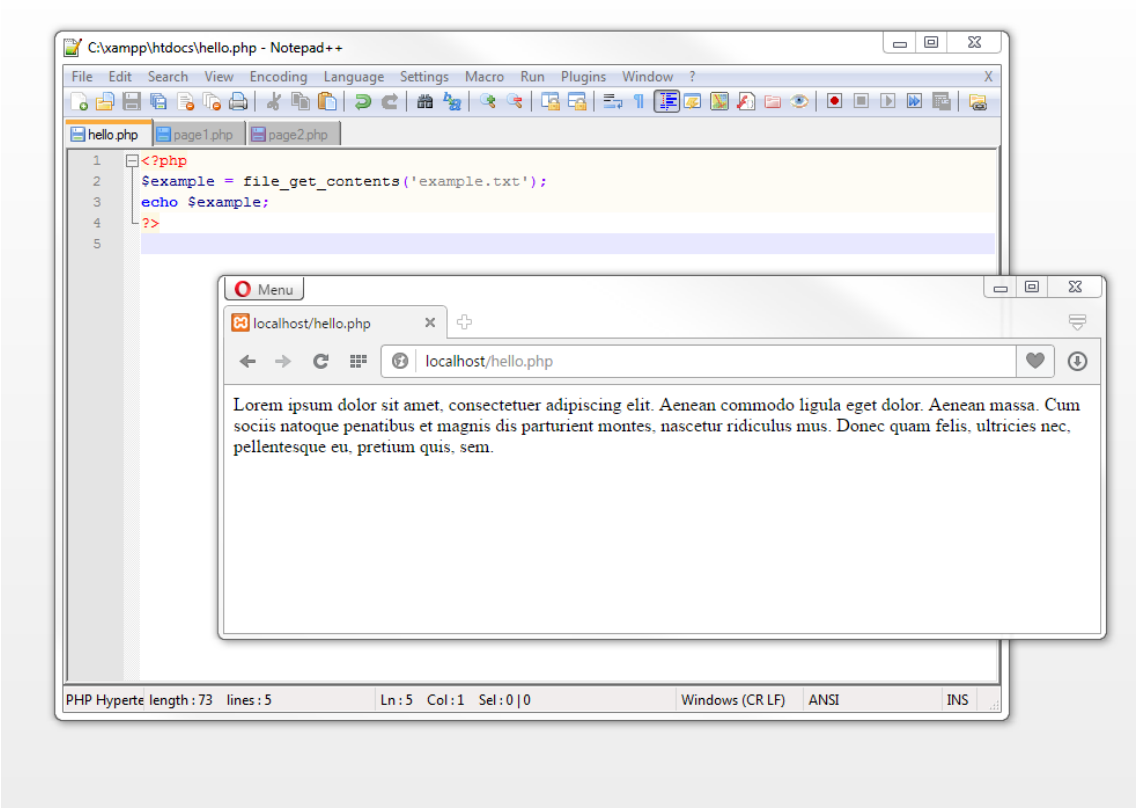
*Donecquamfelis, ultriciesnec, pellentesqueeu, pretiumquis, sem.*

El primer paso consiste en leer todo el archivo como string. Para ello hay que asignarle a la función *file_get_contents()* el nombre del archivo correspondiente como parámetro. Esto tiene lugar en función del esquema siguiente:

*file_get_contents('example.txt')*

Ahora ya podemos trabajar con este string ya leído, al que podemos, por ejemplo, asignar una variable y emitirla como texto en el navegador web:

```
<?php
$example = file_get_contents('example.txt');
echo $example;
?>
```



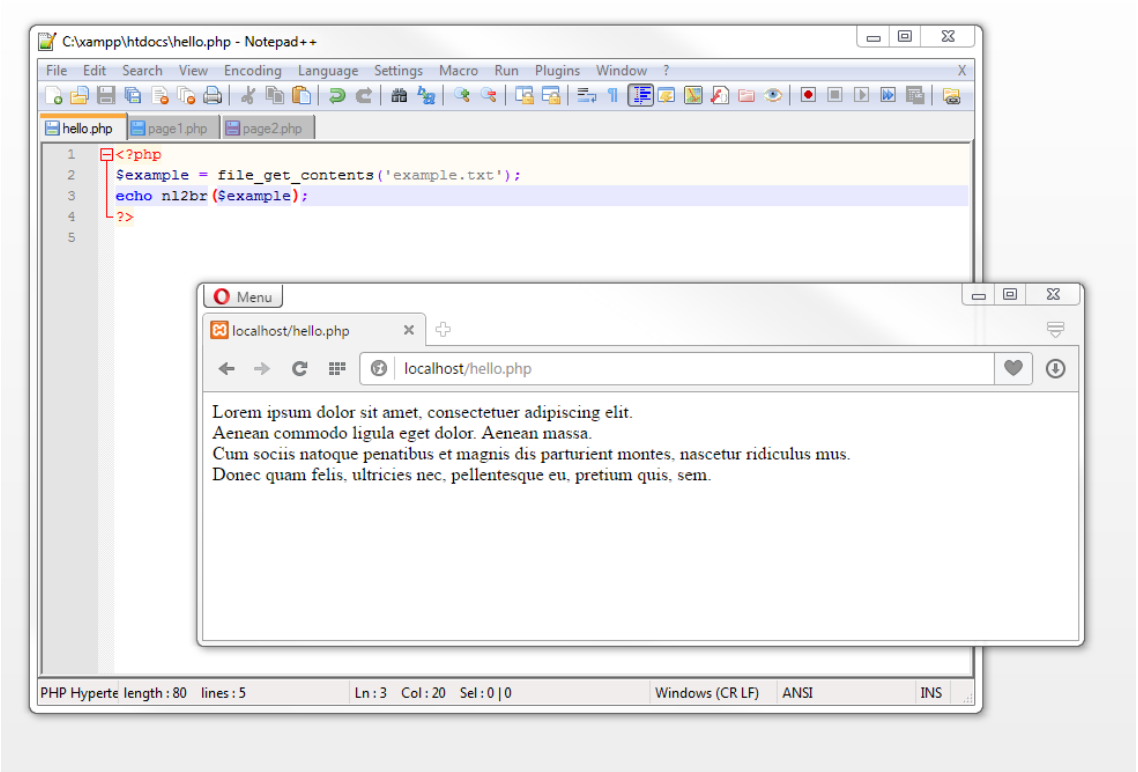

El archivo leído se emite como texto en el navegador.

En la vista del navegador se puede ver el texto devuelto sin párrafos. Los saltos de línea del archivo original no se pueden visualizar, lo que guarda relación con el hecho de que el navegador web interpreta el texto del script como código HTML, por lo que se pierden los ajustes que se definen en los editores de texto.

Si quieres mantener la estructura original, puedes recurrir a diferentes posibilidades. Puedes añadir manualmente la codificación HTML para el salto de línea (*br*) en el archivo de origen, etiquetar con *<pre>*el contenido del archivo y asignar la propiedad CSS *white-space: pre-wrap* a esta sección o puedes utilizar la función *nl2br()* para indicar a PHP que los saltos de línea (*new lines)* deben transformarse automáticamente en saltos de línea en HTML (*breaks*). Se ha de utilizar la función siguiendo el siguiente esquema:

```
<?php
$example = file_get_contents('example.txt');
echo nl2br($example);
?>
```

Si se emplea el constructor del lenguaje *echo* en combinación con *nl2br()*, PHP inserta un salto de línea en HTML antes de cada línea.




La función nl2br() sirve de ayuda para la estructuración de los datos leídos.

Para que las **líneas de un archivo se visualicen por separado,** puedes recurrir a la función *file()*, la cual lee el archivo, numera todas las líneas que comienzan por 0 y guarda sus contenidos como elementos de un array. Trasladado a nuestro ejemplo, se obtiene la siguiente clasificación:

[0] = *Loremipsum dolor sitamet, consectetueradipiscingelit.*

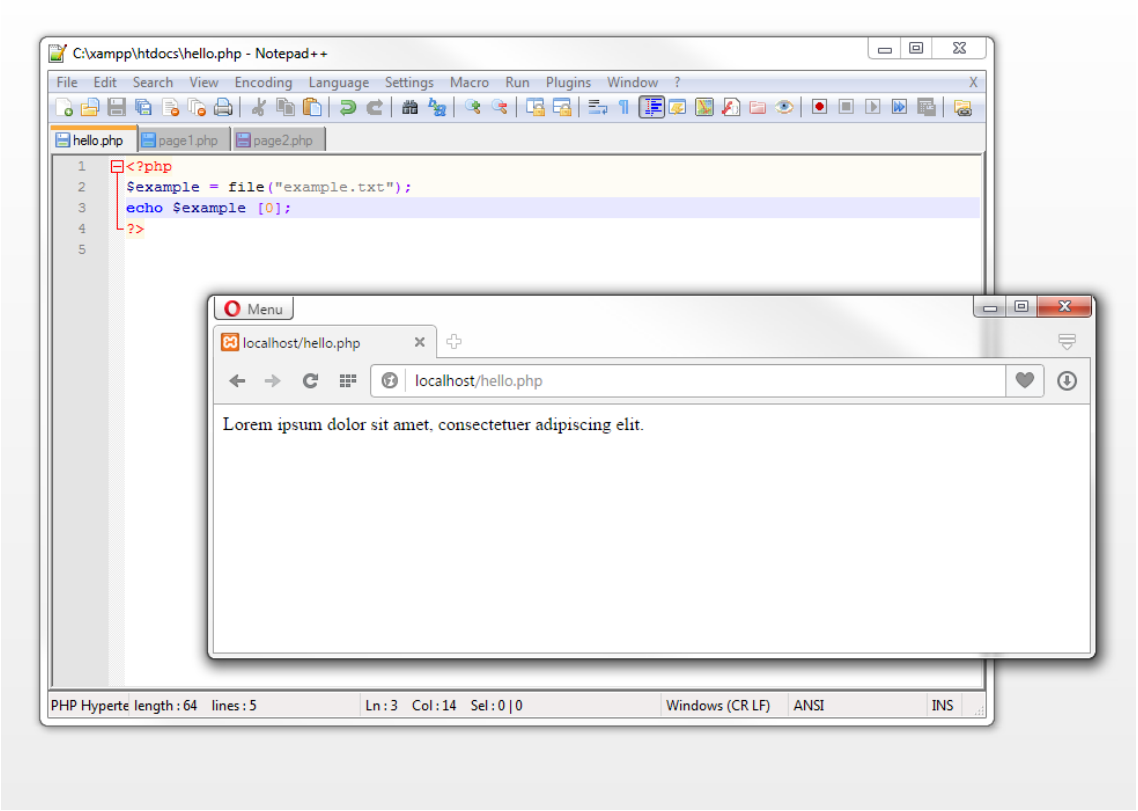
[1] = *Aenean commodo ligulaeget dolor. Aeneanmassa.*

[2] = *Cum sociisnatoquepenatibus et magnisdisparturient montes, nasceturridiculus mus.*

[3] = *Donecquamfelis, ultriciesnec, pellentesqueeu, pretiumquis, sem.*

Para emitir el contenido correspondiente mediante el constructor del lenguaje *echo*, es necesario indicar el número de línea deseado. Así, el siguiente script de ejemplo entrega al navegador la primera línea del archivo *example.txt*:

```
<?php
$example = file("example.txt");
echo $example [0];
?>
```




Escoge el elemento del array que quieres publicar.

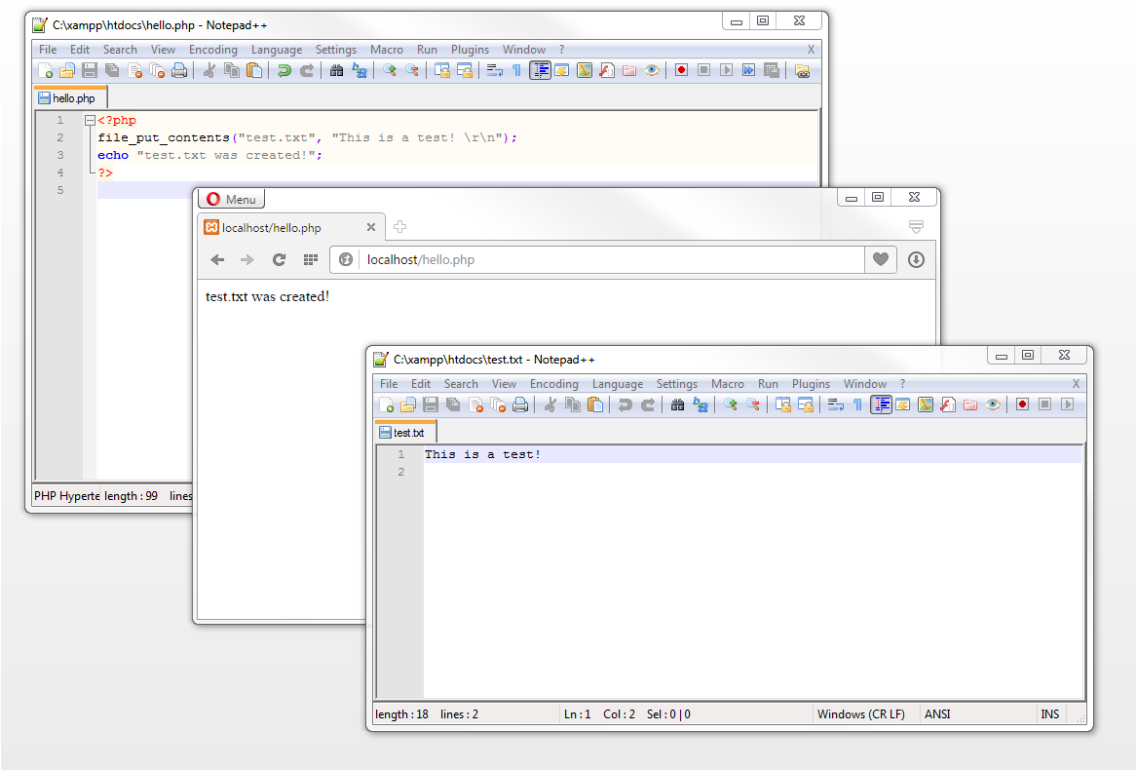
## Escritura de archivos

Con PHP no solo se pueden leer archivos, sino que el lenguaje de programación también da la posibilidad de crearlos y de describirlos con contenidos.

Para ello, se utiliza la función de PHP **file_put_contents()**, que espera **dos parámetros:** el nombre del archivo que se tiene que crear o actualizar y los datos en forma de string o array.

El siguiente script crea el fichero test.txt y escribe el string This is a test! (¡Esto es una prueba!) en la primera línea. El suplemento \r\n da lugar a un salto de línea en el archivo de destino. Veámoslo:

```
<?php
file_put_contents("test.txt", "This is a test! \r\n");
echo "test.txt was created!";
?>
```



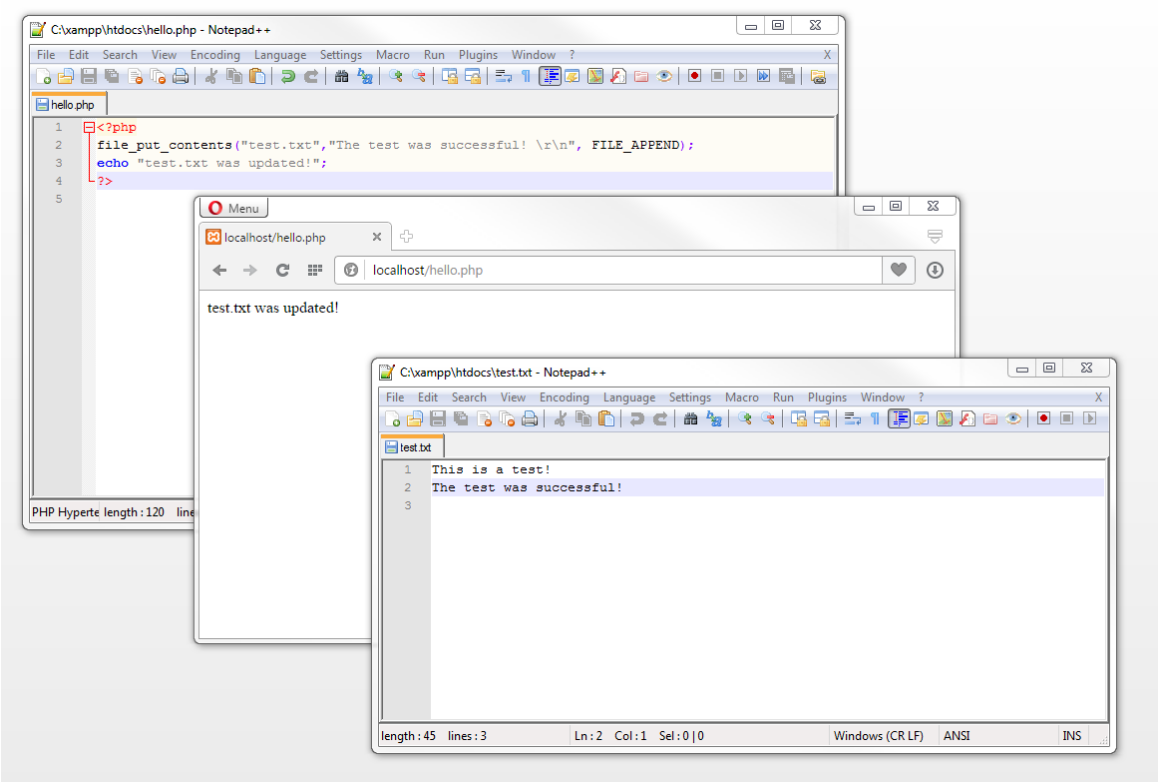

El script de PHP escribe el string Thisis a test! en el archivo test.txt.

Puesto que la función file_put_contents no entrega emisión alguna para el navegador, añadimos una instrucción echo que nos indica cuál es la acción que se va a llevar a cabo.

Si la carpeta de destino ya contiene un archivo con el mismo nombre, este se sobrescribirá, lo que se evita determinando el parámetro FILE_APPEND. Veamos la traducción de la captura de pantalla correspondiente al español:

```
<?php
file_put_contents("test.txt","The test was successful! \r\n",
FILE_APPEND);
echo "test.txt was updated!";
?>
```

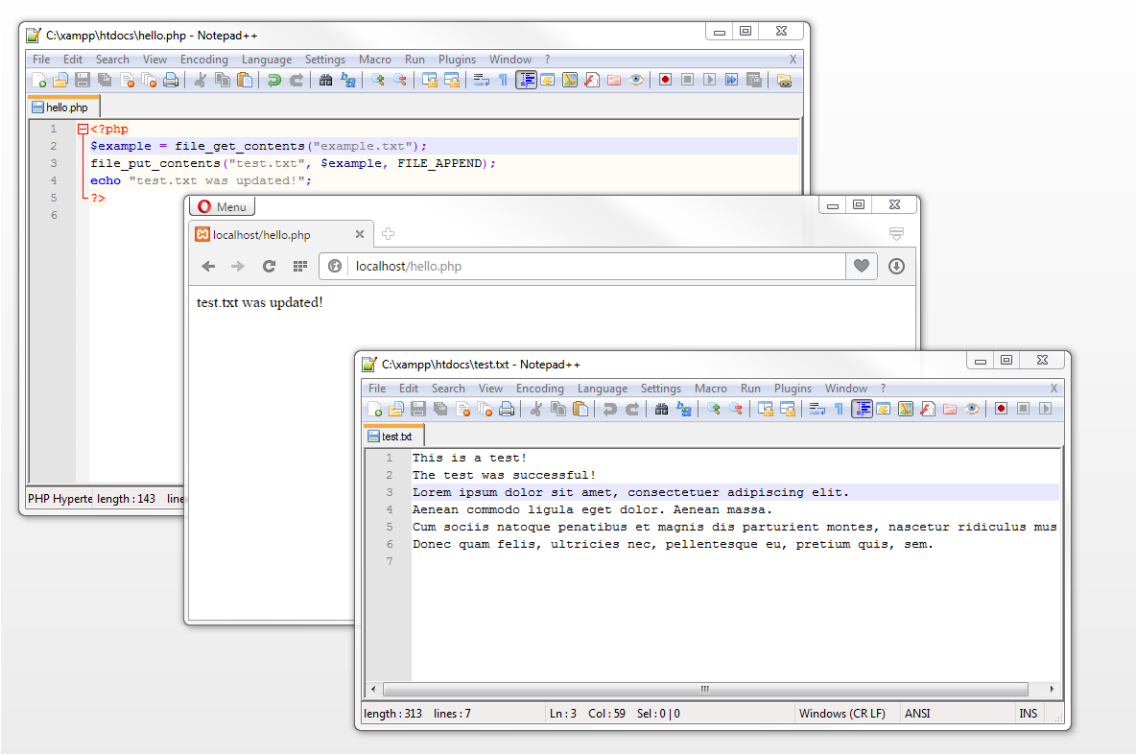
Si se utiliza file_put_contents() con el parámetro FILE_APPEND, se agregarán nuevos contenidos a los ya existentes.




El script de PHP agrega el string The test was successful! al archivo test.txt.

Lo que PHP escribe en el archivo de destino no tiene por qué ir definido obligatoriamente en el script. Como alternativa también existe la posibilidad de **transmitir los contenidos de un archivo a otro**. El siguiente script lee el contenido del fichero example.txt y lo inserta en el fichero test.txt:

```
<?php
$example= file_get_contents("example.txt");
file_put_contents("test.txt", $example, FILE_APPEND);
echo "test.txt was updated!";
?>
```




El script de PHP lee los datos del fichero example.txt y los añade al fichero test.txt.

## Funciones de directorio

Para funciones relacionadas, como dirname(), is_dir(), mkdir(), y rmdir(), visita Filesystem sección.

- chdir — Cambia de directorio
- chroot — Cambia el directorio raíz
- closedir — Cierra un gestor de directorio
- dir — Devuelve una instancia de la clase Directory
- getcwd — Obtiene el directorio actual en donde se está trabajando
- opendir — Abre un gestor de directorio
- readdir — Lee una entrada desde un gestor de directorio
- rewinddir — Regresar el gestor de directorio
- scandir — Enumera los ficheros y directorios ubicados en la ruta especificada

Ejemplo opendir()

```
<?php
$dir = "/etc/php5/";

// Abre un directorio conocido, y procede a leer el contenido
if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            echo "nombre archivo: $file : tipo archivo: " . filetype($
dir . $file) . "\n";
        }
        closedir($dh);
    }
}
?>
```

El resultado del ejemplo sería algo similar a:

```
nombre archivo: . : tipo archivo: dir
nombre archivo: .. : tipo archivo: dir
nombre archivo: apache : tipo archivo: dir
nombre archivo: cgi : tipo archivo: dir
nombre archivo: cli : tipo archivo: dir
```

## Funciones de manipulación de cadenas

- **strlen**: obtiene la longitud de una cadena y devuelve un número entero

```
<?php
$cade = "cadena1";
$n = strlen($cade);
echo "La longitud de cadena1 es: $n";
?>
```

- **substr**: devuelve una subcadena de la cadena original

```
<?php
$cade = "cadena";
$sub1 = substr($cade, 2); //La numeración empieza en 0
$sub2 = substr($cade, 2, 2);
echo "La subcadena número uno es la siguiente: $sub1";
echo "La subcadena número dos es la siguiente: $sub2";
?>
```

- **str_replace**: reemplaza caracteres en una cadena. Ejemplo:

```
<?php
//Ejemplo reemplaza en cadena
$texto = "Donde dije digo digo Diego.";
echo str_replace("Diego", "recortes", $texto);
echo "<br />";
echo $texto;
?>
```

- **strtolower y strtoupper:** Transforman una cadena de caracteres en la misma cadena en minúsculas o mayúsculas respectivamente. Ejemplo:

```
<?php
//Ejemplo funciones básicas
$cadena = "EstO eS UnA cadeNA de CARActeres";
echo strtolower($cadena);
echo "<br />";
echo strtoupper($cadena);
?>
```

- **count_chars:** Sirve para contar el número de apariciones de un carácter en una cadena. Sintaxis: **count_chars ($cadena, $opcModo)**. $opcModo es opcional. Si no se especifica vale 0 por defecto. Valores permitidos:
    - 0: devolverá un array con el valor numérico ascii como índice y la frecuencia de cada carácter ascii como valor.
    - 1: devolverá un array con el valor numérico ascii como índice y la frecuencia de cada carácter que aparezca al menos una vez como valor.
    - 2: devolverá un array de caracteres que no aparecen en la cadena, con el valor numérico ascii como índice y la frecuencia de cada carácter ascii que no aparece como valor.
    - 3: devuelve una cadena que contiene todos los caracteres únicos.
    - 4: devuelve una cadena que contiene todos los caracteres no utilizados.

  Ejemplo:

```
<?php
//Ejemplo count_chars
$cadena = 'es viernes';
$miArray = count_chars ( $cadena, 1);
foreach ($miArray as $indiceNum => $veces) {
echo 'Letra: '.chr($indiceNum).' , encontrada '. $veces.' veces
';
}
?>
```

- **substr_count($cadena, $subcadena):** Devuelve el número de veces que aparece la subcadena dentro de la cadena. Ejemplo:

```
<?php
//Ejemplo subcadena
$cadena1 = 'Pedro Juan Rita María Luis Pedro Juan Rita Pedro';
$cadena2 = 'Rita';
echo 'Rita aparece '.substr_count($cadena1, $cadena2). ' veces'
?>
```

- **strcmp:** Realiza la comparación de cadenas y devuelve un valor numérico. Sintaxis:

```
if (strcmp ($cadena1 , $cadena2 ) == 0) { … }
```

  Devuelve un valor numérico que puede ser:

  - 0: cuando ambas cadenas son iguales. En caso contrario el valor devuelto es distinto de cero, por lo que si queremos saber si dos cadenas son distintas podemos usar if (strcmp ($cadena1 , $cadena2) !== 0) { … }

  - Un valor numérico menor que 0 si la cadena 1 es menor que la cadena 2

  - Un valor numérico mayor que cero si la cadena 2 es mayor que la cadena 1

  Ejemplo strcmp:

```
<?php
$var1 = "DAWEB";
$var2 = "DAWeb";
if (strcmp($var1, $var2) !== 0) {
      echo '$var1 no es igual a $var2 en una comparación que
      considera mayúsculas y minúsculas';
}
?>
```

- **str_pad:** Realiza un relleno de cadenas ampliando la cadena hasta una longitud especificada y rellenándola con el carácter o caracteres especificados. Sintaxis:

  ```
  str_pad (string $cadena, int $nuevaLongitud $opcCarRelleno,
  opcTipoDeRelleno)
  ```

  *opcCarRelleno* es opcional e indica el carácter o caracteres de relleno que se emplearán. Si no se especifica, se tomará el espacio en blanco como carácter de relleno.

  *opcTipoDeRelleno* es opcional e indica cómo se rellenará hasta alcanzar la nueva longitud: por la derecha, por la izquierda o por ambos lados. Los valores que puede tomar son: STR_PAD_RIGHT, STR_PAD_LEFT, o STR_PAD_BOTH. Si no se especifica por defecto se rellenará usando STR_PAD_RIGHT.

- **explode()**
  - Divide una cadena en subcadenas.
  - array explode (string separador, string cadena [, int limite])
- **rtrim(), ltrim(), trim()**
  - Eliminan caracteres a la derecha, a la izquierda o por ambos lados de una cadena.
  - string rtrim ( string cadena [, string lista_caracteres])
- **strstr()**
  - Busca la primera ocurrencia de una subcadena
- **strcmp() / strcasecmp()**
  - Compara dos cadenas con/sin distinción de mayúsculas

## Funciones para manejo del tiempo, fecha y hora

La función **date** devuelve una cadena formateada según el formato dado usando el parámetro de tipo integer timestamp dado o el momento actual si no se da una marca de tiempo. En otras palabras, timestamp es opcional y por defecto es el valor de time().

Sintaxis:

```
string date ( string $format [, int $timestamp = time() ] )
```

Ejemplo:

```
<?php echo date("d-m-Y H:i:s"); ?>
```

Admite los siguientes parámetros:

Para representar el año:

| Carácter de formato | Significado | Valores que devuelve |
|---|---|---|
| L (mayúscula) | Indica si un año es bisiesto (1) o no (0) | 0 o 1 |
| Y (mayúscula) | Año en formato numérico de 4 dígitos | 1000 a 9999 |
| y (minúscula) | Año en formato numérico de 2 dígitos | El que corresponda |
| o (minúscula) | Número de año según ISO-8601. En general devuelve el mismo resultado que Y | 1000 a 9999 |

Para representar la hora:

| Formato | Significado | Valores que devuelve |
|---|---|---|
| **a (minúscula)** | Incluye am o pm para indicar antes del meridiano o después del meridiano | am o pm |
| **A (mayúscula)** | Igual que a pero en mayúsculas | AM o PM |
| **B (mayúscula)** | Hora internet, también llamada Swatch Internet Time o Biel Mean Time (BMT), poco usada. | 000 a 999 |
| **g (minúscula)** | Número de hora desde 1 hasta 12 sin ceros iniciales | 1 a 12 |
| **G (mayúscula)** | Número de hora desde 0 hasta 23 sin ceros iniciales | 0 a 23 |
| **h (minúscula)** | Número de hora desde 01 hasta 12, con ceros iniciales | 01 a 12 |
| **H (mayúscula)** | Número de hora desde 00 hasta 23, con ceros iniciales | 00 hasta 23 |
| **i (minúscula)** | Minutos desde 00 hasta 59, con ceros iniciales | 00 a 59 |
| **s (minúscula)** | Segundos desde 00 hasta 59, con ceros iniciales | 00 a 59 |

Para representar la zona horaria:

| Carácter de formato | Significado | Valores que devuelve |
|---|---|---|
| **e (minúscula)** | Representa la zona horaria en inglés (p.ej. GMT, America/Denver) | Según corresponda |
| **I (letra i mayúscula)** | Indica si se está en horario de verano (0 indica que no se está, 1 que sí se está). | 0 o 1 |
| **O (letra o mayúscula)** | Indica la diferencia respecto a la zona horaria de Greenwich u hora GMT, en horas con cuatro dígitos y símbolo. | Por ejemplo +0300 o -0500 |
| **P (mayúscula)** | Indica la diferencia respecto a la zona horaria de Greenwich u hora GMT, en horas con cuatro dígitos con separador dos puntos y símbolo. | Por ejemplo +03:00 o -05:00 |
| **T (mayúscula)** | Abreviatura de la zona horaria (inglés) | Según corresponda |
| **Z (mayúscula)** | Diferencia en segundos entre la hora utilizada y la hora UTC, con signo positivo o negativo | -43200 hasta 50400 |

Además se dispone de:

La letra c minúscula para representar la fecha y hora completa en formato ISO-8601 (ej.: 2089-03-12T12:19:21+03:00).

La letra r minúscula para representar la fecha y hora completa con formato RFC-2822 (ej.: Mon, 21 Jan 2089 11:41:05 -0300).

La letra W mayúscula para representar la semana del año según ISO-8601, considerando que las semanas comienzan en lunes. Por ejemplo, un año tiene 52 semanas y con este formato podemos obtener que estamos en la semana 24 del año.

## Otras funciones para fecha, hora y tiempo PHP

PHP define otras funciones nativas para manejo de fechas, horas y tiempo

| FUNCIÓN | UTILIDAD | EJEMPLOS |
|---|---|---|
| **mktime(hora,min, seg,mes,dia,año)**<br><br>**hora: 0 a 23**<br><br>**min y seg: 0 a 59**<br><br>**mes: 1 a 12**<br><br>**dia: 1 a 31** | Devuelve la hora UTC (GMT) para una fecha local indicada, expresada esta hora GMT en segundos (número de segundos desde el uno de enero de 1970 00:00:00 GMT, valores negativos indican momentos anteriores ). Si faltan uno o más parámetros se tomarán los de la fecha local actual de derecha a izquierda. El valor obtenido depende de la hora local del servidor . | echo "Tiempo Unix para el 15-Enero-2089 a las 12h 0m 0s = " . mktime(12,0,0,1,15,2089);<br><br>// Tiempo Unix para el 15-Enero-2089 a las 12h 0m 0s = 3753975600 (el resultado variará según dónde esté localizado el servidor, porque las 12 h del 15 de enero de 2089 corresponden a distintas fechas-horas GMT según dónde se encuentre el servidor. |

| FUNCIÓN | UTILIDAD | EJEMPLOS |
|---|---|---|
| **año: 4 dígitos** | | |

| FUNCIÓN | UTILIDAD | EJEMPLOS |
|---|---|---|
| **gmmktime(hora,min, seg,mes,dia,año)**<br><br>**hora: 0 a 23**<br><br>**min y seg: 0 a 59**<br><br>**mes: 1 a 12**<br><br>**día: 1 a 31**<br><br>**año: 4 dígitos** | Devuelve la hora UTC (GMT) expresada en segundos para una fecha GMT indicada en los argumentos recibidos. El valor devuelto es el número de segundos transcurridos desde el uno de enero de 1970 00:00:00 GMT, valores negativos indican momentos anteriores a esta fecha. Si faltan uno o más parámetros se tomarán los de la fecha GMT actual de derecha a izquierda. El valor obtenido no depende del servidor, ya que la hora GMT es única. | gmmktime(12,0,0,1,15,2089)<br><br>// 3756628800 independientemente de la hora local |

## Expresiones regulares en PHP

**Serie de caracteres que forman un patrón para poder compararlo con otro grupo de caracteres.**

Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para ver las coincidencias.

Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio dialecto.

Si es la primera vez que te acercas al concepto de expresiones regulares (regex para abreviar) te animará saber que seguro que ya las has usado, aún sin saberlo, al menos en su vertiente mas básica. Por ejemplo, cuando en una ventana DOS ejecutamos dir *.* para obtener un listado de todos los archivos de un directorio, estamos utilizando el concepto de expresiones regulares, donde el patrón * coincide con cualquier cadena de caracteres.

Unos ejemplos simplificados:

*<? am // este es nuestro patrón. Si lo comparamos con:*

*am // coincide*

*panorama // coincide*

*ambicion // coincide*

*campamento // coincide*

*mano // no coincide*

*?>*

Se trata sencillamente de ir cotejando un patrón (pattern) -que en este ejemplo es la secuencia de letras 'am'- con una cadena (subject) y ver si dentro de ella existe la misma secuencia. Si existe, decimos que hemos encontrado una coincidencia (match, en inglés).

Otro ejemplo:

*patrón: el*

*el ala aleve del leve abanico*

Hasta ahora los ejemplos han sido sencillos, ya que los patrones usados eran literales, es decir que solo encontramos coincidencias cuando hay una ocurrencia exacta.

Si sabemos de antemano la cadena exacta a buscar, no es necesario quebrarse con un patrón complicado, podemos usar como patrón la exacta cadena que buscamos, y esa y no otra será la que de coincidencia. Así, si en una lista de nombres buscamos los datos del usuario pepe podemos usar pepe como patrón. Pero si además de pepe nos interesa encontrar ocurrencias de pepa y pepito los literales no son suficientes.

El poder de las expresiones regulares radica precisamente en la flexibilidad de los patrones, que pueden ser confrontados con cualquier palabra o cadena de texto que tenga una estructura conocida.

De hecho, normalmente no es necesario usar funciones de expresiones regulares si vamos a usar patrones literales. Existen otras funciones (las funciones de cadena) que trabajan mas eficaz y rápidamente con literales.

### Caracteres y meta caracteres

Nuestro patrón puede estar formado por un conjunto de caracteres (un grupo de letras, números o signos) o por meta caracteres que representan otros caracteres, o permiten una búsqueda contextual.

Los meta-caracteres reciben este nombre porque no se representan a ellos mismos, sino que son interpretados de una manera especial.

He aquí la lista de meta caracteres mas usados:

**. * ? + [ ] ( ) { } ^ $ |**

Iremos viendo su utilización, agrupándolos según su finalidad.

Meta caracteres de posicionamiento, o anclas

Los signos ^ y $ sirven para indicar donde debe estar situado nuestro patrón dentro de la cadena para considerar que existe una coincidencia.

Cuando usamos el signo ^ queremos decir que el patrón debe aparecer al principio de la cadena de caracteres comparada. Cuando usamos el signo $ estamos indicando que el patrón debe aparecer al final del conjunto de caracteres. O mas exactamente, antes de un carácter de nueva linea Asi:

*<?*

*^am // nuestro patrón*

*am // coincide*

*cama // no coincide*

*ambidiestro // coincide*

*Pam // no coincide*

*caramba // no coincide*

*am$*

*am // coincide*

*salam // coincide*

*ambar // no coincide*

*Pam // coincide*

*^am$*

*am // coincide*

*salam // no coincide*

*ambar // no coincide*

*?>*

o como en el ejemplo anterior:

patrón: ^el

el ala aleve del leve abanico

Las expresiones regulares que usan anclas solo devolveran una ocurrencia, ya que por ejemplo, solo puede existir una secuencia el al comienzo de la cadena.

patrón: el$

el ala aleve del leve abanico

Y aquí no encontramos ninguna, ya que en la cadena a comparar (la línea en este caso) el patrón “el” no está situado al final.

Para mostrar una coincidencia en este ejemplo, tendríamos que buscar “co”:

patrón: co$

con el ala aleve del leve abanico

Hemos comenzado por unos metacaracteres especiales, ya que ^ $ no representan otros caracteres, sino posiciones en una cadena. Por eso, se conocen también como anchors o anclas.

### Escapando caracteres

Puede suceder que necesitemos incluir en nuestro patrón algún meta carácter como signo literal, es decir, por si mismo y no por lo que representa. Para indicar esta finalidad usaremos un carácter de escape, la barra invertida.

Así, un patrón definido como 12\$ no coincide con una cadena terminada en 12, y sí con 12$:

*patrón: 100$*

*el ala aleve del leve abanico cuesta 100$*

*patrón: 100\$*

*el ala aleve del leve abanico cuesta 100$*

Fíjate en los ejemplos anteriores. En el primero, no hay coincidencia, porque se interpreta "busca una secuencia consistente en el número 100 al final de la cadena", y la cadena no termina en 100, sino en 100$.

Para especificar que buscamos la cadena 100$, debemos escapar el signo $: 100\$

Como regla general, la barra invertida convierte en normales caracteres especiales, y hace especiales caracteres normales.

### El punto . como metacaracter

Si un meta carácter es un carácter que puede representar a otros, entonces el punto es el meta carácter por excelencia. Un punto en el patrón representa cualquier carácter excepto nueva línea.

Y como acabamos de ver, si lo que queremos buscar en la cadena es precisamente un punto, deberemos escaparlo: .

*patrón: '.l'*

*el ala aleve del leve abanico*

Observa en el ejemplo anterior como el patrón es cualquier carácter (incluido el de espacio en blanco) seguido de una l.

### Metacaracteres cuantificadores

Los metacaracteres que hemos visto ahora nos informan si nuestro patrón coincide con la cadena a comparar. Pero ¿y si queremos comparar con nuestra cadena un patrón que puede estar una o mas veces, o puede no estar? Para esto usamos un tipo especial de meta caracteres: los multiplicadores.

Estos metacaracteres que se aplican al carácter o grupo de caracteres que les preceden indican en que número deben encontrarse presentes en la cadena para que haya una ocurrencia.

Por ello se llaman cuantificadores o multiplicadores. Los mas usados son * ? +

*<?*

** // coincide si el carácter (o grupo de caracteres) que le*

*// precede esta presente 0 o mas veces*

*// ab* coincide con “a”, “ab”, “abbb”, etc.*

*//ejemplo:*

*cant*a // coincide con canta, cana, canttta*

*? // coincide si el carácter (o grupo de caracteres) que precede*

*// esta presente 0 o 1 vez*

*// ab? coincide con “a”, “ab”, no coincide con “abb”*

*// ejemplo:*

*cant?a // coincide con canta y cana*

*d?el // coincide con del y el*

*(ala)?cena // coincide con cena y alacena*

*+ // coincide si el carácter (o grupo) que le precede está*

*// presente al menos 1 o mas veces.*

*// ab+ coincide con “ab”, “abbb”, etc. No coincide con “a”*

*//ejemplo:*

*cant+a // coincide con canta, cantttta, NO coincide con*

*// cana ?>*

Además de estos cuantificadores sencillos también podemos especificar el numero de veces máximo y mínimo que debe darse para que haya una ocurrencia:

*<?*

*{x,y} // coincide si la letra (o grupo) que le precede esta presente*

*// un minimo “x” veces y como máximo “y” veces*

*// “ab{2}” coincide con “abb”: exactamente dos ocurrencias de “b”*

*// “ab{2,}” coincide con “abb”, “abbbb” … Como mínimo dos*

*// ocurrencias de b, máximo indefinido*

*// “ab{3,5}” coincide con “abbb”, “abbbb”, o “abbbbb”: Como minimo*

*// dos ocurrencias, como máximo 5*

*a{2,3} // coincide con casaa, casaaa*

*a{2, } // coincide con cualquier palabra que tenga al*

*// menos dos “a” o mas: casaa o casaaaaaa, no con casa*

*a{0,3} // coincide con cualquier palabra que tenga 3 o*

*// menos letras “a”.*

*// NOTA: puedes dejar sin especificar el valor máximo. NO*

*// puedes dejar el valor inicial vacío*

*a{5} // exactamente 5 letras “a”*

*?>*

Por tanto, los cuantificadores * + ? pueden también ser expresados así:

* equivale a {0,} (0 o mas veces)

+ equivale a {1,} (1 o mas veces)

? equivale a {0,1} (0 o 1 vez)

### Metacaracteres de rango

Los corchetes [] incluidos en un patrón permiten especificar el rango de caracteres válidos a comparar. Basta que exista cualquiera de ellos para que se de la condición:

*<?*

*[abc] // El patrón coincide con la cadena si en esta hay*

*// cualquiera de estos tres caracteres: a, b, c*

*[a-c] // coincide si existe una letra en el rango (“a”, “b” o “c”)*

*c[ao]sa // coincide con casa y con cosa*

*[^abc] // El patrón coincide con la cadena si en esta NO hay*

*// ninguno de estos tres caracteres: a, b, c*

*// Nota que el signo ^ aquí tiene un valor excluyente*

*c[^ao]sa // Coincide con cesa, cusa, cisa (etc); no coincide*

*// con casa ni cosa*

*[0-9] // Coincide con una cadena que contenga cualquier*

*// número entre el 0 y el 9*

*[^0-9] // Coincide con una cadena que NO contenga ningun*

*// número*

*[A-Z] // Coincide con cualquier carácter alfabético,*

*// en mayúsculas. No incluye números.*

*[a-z] // Como el anterior, en minúsculas*

*[a-Z] // Cualquier carácter alfabético, mayúsculas o minúsculas*

*?>*

Una cuestión a recordar es que las reglas de sintaxis de las expresiones regulares no se aplican igual dentro de los corchetes. Por ejemplo, el metacarácter ^ no sirve aquí de ancla, sino de carácter negador. Tampoco es necesario escapar todos los metacaracteres con la barra invertida. Solo será necesario escapar los siguientes metacaracteres: ] ^ –

El resto de metacaracteres pueden incluirse ya que son considerados -dentro de los corchetes- caracteres normales.

Como estos patrones se usan una y otra vez, hay atajos:

*<?*

*// atajo equivale a significado*

*d [0-9] // numeros de 0 a 9*

*D [^0-9] // el contrario de d*

*w [0-9A-Za-z] // cualquier numero o letra*

*W [^0-9A-Za-z] // contrario de w, un carácter que no*

*// sea letra ni numero*

*s [ tnr] // espacio en blanco: incluye espacio,*

// tabulador, nueva linea o retorno

*S [^ tnr] // contrario de s, cualquier carácter*

*// que no sea espacio en blanco*

*// solo regex POSIX*

*[[:alpha:]] // cualquier carácter alfabético aA – zZ.*

*[[:digit:]] // Cualquier número (entero) 0 – 9*

*[[:alnum:]] // Cualquier carácter alfanumérico aA zZ 0 9*

*[[:space:]] // espacio*

*?>*

## Metacaracteres de alternancia y agrupadores

```
<?

(xyz) // coincide con la secuencia exacta xyz

x|y // coincide si esta presente x ó y

(Don|Doña) // coincide si precede “Don” o “Doña”

?>
```

Un ejemplo típico sería una expresión regular cuyo patrón capturase direcciones url validas y con ellas generase links al vuelo:

```
<? $text = “una de las mejores páginas es http://www.blasten.com”;

$text = ereg_replace(“http:///(.*.(com|net|org))”,

“1”, $text);

print $text;

?>
```

El anterior ejemplo produciría un enlace usable, donde la url se tomaría de la retro-referencia y la parte visible de la retro-referencia 1 una de las mejores páginas es www.ignside.net

Fíjate que en el ejemplo anterior usamos dos grupos de paréntesis (anidados), por lo que se producirían dos capturas: La retro-referencia coincide con la coincidencia buscada. Para capturarla no es preciso usar paréntesis.

La retro-referencia 1 coincide en este caso con “www.blasten.com” y es capturada por el paréntesis (.*.(com|net|org))

La retro-referencia 2 coincide con “net” y se corresponde con el paréntesis anidado (com|net|org)

Ten en cuenta que esta característica de capturar ocurrencias y tenerlas disponibles para retroreferencias consume recursos del sistema. Si quieres usar paréntesis en tus expresiones regulares, pero sabes de antemano que no vas a reusar las ocurrencias, y puedes prescindir de la captura, coloca después del primer paréntesis ?:

```
<?

text = ereg_replace(“http:///(.*.(?:com|net|org))”,

“<a href=””>1</a>”, $text);

?>
```

Al escribir (?:com|net|org) el subpatron entre paréntesis sigue agrupado, pero la coincidencia ya no es capturada.

Como nota final sobre el tema, PHP puede capturar hasta 99 subpatrones a efectos de retro-referencia, o hasta 200 (en total) si buscamos subpatrones sin capturarlos.

## Un ejemplo práctico

Hemos dicho que las expresiones regulares son uno de los instrumentos mas útiles en cualquier lenguaje de programación. ¿Para que podemos usarlas? Uno de sus usos mas típicos es el de validar entradas de datos que los visitantes de una página puedan mandarnos a través de formularios html.

El ejemplo mas corriente es el de una dirección email. Imaginemos que queremos filtrar las direcciones introducidas por los visitantes, para evitar introducir en la base de datos la típica dirección basura ghghghghghghg. Todos sabemos la estructura de una dirección email, formada por la cadena nombreusuario, el signo @ y la cadena nombredominio. También sabemos que nombredominio esta formado por dos subcadenas, 'nombredomino', un '.' y un sufijo 'com', 'net', 'es' o similar.

Por tanto, la solución a nuestro problema es idear una expresión regular que identifique una dirección email valida típica, y confrontarla con la cadena (dirección email) pasada por el visitante

Por ejemplo:

```
<?

^[^@ ]+@[^@ ]+.[^@ .]+$

?>
```

Vamos a diseccionar nuestra expresión regular:

```
<?

^ // queremos decir que el primer carácter que buscamos

// debe estar al principio de la cadena a comparar.

[^@ ] // ese primer signo no debe ser ni el signo @

// ni un espacio

+ // y se repite una o mas veces

@ // luego buscamos el signo @

[^@ ]+ // Seguido de otro signo que no es un @ ni un

// espacio y se repite una o mas veces

. // Seguido de un .

[^@ .] // Seguido de un carácter que no sea ni @,

// ni espacio ni punto

+$ // Que se repite una o mas veces y el último esta

// al final de la cadena

?>
```

Y para comprobarlo en la práctica, usamos una de las funciones de php relacionadas con las expresiones regulares: **ereg()**.

Acudiendo al manual php, podemos averiguar que esta función tiene la siguiente sintaxis:

```
ereg (string pattern, string string)
```

Busca en string las coincidencias con la expresión regular pattern. La búsqueda diferencia entre mayúsculas y minúsculas.

Devuelve un valor verdadero si se encontró alguna coincidencia, o falso in no se encontraron coincidencias u ocurrió algún error. Podríamos usar esta funcion para un validador email con algo asi como:

*<?*

*// establecemos una secuencia condicional: si la variable $op no existe y*

*// es igual a "ds", se muestra un formulario*

*if ($op != "ds") { ?>*

*<form>*

*<input type=hidden name="op" value="ds">*

*Tu email:<br />*

*<input type=text name="email" value="" size="25" />*

*<input type=submit name="submit" value="Enviar" />*

*</form>*

*<?*

*}*

*// Si $op existe y es igual a "ds"m, se ejecuta la función ereg buscando*

*// nuestra cadena dentro del patrón $email que es la direccion enviada por*

*// el usuario desde el formulario anterior*

*else if ($op == "ds")*

*{*

*if (ereg("^[^@ ]+@[^@ ]+.[^@ .]+$", $email ) )*

*{*

*print "<BR>Esta dirección es correcta: $email"; }*

*else {echo "$email no es una dirección valida";}*

*}*

*?>*

No hace falta advertir que se trata de un ejemplo muy elemental, que dará por válida cualquier dirección email que tenga una mínima apariencia de normalidad (por ejemplo, daria por valida 'midireccionnn@noteimporta.commm')

Para tener a mano …

Una breve referencia de los metacaracteres y su significado, tomada de un comentario del manual de php.net.

*<?*

*^ // Comienzo de la zadena*

*$ // Final de la cadena*

*n* // Cero o mas "n" (donde n es el carácter precedente)*

*n+ // Uno o mas "n"*

*n? // Un posible "n"*

*n{2} // Exactamente dos "n"*

*n{2,} // Al menos dos o mas "n"*

*n{2,4} // De dos a cuatro "n"*

*() // Parentesis para agrupar expresiones*

*(n|a) // o "n" o "a"*

*. // Cualquier carácter*

*[1-6] // un número entre 1 y 6*

*[c-h] // una letra en minuscula entre c y h*

*[D-M] // una letra en mayúscula entre D y M*

*[^a-z] // no hay letras en minuscula de a hasta z*

*[_a-zA-Z] // un guion bajo o cualquier letra del alfabeto*

*^.{2}[a-z]{1,2}_?[0-9]*([1-6]|[a-f])[^1-9]{2}a+$*

*/* Una cadena que comienza por dos caracteres cualquiera*

*Seguidos por una o dos letras (en minuscula)*

*Seguidos por un guion _ bajo opcional*

*Seguidos por cero o mas números*

*Seguidos por un numero del 1 al 6 o una letra de la -a- a la -f-*

*Seguidos por dos caracteres que no son números del 1 al 9*

*Seguidos de uno o mas caracteres al final de la cadena*

*Tomado de una anotacion al manual de php.net, de mholdgate –*

*wakefield dot co dot uk */*

*?>*

## Funciones PHP para expresiones regulares

### ereg

(PHP 4, PHP 5)

ereg — Comparación de una expresión regular

**Advertencia**

Esta función está *OBSOLETA* en PHP 5.3.0, por lo tanto, será *ELIMINADA* en PHP 7.0.0.

Las alternativas a esta función son:

preg_match()

**Descripción**

`int` **`ereg`** `( string $pattern , string $string [, array &$regs ] )`
Busca en `string` coincidencias con la expresión regular dada en `pattern` de una forma sensible a mayúsculas-minúsculas.

**Parámetros**

- `Pattern`: Expresión regular sensible a mayúsculas-minúsculas.
- `String`: La cadena de entrada.
- `Regs`: Si se encontraron coincidencias con las sub-cadenas entre paréntesis de `pattern` y la función es llamada con el tercer argumento `regs`, las coincidencias serán almacenadas en los elementos de la matriz `regs`.

*$registros[1]* contendrá la sub-cadena que comienza con el primer paréntesis de la izquierda; *$registros[2]* contendrá la segunda sub-cadena, y así sucesivamente. *$registros[0]* contendrá una copia de la cadena coincidente completa.

**Valores devueltos**

Devuelve la longitud de la cadena coincidente si una coincidencia de `pattern` se encontró en `string`, o **`FALSE`** si no se encontraron coincidencias o se produjo un error.

Si el parámetro opcional `regs` no fue pasado o la longitud de la cadena coincidente es 0, esta función devuelve 1.

**Ejemplos**

**Ejemplo #1 Ejemplo de ereg()**

El siguiente trozo de código toma una fecha en formato ISO (AAAA-MM-DD) y la imprime en formato DD.MM.AAAA:

```
<?php
if (ereg ("([0-9]{4})-([0-9]{1,2})-([0-
9]{1,2})", $fecha, $registros)) {
    echo "$registros[3].$registros[2].$registros[1]";
} else {
    echo "Formato de fecha no válido: $fecha";
}
?>
```

Ver también ¶

eregi() - Comparación de una expresión regular de forma insensible a mayúsculas-minúsculas

ereg_replace() - Sustituye una expresión regular

eregi_replace() - Sustituye una expresión regular de forma insensible a mayúsculas-minúsculas

preg_match() - Realiza una comparación con una expresión regular

strpos() - Encuentra la posición de la primera ocurrencia de un substring en un string

strstr() - Encuentra la primera aparición de un string

quotemeta() - Escapa meta caracteres

## preg_match

(PHP 4, PHP 5, PHP 7)

Realiza una comparación con una expresión regular

**Descripción**

```
int preg_match ( string $pattern , string $subject [, array &$matches
[, int $flags = 0 [, int $offset = 0 ]]] )
```

Busca en `subject` una coincidencia con la expresión regular dada en `pattern`.

**Parámetros**

- `Pattern`: El patrón de búsqueda, como cadena.
- `Subject`: La cadena de entrada.
- `Matches`: Si se proporciona `matches`, entonces éste se llena con los resultados de la búsqueda. *$matches[0]* contendrá el texto que coincidió con el patrón completo, *$matches[1]*

tendrá el texto que coincidió con el primer sub-patrón entre paréntesis capturado, y así sucesivamente.

- `Flags: flags` puede estar seguido de la siguiente bandera:

**PREG_OFFSET_CAPTURE**

Si se pasa esta bandera, por cada coincidencia producida, el índice de la cadena añadida también será devuelto. Observe que esto cambia el valor de `matches` dentro de una matriz donde cada elemento es una matriz consistente en la cadena coincidente en el índice *0* y su índice dentro de la cadena `subject` en el índice *1*.

```
<?php
preg_match('/(foo)(bar)(baz)/', 'foobarbaz', $matches, PREG_OFFSET_
CAPTURE);
print_r($matches);
?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => Array
        (
            [0] => foobarbaz
            [1] => 0
        )

    [1] => Array
        (
            [0] => foo
            [1] => 0
        )

    [2] => Array
        (
            [0] => bar
            [1] => 3
        )

    [3] => Array
        (
            [0] => baz
            [1] => 6
        )

)
```

- `Offset:` Normalmente, la búsqueda comienza por el principio de la cadena objetivo. El parámetro opcional `offset` se puede usar para especificar el lugar alternativo desde el cual comenzar la búsqueda (en bytes).

**Nota**:

Usar `offset` no es equivalente a pasar *substr($sujeto, $índice)* a **preg_match()** en lugar de la cadena objetivo, ya que `pattern` puede contener declaraciones como ^, *$* o *(?<=x)*. Compare:

```
<?php
$sujeto = "abcdef";
$patrón = '/^def/';
preg_match($patrón, $sujeto, $coincidencias, PREG_OFFSET_CAPTURE, 3);
print_r($coincidencias);
?>
```

El resultado del ejemplo sería:

```
Array
(
)
```

mientras que este ejemplo

```
<?php
$sujeto = "abcdef";
$patrón = '/^def/';
preg_match($patrón, substr($sujeto,3), $coincidencias, PREG_OFFSET_CAP
TURE);
print_r($coincidencias);
?>
```

producirá

```
Array
(
    [0] => Array
        (
            [0] => def
            [1] => 0
        )

)
```

**Valores devueltos**

**preg_match()** devuelve 1 si `pattern` coincide con el `subject` dado, 0 si no, o **`FALSE`** si ocurrió un error.

**Advertencia**

Esta función puede devolver el valor booleano **`FALSE`**, pero también puede devolver un valor no booleano que se evalúa como **`FALSE`**. Por favor lea la sección sobre Booleanos para más información. Use el operador === para comprobar el valor devuelto por esta función.

**Ejemplos**

**Ejemplo #1 Busca la cadena de texto "php"**

```
<?php
// La "i" después del delimitador de patrón indica una búsqueda
```

```
// sin tener en cuenta mayúsculas/minúsculas
if (preg_match("/php/i", "PHP es el lenguaje de secuencias de comandos
 web preferido.")) {
    echo "Se encontró una coincidencia.";
} else {
    echo "No se encontró ninguna coincidencia.";
}
?>
```

**Ejemplo #2 Busca la palabra "web"**

```
<?php
/* El \b en el patrón indica un límite de palabra, por lo que sólo la
palabra
 * definida "web" se compara, y no una palabra parcial como "webbing"
o "cobweb" */
if (preg_match("/\bweb\b/i", "PHP es el lenguaje de secuencias de coma
ndos web preferido.")) {
    echo "Se encontró una coincidencia.";
} else {
    echo "No se encontró ninguna coincidencia.";
}

if (preg_match("/\bweb\b/i", "PHP es el lenguaje de secuencias de coma
ndos website preferido.")) {
    echo "Se encontró una coincidencia.";
} else {
    echo "No se encontró ninguna coincidencia.";
}
?>
```

**Ejemplo #3 Obtener el nombre de dominio de una URL**

```
<?php
// obtiene el nombre del host de la URL
preg_match('@^(?:http://)?([^/]+)@i',
    "http://www.php.net/index.html", $coincidencias);
$host = $coincidencias[1];

// obtiene los dos últimos segmentos del nombre del host
preg_match('/[^.]+\.[^.]+$/', $host, $coincidencias);
echo "El nombre de dominio es: {$coincidencias[0]}\n";
?>
```

El resultado del ejemplo sería:

El nombre de dominio es: php.net

**Ejemplo #4 Usar sub-patrones nominados**

```
<?php

$cadena = 'foobar: 2008';
```

```
preg_match_all('/(?P<nombre>\w+): (?P<dígito>\d+)/', $cadena, $coincid
encias);

/* Esto también funciona en PHP 5.2.2 (PCRE 7.0) y posteriores, sin em
bargo
 * la forma de arriba es la recomendada por compatibilidad con version
es anteriores */
// preg_match_all('/(?<nombre>\w+): (?<dígito>\d+)/', $cadena, $coinci
dencias);

print_r($coincidencias);

?>
```

El resultado del ejemplo sería:

```
Array
(
    [0] => foobar: 2008
    [name] => foobar
    [1] => foobar
    [digit] => 2008
    [2] => 2008
)
```

Notas ¶

**Sugerencia**

No use **preg_match()** si solo quiere comprobar si una cadena está contenida en otra cadena. Use strpos() en su lugar ya que será más rápida.

Ver también

Patrones de PCRE

preg_quote() - Escapar caracteres en una expresión regular

preg_match_all() - Realiza una comparación global de una expresión regular

preg_replace() - Realiza una búsqueda y sustitución de una expresión regular

preg_split() - Divide un string mediante una expresión regular

preg_last_error() - Devuelve el código de error de la última ejecución de expresión regular de PCRE

## Utilizar código antiguo en nuevas versiones de PHP

Ahora que PHP ha crecido y se ha convertido en un lenguaje popular, hay muchos más repositorios y bibliotecas que contienen código que puede reutilizar. Los desarrolladores de PHP han intentado preservar la retrocompatibilidad, es decir, si un script fue escrito para una versión antigua, funcionará

(idealmente) sin ningún cambio en una versión reciente de PHP. En la práctica, normalmente son necesarios algunos cambios.

Dos de los cambios más importantes que afectan el código antiguo son:

Los antiguos arrays `$HTTP_*_VARS` ya no están disponibles a partir de PHP 5.4.0. Los siguientes arrays superglobales fueron introducidos en PHP » 4.1.0. Son: `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_ENV`, `$_REQUEST`, y `$_SESSION`.

Las variables externas ya no son registradas en el ámbito global de forma predeterminada. En otras palabras, a partir de PHP » 4.2.0, la directiva de PHP register_globals está desactivada (*off*) por defecto en `php.ini`. El mejor método para acceder a estos valores es por medio de las variables superglobales mencionadas anteriormente. Los scripts, libros y tutoriales antiguos podrían contar con que esta directiva esté activada (*on*). Si fuera *on*, por ejemplo, se podría usar `$id` desde el URL *http://www.example.com/foo.php?id=42*. Ya esté activada o desactivada, `$_GET['id']` está siempre disponible.

# Algunos ejercicios para ir practicando

## Ejercicio 1

Concatena dos cadenas con el operador punto (.) e imprimir su resultado.

```
<!DOCTYPE html>
<html>
<head>
<title>Ejercicio 1</title>
</head>
<body>
<?php
	$ini = "Hola ";
	$fin = " a todos";
	$todo = $ini.$fin;
	echo $todo;
?>
</body>
</html>
```

## Ejercicio 2

Hacer un programa que sume dos variables que almacenan dos números distintos.

```
<html>
<head><title>Ejercicio 2</title></head>
<body>
<?php
	$n1=1;
	$n2=2;
	$suma=$n1+$n2;
	echo "suma = ".$suma. "<br>";
	echo "$n1+$n2";
?>
</body>
</html>
```

## Ejercicio 3

Hacer un programa que muestre en pantalla información de PHP con la función **phpinfo**(). Muestre la información centrada horizontalmente en la pantalla.

```
<html>
<head><title>Ejercicio 3</title></head>
<body>
<center>
<?php
	echo phpinfo();
?>
</center>
</body>
</html>
```

## Ejercicio 4

Mostrar en pantalla una tabla de 10 por 10 con los números del 1 al 100

```
<html>
<head><title>ejercicio 4</title> </head>
<body>
<?php
	echo "<table border=1>";
	$n=1;
	for ($n1=1; $n1<=10; $n1++)
	{
		echo "<tr>";
		for ($n2=1; $n2<=10; $n2++)
		{
			echo "<td>", $n, "</td>";
			$n=$n+1;
		}
		echo "</tr>";
	}
	echo "</table>";
?>
</body>
</html>
```

## Ejercicio 5

Idem al ejercicio anterior, pero colorear las filas alternando gris y blanco. Además, el tamaño será una constante: define(TAM, 10)

```
<html>
<head><title>Ejercicio 5</title></head>
<body>
<?php
	define(TAM,10);
	echo "<table border=1>";
	$n=1;
	for ($n1=1; $n1<=TAM; $n1++)  {
		if ($n1 % 2 == 0)
			echo "<tr bgcolor=#bdc3d6>";
		else
			echo "<tr>";
		for ($n2=1; $n2<=TAM; $n2++)		{
			echo "<td>", $n, "</td>";
			$n=$n+1;
		}
		echo "</tr>";
	}
	echo "</table>";
?>
</body>
</html>
```

## Ejercicio 6

Mostrar una tabla de 4 por 4 que muestre las primeras 4 potencias de los números del uno 1 al 4 (hacer una función que las calcule invocando la función **pow**). En PHP las funciones hay que definirlas antes de invocarlas. Los parámetros se indican con su nombre ($cantidad) si son por valor y antecedidos de & (&$cantidad) si son por referencia.

```
<html>
<head>
<title>Ejercicio 6</title>
</head>
<body>
<?php
	define(TAM,4);
	function potencia ($v1, $v2)
	{
		$rdo= pow($v1, $v2);
		return $rdo;
	}
	echo "<table border=1>";
	for ($n1=1; $n1<=TAM; $n1++)
	{
		echo "<tr>";
		for ($n2=1; $n2<=TAM; $n2++)
		echo "<td>". potencia($n1,$n2). "</td>";
		echo "</tr>";
	}
	echo "</table>";
?>
</body>
</html>
```

## Ejercicio 7

Hacer un programa que muestre en una tabla de 4 columnas todas las imágenes del directorio "fotos". Para ello consulte el manual (en concreto la referencia de funciones de directorios). Suponga que en el directorio sólo existen fotos.

```
<html>
<head>
<title>Ejercicio 7</title>
</head>
<body>
<?php
	if ($gestor = opendir('fotos'))
	{
		echo "<table border=1>";
		echo "<tr>";
		$i=0;
		while (false !== ($archivo = readdir($gestor)))
		{
			if ($archivo!="." && $archivo!="..")
			{
				if ($i==4)
				{
					$i=0;
					echo "</tr>";
					echo "<tr>";
				}
				$i++;
				echo "<td>";
				echo		"<a		href=fotos/$archivo><img
src=fotos/$archivo>
				</a>";
				echo "</td>";
			}
		}
		echo "</tr>";
		echo "</table>";
		closedir($gestor);
	}
?>
</body>
</html>
```

## Ejercicio 8

Idem al anterior, pero que muestre las fotos en 100x100 y que al pulsar abra la foto entera. Compruebe que sólo muestra fotos con extensión .jpg, .png, bmp o .gif (haga una función que lo compruebe usando las expresiones regulares como aparecen en el manual).

```
<!DOCTYPE html>
<html>
<head>
<title>Ejercicio 8</title>
</head>
<body>
<?php
	echo "<h1>Tabla de Fotos con Enlace</h1>";
	function valida_foto($fotos)
	{
		$rdo=0;
		if (ereg("[Jj][Pp][Gg]$", $fotos)) rdo=1;
		if (ereg("[Gg][Ii][Ff]$", $fotos)) rdo=1;
		if (ereg("[Pp][Nn][Gg]$", $fotos)) rdo=1;
		if (ereg("[Bb][Mm][Pp]$", $fotos)) rdo=1;
		return $rdo;
	}
	echo "<table border=1>";
	$puntero = opendir('fotos');
	$i=1;
	while (false !== ($foto = readdir($puntero)))
	{
		if ($foto!="." && $foto!=".." && valida_foto($foto))
		{
		if ($i==1)
			echo "<tr>";
		echo "<td><a href='fotos/$foto'>";
		echo "<img src='fotos/$foto' width=100 height=100></img>";
		echo "</a></td>";
		if ($i==4)
		{
			echo "</tr>"; $i=0;}
			$i++;
		}
	}
	closedir($puntero);
	echo "</table>";
?>
</body>
</html>
```

## Ejercicio 9

Idem al anterior, pero que por cada foto tenga una miniatura. Para la foto playa.jpg la miniatura será MINI-playa.jpg.

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

```
<html>
<head>
<meta   content="text/html;   charset=ISO-8859-1"   httpequiv="content-
type">
<title>Ejercicio 9</title>
</head>
<body>
<?php
      echo "<h1>Galeria de imagenes con thumbnails</h1>";
      function valida_foto($fotos)
      {
            $rdo=0;
            if (ereg("[Jj][Pp][Gg]$", $fotos)) rdo=1;
            if (ereg("[Gg][Ii][Ff]$", $fotos)) rdo=1;
            if (ereg("[Pp][Nn][Gg]$", $fotos)) rdo=1;
            if (ereg("[Bb][Mm][Pp]$", $fotos)) rdo=1;
            return $rdo;
      }
      echo "<table border=1>";
      $puntero = opendir('fotos');
      $i=1;
      while (false !== ($foto = readdir($puntero)))
      {
            if ($foto!="." && $foto!=".." && valida_foto($foto))
            {
                  if ($i==1)
                        echo "<tr>";
                  echo "<td><a href='fotos/tumbs/MINI-$foto'>";
                  echo      "<img      src='fotos/$foto'      width=100
height=100></img>";
                  echo "</a></td>";
                  if ($i==4)
                  {
                        echo "</tr>";
                        $i=0;
                  }
                  $i++;
            }
      }
      closedir($puntero);
      echo "</table>";
?>
</body>
</html>
```

## Ejercicio 10

Idem al anterior, pero que si no existe la miniatura de una foto debe de crearla. Para generar la miniatura se usa el programa *convert* (hay que invocarlo en línea de comandos desde PHP con la función *system*).

```
<!DOCTYPE>
<html>
<head><meta           content="text/html;           charset=ISO-8859-1"
httpequiv="content-type">
<title>Ejercicio 10</title>
</head>
<body>
<?php
      echo "<h1>Galeria de imagenes con thumbnails</h1>";
      function valida_foto($fotos)  {
            $rdo=0;
            if (ereg("[Jj][Pp][Gg]$", $fotos)) rdo=1;
            if (ereg("[Gg][Ii][Ff]$", $fotos)) rdo=1;
            if (ereg("[Pp][Nn][Gg]$", $fotos)) rdo=1;
            if (ereg("[Bb][Mm][Pp]$", $fotos)) rdo=1;
            return $rdo;
      }
      function crea_tumbs($foto)    {
            if (!is_dir('fotos/tumbs'))
            mkdir ('fotos/tumbs', 0777);
            if (!is_file('fotos/tumbs/MINI-$foto'))
            system     ("convert     -sample     40x40     /fotos/$foto
/fotos/tumbs/MINI-$foto");
      }
      echo "<table border=1>";
      $puntero = opendir('fotos');
      $i=1;
      while (false !== ($foto = readdir($puntero)))  {
       if ($foto!="." && $foto!=".." && valida_foto($foto))      {
                  crea_tumbs($foto);
                  if ($i==1)
                        echo "<tr>";
                  echo "<td><a href='fotos/tumbs/MINI-$foto'>";
                  echo      "<img      src='fotos/$foto'      width=100
height=100></img>";
                  echo "</a></td>";
                  if ($i==4)
                  {echo "</tr>"; $i=0;}
                  $i++;
            }
      }
      closedir($puntero);
      echo "</table>";
?>
</body>
</html>
```

## Ejercicio 11

PHP desde línea de comandos. Suponga que tenemos un servidor que no soporta PHP. Genere una página estática con la galería de fotos del ejercicio anterior.

Las razones para usar PHP generando contenidos estáticos pueden ser, además de la indicada anteriormente: para facilitar la indexación de contenidos (con spiders), para cargar menos el servidor, para realizar una página que funciona off-line (por ejemplo, una recopilación de información para grabarla en CD/DVD), etc.

Simplemente abría que invocar, desde la línea de comandos *php ejercicio10.php > pag.html*

## Ejercicio 12

Arrays. Almacene en un array los 10 primeros número pares. Imprímalos cada uno en una línea (recuerde que el salto de línea en HTML es <BR>).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" httpequiv=" content-type">
<title>Ejercicio 12</title>
</head>
<body>
<?php
      for ($i=0;$i<10;$i++)
            $v[$i]=$i*2;
      for ($i=0;$i<10;$i++)
            echo "$v[$i]<br>";
?>
</body>
</html>
```

## Ejercicio 13

Imprima los valores del vector asociativo siguiente usando la estructura de control foreach:

- $v[1]=90;
- $v[30]=7;
- $v['e']=99;
- $v['hola']=43;

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content="text/html; charset=ISO-8859-1" httpequiv="
content-type">
<title>Ejercicio 13</title>
</head>
<body>
<?php
	$v[1]=90;
	$v[30]=7;
	$v['e']=99;
	$v['hola']=43;
	foreach ($v as $indice => $valor)
	{
		echo "El elemento de indice $indice vale $valor <br>";
	}
?>
</body>
</html>
```