



Primeros pasos con PHP

Nivelación



Primeros pasos con PHP

El objetivo de este fastbook es que comiences a familiarizarte con la sintaxis básica de PHP. A lo largo de estas páginas, detallaremos las operaciones más básicas que se pueden realizar con este lenguaje de programación. En último lugar se indican unas pautas a seguir para preparar un entorno de desarrollo de PHP en cualquier sistema operativo, de cara a poder probar y afianzar los conceptos desarrollados a lo largo del tema.

Autor: Jesús Donoso

☰ Introducción

☰ Conceptos básicos para generación de código PHP

☰ Operadores

☰ Arrays

☰ Estructuras de control

☰ Ficheros

☰ Formularios

☰ Anexos

Introducción



PHP (Hypertext Preprocessor) es un lenguaje de código abierto ampliamente utilizado en el mundo del desarrollo web para crear páginas dinámicas y aplicaciones interactivas.

Gracias a su facilidad de aprendizaje y de uso, es uno de los lenguajes de programación más utilizados actualmente. Los sitios web de marcas mundialmente conocidas en la actualidad están desarrolladas en PHP. Estas son algunas de las principales

características de PHP:

- Integración con HTML.** PHP se integra fácilmente con HTML. Esto permite a los desarrolladores la creación de páginas web dinámicas en la que el contenido se genera en función de diferentes variables, información contenida en bases de datos o en formularios de usuario.
- Lenguaje ejecutado en el servidor.** El código PHP se ejecuta a nivel de servidor y genera el contenido HTML dinámico que se envía al cliente, en este caso, un navegador web. El usuario visualiza en el navegador la interpretación del código HTML recibido, pero no conoce que en el servidor fue ejecutado un script PHP.
- Amplia disponibilidad y código abierto.** PHP es compatible con la mayoría de los sistemas operativos y servidores web más populares actualmente. Algunos ejemplos de servidores web son Apache, NGinx e IIS. La configuración de estos servidores puede descargarse de forma gratuita y modificarse según las necesidades del momento.

- Fácil aprendizaje y uso.** El lenguaje PHP tiene una sintaxis sencilla y está diseñado para que sea bastante accesible. Cuenta con un alto nivel de documentación, una gran comunidad de desarrollo activa y una gran cantidad de recursos en línea.
- Acceso a bases de datos.** Proporciona un amplio soporte para poder trabajar con diferentes sistemas de gestión de bases de datos como, por ejemplo, MySQL, PostgreSQL o MongoDB.
- Librerías y frameworks.** Cuenta con un amplio abanico de librerías y frameworks que ayudan a simplificar el desarrollo web. Algunos ejemplos de estos frameworks son Laravel, Symfony y Codeigniter. Estos frameworks proporcionan estructuras y herramientas predefinidas para agilizar la creación de código PHP.

Conceptos básicos para generación de código PHP



Antes de aprender a lanzar código en este lenguaje, es importante que obtengas una buena base conceptual y entiendas cómo funciona. ¡Vamos allá!

Sintaxis básica

El código PHP se desarrolla en scripts con extensión **.php**. Para la generación de código PHP se utiliza la siguiente nomenclatura de etiquetas de apertura y cierre: **<?php**, antes de la primera instrucción, y **?>** tras la última instrucción. Cada bloque de instrucciones se denomina *script*. Todas las instrucciones de un script **deben terminar en punto y coma** para separarlas. A continuación, se puede observar un ejemplo de código PHP embebido dentro de código HTML.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Bienvenido a PHP</title>
    </head>
    <body>
        <?php
            echo "Bienvenido a introducción en PHP";
        ?>

        </body>
    </html>
```

El resultado de la ejecución del script anterior se puede observar en la imagen a continuación. En el navegador web se puede observar en el título de la pestaña la frase 'Bienvenido a PHP' y en la pantalla el texto 'Bienvenido a introducción en PHP'.



En los siguientes apartados, detallaremos las operaciones básicas para programar en PHP como son la creación de comentarios de texto, las impresiones por pantalla y la declaración de constantes, variables y funciones.

Comentarios de código

Al igual que en otros lenguajes de programación, una línea de comentario es una línea de texto o de código que existe dentro del script pero que no se ejecutará. Esta sería la sintaxis para comentar una sola línea o varias líneas de código en PHP.

- Comentar una sola línea: **//Texto comentado**
- Comentar varias líneas: **/*Texto comentado*/**

En cualquier tipo de desarrollo de programación, los comentarios son útiles durante el desarrollo y, *a posteriori*, para la explicación de las distintas funcionalidades del código. En el fragmento de código que muestro más abajo podemos observar algunos ejemplos de comentario de una línea y de varias líneas en PHP.

```
<!DOCTYPE html>
<?php
    // Comentario de una línea
    echo "Hello from the docket container";

    /*
    Comentario
    de varias líneas.
    */
    print("Texto");
?>
</html>
```

Salidas por pantalla

Existen dos instrucciones en PHP que al ejecutarse proporcionan la funcionalidad de mostrar texto en el navegador web. Estas funciones son ***echo*** y ***print***.

Para usar la instrucción ***echo*** basta con indicar la cadena de texto que se quiere mostrar a continuación de la instrucción. Por el contrario, para hacer uso de la instrucción ***print*** es necesario encerrar entre paréntesis la cadena de texto a mostrar.

En los siguientes fragmentos de código vemos los ejemplos de las instrucciones ***echo*** y ***print*** para mostrar por pantalla el mismo texto.

```
<!DOCTYPE html>
<head>
    <title>Bienvenido a PHP</title>
</head>
<body>
    <?php
        echo "Hello from the docket container <br>";
        print ("Hello from the docket container");
    ?>
</body>
</html>
```

Declaración de constantes

Al igual que en otros lenguajes de programación, una constante es un valor que no va a ser modificado a lo largo del proceso de ejecución de los scripts que contiene un documento. Estos valores se mantienen incluso cuando son invocados dentro de una función.

En PHP las constantes se definen mediante la siguiente instrucción: **define ("Nombre", "Valor")**. ¿Qué datos debemos tener en cuenta con respecto a estas constantes? ¡Toma nota!

- No es necesario que escribas entre comillas los valores de las constantes cuando se trata de constantes numéricas.
- Si se realizan operaciones aritméticas con constantes tipo cadena y su valor comienza por una letra, PHP les asigna valor cero.

Aquí dispones de un ejemplo en código PHP de la definición de una constante:

```
<?php  
    define("Pi", 3.14);  
    define("Saludo", "Hola")  
?>
```

Declaración de variables

Las variables son el espacio de memoria RAM que el ordenador reserva para almacenar un determinado tipo de datos cuyos valores son susceptibles de ser modificados. ¿A qué aspectos debemos prestar atención en cuanto a las variables?

- 1 Que se definen con el símbolo **\$**.
- 2 Inmediatamente después del símbolo clave para la declaración de las variables, estas han de llevar una letra y no un número. **\$nombre_variable** es un nombre de variable correcto, mientras que **\$1nombre_variable** no es un nombre aceptado.
- 3 PHP es *case sensitive*, es decir, hay diferencia entre caracteres en mayúscula y minúscula.
- 4 Si se declaran dos variables con el mismo nombre realmente solo se declara una y contiene el último valor declarado.
- 5 No se requiere una definición previa a su uso. Se definen en el momento que son necesarias y para ello basta asignarles un valor.
- 6 Si se usa una variable de forma previa a su definición PHP, se considera que la variable tiene valor cero, en caso de ser número; o es una cadena de caracteres vacía en caso de ser texto.

7

Tras la definición de una variable, los valores de esta pueden ser utilizados desde cualquier otra parte del mismo script, exceptuando su uso dentro de las funciones declaradas en el script, donde las variables tendrían que ser pasadas como parámetros.

8

Una variable definida dentro de una función puede ser utilizada únicamente dentro de la misma función.

9

Si definimos dos variables con el mismo nombre, una dentro de una función y otra fuera, PHP las considerará distintas. La variable declarada dentro de la función utilizará sus propios valores sin que sus resultados modifique la variable externa.

En el fragmento de código a continuación se puede observar la declaración de diferentes variables:

```
<?php  
$nombre = 'Pilar';  
$edad = 33;  
$mayorDeEdad = false;  
?>
```

Declaración de funciones

Una función en PHP, al igual que en otros lenguajes de programación, es una sucesión de acciones que se realizan de manera independiente respecto al resto del script. A continuación, se puede ver un ejemplo de la sintaxis necesaria para la definición de una función en PHP:

```
<?php
    function suma($arg, $arg2) {
        $suma = $arg + $arg2;
        return $suma;
    }
    $valor1 = 10;
    $valor2 = 20;
    $resultado = return_sum($valor1, $valor2)
?>
```

En el ejemplo se puede observar cómo la función **suma** recibe dos variables como argumento. Dentro de la función se crea la variable **\$suma** en la que se almacena el resultado de la suma de ambas variables. La función devuelve el valor almacenado en la variable **suma**. Como resultado del script, en la variable **\$resultado** se almacena el valor 30.

Operadores



Qualentum Lab

En este apartado se indica la sintaxis en PHP de diferentes operadores aritméticos y de comparación necesarios para una programación básica. Estos operadores son fundamentales en PHP y se utilizan ampliamente en la programación para realizar cálculos y tomar decisiones basadas en comparaciones de valores.



En la documentación oficial de [PHP](#) se pueden encontrar todos los tipos de operadores y su sintaxis.

Te recomiendo encarecidamente que la estudies para completar tu formación.

A continuación, listaré algunos de los operadores más básicos.

Operadores aritméticos

suma (+): se utiliza para sumar dos valores.

```
<?php $resultado = $a + $b; ?>
```

Resta (-): se utiliza para restar el valor de la derecha del valor de la izquierda.

```
<?php $resultado = $a - $b; ?>
```

Multiplicación (*): se utiliza para multiplicar dos valores.

```
<?php $resultado = $a * $b; ?>
```

División (/): se utiliza para dividir el valor de la izquierda por el valor de la derecha.

```
<?php $resultado = $a / $b; ?>
```

Módulo (%): devuelve el resto de la división del valor de la izquierda por el valor de la derecha.

```
<?php $resultado = $a % $b; ?>
```

Incremento (++): se utilizan para aumentar el valor de una variable en 1.

```
<?php $a++; ?>
```

Decremento (--): se utilizan para disminuir el valor de una variable en 1.

```
<?php $a--; ?>
```

Operadores de comparación

Igual (==): compara si dos valores son iguales en contenido, sin tener en cuenta el tipo de datos.

```
<?php if ($a == $b) {} ?>
```

No igual (!=): compara si dos valores no son iguales en contenido, sin tener en cuenta el tipo de datos

```
<?php if ($a != $b) {} ?>
```

Igual estricto (==): compara si dos valores son iguales en contenido y tipo de datos.

```
<?php if ($a === $b) {} ?>
```

No igual estricto (!==): compara si dos valores no son iguales en contenido o tipo de datos.

```
<?php if ($a !== $b) {} ?>
```

Mayor que (>): compara si el valor de la izquierda es mayor que el valor de la derecha.

```
<?php if ($a > $b) {} ?>
```

Menor que (<): compara si el valor de la izquierda es menor que el valor de la derecha.

```
<?php if ($a < $b) {} ?>
```

Mayor o igual que (>=): compara si el valor de la izquierda es mayor o igual que el valor de la derecha.

```
<?php if ($a >= $b) {} ?>
```

Menor o igual que (<=): compara si el valor de la izquierda es menor o igual que el valor de la derecha.

```
<?php if ($a <= $b) {} ?>
```

Arrays



Un array es una estructura de datos que permite **almacenar múltiples valores en una sola variable** donde normalmente cada elemento está asociado a una clave. Estos valores pueden ser de diferentes tipos de datos, como números, cadenas de texto, objetos u otros arrays. Al igual que en otros lenguajes de programación, los arrays son una estructura fundamental en PHP y se utilizan con frecuencia para almacenar y manipular información.

En PHP existen **varios tipos de arrays**. A continuación, se detallan dos de ellos: los arrays indexados y los arrays asociativos. Los arrays indexados asocian los valores a claves numéricas según su posición dentro de la cadena. En los arrays asociativos la clave asignada al valor es customizable.

Vamos a estudiar en detalle ambos tipos de arrays con algunos ejemplos en código. ¡Presta atención!



Arrays indexados

Los valores almacenados dentro del array son asignados a una clave o **índice numerado**. La numeración de los elementos siempre comienza en cero. A continuación, podemos observar el código necesario para la declaración de un array indexado. Fíjate bien, en este caso el array está formado por tres cadenas de texto. Las sentencias echo devuelven cada uno de los valores de la cadena.

```
<?php
$colores = array("rojo", "verde", "azul");
echo $colores[0]; // Se imprime por pantalla la cadena "rojo"
echo $colores[1]; // Se imprime por pantalla la cadena "verde"
echo $colores[2]; // Se imprime por pantalla la cadena "azul"
echo $colores[3]; // Se interrumpe la ejecución del script por acceso a
                  una posición del array incorrecta
?>
```



Arrays asociativos

En este caso, los valores almacenados dentro del array son relacionados en el momento de su declaración con una clave o **nombre** que lo identifique. Los arrays asociativos se utilizan comúnmente para representar datos en formas de pares clave–valor. A continuación, se puede observar un ejemplo de la declaración de un array indexado. En este caso los datos almacenados se corresponden con atributos de una persona. También se muestra como acceder a los datos contenidos en el array.

```
<?php
$persona = array (
    "nombre" => "Alex",
    "edad" => 35,
    "ciudad" => "Madrid"
);
echo $persona["nombre"]; // Se imprime por pantalla la cadena "Alex"
echo $persona["edad"]; // Se imprime por pantalla el entero 35
echo $persona["ciudad"]; // Se imprime por pantalla la cadena "Madrid"
?>
```

Existen funciones para la modificación de los arrays en PHP. De hecho, en el fragmento de código que muestro a continuación vemos el uso de las funciones `array_push` y `unset` para añadir y eliminar elementos del array respectivamente.

```
<?php
    $persona = array("Rojo", "Verde");
    //Se añade "Amarillo" al array
    $persona = array_push($persona, "Amarillo");
    //El array persona tiene los valores ["Rojo", "Verde", "Amarillo"]
    //Se elimina el primer elemento del array
    unset($persona[0]);
    //El array persona tiene los valores ["Verde", "Amarillo"]
?>
```

Además de los arrays indexados y asociativos, PHP también admite otros tipos de arrays como los **arrays multidimensionales** (arrays que contienen otros arrays), **arrays numéricos** (arrays indexados sin claves asociadas) y más.



Para consultar otros tipos de arrays y más funciones para su gestión consulta la documentación oficial de [PHP](#).

Estructuras de control



Qualentum Lab

Damos un paso más en la programación con PHP para que conozcas dos elementos clave de este lenguaje: las **sentencias** y los **bucles**; y sepas cómo y para qué utilizarlos.

Sentencias condicionales

1

Sentencia if

La estructura **if** se utiliza para ejecutar una sección de nuestro código si se cumple la condición que estamos representando en su llamada. Un ejemplo de uso del condicionante **if** sería este que comparto:

```
<?php
$edad = 30;
if ($edad >= 18) {
    echo "Mayor de edad";
}
?>
```

En este caso, como se ve en el código, se imprime por pantalla la cadena “*Mayor de edad*” ya que se cumple la condición especificada en el *if*.

2

Sentencia **else**

La estructura **else** se utiliza en conjunto con el condicionante *if*. Si no se cumple la condición especificada en la sentencia *if*, se ejecuta el código contenido en el apartado *else*. Un ejemplo de uso del condicionante *else* sería el siguiente:

```
<?php
$edad = 10;
if ($edad >= 18) {
    echo "Mayor de edad";
} else {
    echo "Menor de edad";
}
?>
```

En este caso, como se ve en el código, se imprime por pantalla la cadena “*Menor de edad*” ya que no se cumple la condición especificada en el *if*.

3

Sentencia **elseif**

La estructura **elseif** se utiliza en conjunto con la sentencia *if* cuando existen múltiples condiciones que se quieren verificar en un determinado orden. En el momento que una condición se cumple el resto no se evalúan. Un ejemplo de uso del condicionante *elseif* sería el siguiente:

```
<?php  
$edad = 35;  
if ($edad > 50) {  
    echo "Mayor de 50";  
} elseif ($edad >= 30) {  
    echo "Entre 30 y 50 años";  
} elseif ($edad >= 18) {  
    echo "Entre de 18 y 29";  
} else {  
    echo "Menor de edad";  
}  
?  
>
```

En este caso la variable edad tiene como valor 35. En primer lugar, se comprueba si la variable edad es mayor a 50. Al no cumplirse esta condición, se ejecuta la primera de las sentencias con el condicionante `elseif`. En este caso, se evalúa si la variable edad es mayor o igual a 30. Ya que la condición especificada sí se cumple, se imprime por pantalla la cadena *"Entre 30 y 50 años"*.

Bucles

Al igual que en otros lenguajes de programación los bucles son estructuras de control que tienen como resultado la ejecución de un mismo código de manera iterativa, siempre y cuando se cumpla una condición.

Al igual que en el apartado anterior, vamos a estudiar los **tipos de bucles**, al menos, los más comunes.

1

Bucle **for**

Un bucle **for** es una estructura muy útil cuando se conoce de antemano cuántas ejecuciones del mismo fragmento de código se necesitan. La sintaxis general de un bucle *for* en PHP es la siguiente:

```
<?php
    for (inicialización; condición; incremento/decremento) {
        // Bloque de código a ejecutar en cada iteración
    }
?>
```

La sentencia *for*, por tanto, recibe tres parámetros:

A continuación, te comarto un ejemplo de bucle *for* en PHP. En este bucle, la variable de control *\$i* comienza con valor 1. El bucle se ejecutará mientras que *\$i* sea menor o igual a 5. En cada iteración *\$i* se incrementa en 1. Por lo tanto, el resultado de la ejecución del bucle es la impresión por pantalla de los valores "1 2 3 4 5".

INICIALIZACIÓN

CONDICIÓN

INCREMENTO/DECREMENTO

Se establece la variable de control y se le asigna un valor inicial. Esto se ejecuta solo una vez al principio del bucle.

INICIALIZACIÓN**CONDICIÓN****INCREMENTO/DECREMENTO**

Verifica la condición en cada iteración del bucle. Si la concisión es verdadera él bucle continúa ejecutándose. Si es falsa el bucle terminaría y se continuaría con la ejecución del script.

INICIALIZACIÓN**CONDICIÓN****INCREMENTO/DECREMENTO**

Se aumenta o decrementa el valor de iniciación en cada iteración.

```
<?php  
    for ($i = 1; $i <= 5 ; $i++) {  
        // Bloque de código a ejecutar en cada iteración  
        echo $i . " ";  
    }  
?>
```

Un bucle *for* se puede utilizar para recorrer arrays, realizar tareas repetitivas y generar secuencias numéricas que sean predecibles. Es importante tener cuidado en la definición de la condición que se verifica en cada iteración del bucle y en el aumento o decremento de este, ya que una mala definición podría generar bucles infinitos.

2

Bucle *while*

Un bucle ***while*** es una estructura de control que permite ejecutar un bloque de código de manera repetida mientras una condición se sigue cumpliendo. A diferencia del bucle *for*, el bucle *while* no se basa en un número específico de iteraciones, sino que evalúa una condición en cada ejecución del bloque de código. Esta es la sintaxis general de un bucle *while* en PHP:

```
<?php
    while (condición) {
        // Bloque de código a ejecutar en cada iteración
    }
?>
```

La condición que se indica en la definición del bucle se comprueba por primera vez. Si la condición se cumple se ejecuta el código contenido en el bucle hasta que la condición deje de ser verdadera.

Veamos un ejemplo de bucle *while* en PHP.

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

Si te fijas, se define una variable `$i` con valor 1. Como se cumple la condición especificada en el `while` se inicia la ejecución del bucle. En cada iteración `$i` se incrementa en 1. Por lo tanto, el resultado de la ejecución del bucle es la impresión por pantalla de los valores “1 2 3 4 5”.

Recuerda: un bucle `while` es útil cuando no se conoce de antemano cuántas veces se ejecutará el bloque de código, pero se necesita que se ejecute mientras una condición sea verdadera. Es importante, por tanto, tener en cuenta que si la condición nunca se vuelve falsa se estaría ejecutando un bucle infinito.

3

Bucle `do-while`

Un bucle `do-while` es una estructura que ejecuta el bloque de código una vez y luego se puede repetir si cumple la condición. En el apartado anterior, hemos visto que el bucle `while` comprueba la condición al iniciarse la ejecución. Sin embargo, un bucle `do-while` ejecuta la primera iteración y después comprueba la condición. La sintaxis general de un bucle `do-while` en PHP sería la siguiente:

```
<?php
do {
    // Bloque de código a ejecutar en cada iteración
} while (condición)
?>
```

Ahora vamos a analizar un ejemplo de bucle *do-while* en PHP: se define una variable *\$i* con valor 1. Se realiza la primera ejecución del código dentro de la parte *do*. Como se cumple la condición especificada en el *while* se continúa con la ejecución del código de la parte *do*. En cada iteración *\$i* se incrementa en 1. Por lo tanto, el resultado de la ejecución del bucle es la impresión por pantalla de los valores “1 2 3 4 5”.

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

Este bucle es útil cuando se desea que el bloque de código se ejecute mínimo una vez, independientemente de la condición, y luego que continúe ejecutándose mientras se cumpla la condición. Al igual que en el bucle *while*, no hay que olvidar que si la condición nunca se vuelve falsa, se estaría ejecutando un bucle infinito.

4

Bucle *foreach*

El bucle ***foreach*** es una estructura de control especialmente diseñada para recorrer elementos de un array o elementos de un objeto iterable.

Esta es la sintaxis general de un bucle *foreach* en PHP:

```
<?php
    foreach ($array as $valor) {
        // Código a ejecutar en cada elemento iterado
    }
?>
```

- **\$array:** es el array que se desea recorrer.
- **\$valor:** es una variable que representa el valor de cada elemento del array en cada iteración del bucle.

El bucle *foreach* recorre cada elemento del array, uno por uno, y ejecuta el bloque de código especificado para cada elemento. El valor de cada elemento se asigna de manera secuencial en cada ejecución.

Vamos a ver ahora el ejemplo de bucle *foreach* en PHP.

```
<?php  
    $colores = array("rojo", "verde", "azul");  
    foreach ($colores as $color) {  
        echo $color . " ";  
    }  
?>
```

El resultado de la ejecución del bucle es la salida por pantalla de las cadenas de caracteres 'rojo verde azul'.

Un bucle `foreach` se puede utilizar también para recorrer arrays asociativos, lo que permite acceder a la clave y al valor asignado a dicho elemento del array. En ese caso, hay que especificar las claves y el valor, aunque la clave no siempre es obligatoria.

Veámoslo con este ejemplo:

```
<?php  
    $persona = array(  
        "nombre" => "Alex",  
        "edad" => 35,  
        "ciudad" => "Madrid"  
    );  
    foreach ($persona as $clave => $valor) {  
        echo "La clave es: ". $clave .", y el valor es: ". $valor . "<br/>"  
    }  
?>
```

El resultado de la ejecución del bucle es la salida por pantalla de las cadenas de caracteres:
*"La clave es: nombre, y el valor es: Alex La clave es: edad, y el valor es: 35 La clave es: ciudad,
y el valor es: Madrid".*

Ficheros



Qualentum Lab

En este apartado aprenderemos las funciones de gestión de **ficheros a nivel de servidor**.

En PHP se pueden realizar diferentes operaciones de lectura, escritura, modificación y eliminación de ficheros en el sistema de archivos. Por tanto, veremos en detalle las operaciones básicas sobre ficheros, como la apertura, lectura y escritura de estos. No obstante, te comarto la documentación oficial de [PHP](#), aquí puedes (y debes) consultar todas las operaciones que se pueden realizar sobre ficheros.

1

Apertura de ficheros

Para la apertura de un fichero en PHP, podemos hacer uso de la función **fopen()**. Los ficheros pueden ser abiertos con diferentes permisos de lectura y/o escritura. Igualmente, al abrir un fichero se genera un puntero, que como en otros lenguajes de programación, es una variable que apunta a una posición concreta de memoria. Al abrir el fichero se puede apuntar al inicio o final de este.

Para que lo entiendas mejor, este es un ejemplo del uso de la función **fopen()** para la apertura de un fichero:

```
<?php  
    fopen('fichero.txt', 'r');  
?>
```

Como primer argumento es necesario especificar la ruta y nombre del fichero. En el ejemplo, el fichero está ubicado en la misma ruta en la que se ejecuta el script y, por ello, únicamente es necesario el nombre del fichero. El segundo argumento es el correspondiente a los permisos con el fichero.



Si quieres conocer en detalle los tipos de permisos, insisto: revisa la documentación oficial de [PHP](#).

2

Lectura de ficheros

La lectura es una operación fundamental en la gestión de ficheros. Dos funciones de las que se puede hacer uso para leer contenido de un fichero son *fgets()* y *file_get_contents()*. Por un lado, la función *fgets()* permite leer el contenido de un archivo línea a línea. Por otro lado, la función *file_get_contents()* permite la lectura del archivo completo.

Como viene siendo habitual, vamos a verlo con un ejemplo de lectura de un fichero usando la función *fgets()* y, después, te comarto explicación de los pasos llevados a cabo.

```
<?php  
    fopen('fichero.txt', 'r');  
  
    while (!feof($archivo)) {  
        $linea = fgets($archivo);  
        echo $linea;  
    }  
    fclose($archivo);  
?>
```

- Abrir el *fichero.txt* con la función *fopen()*.
- Leer el fichero, línea a línea, hasta que se llegue al final. La función *feof()* devuelve el valor *true* cuando se termina de recorrer el fichero.
- Una vez terminada la lectura del fichero, se ejecuta el cierre del mismo con la función *fclose()*. Es importante cerrar los ficheros tras su uso ya que esta acción libera recursos y garantiza que, de haberse producido modificaciones, se guarden correctamente.

3

Escritura de ficheros

Otra de las operaciones básicas que se puede realizar con ficheros es la escritura. Al igual que en la lectura de ficheros, en PHP existe la posibilidad de escribir línea a línea o insertar un bloque de texto de una sola vez. Estas funciones son, respectivamente, *fwrite()* y *file_put_contents()*.

A continuación, te muestro el ejemplo de escritura en un fichero usando la función *fwrite()* para después explicarte los pasos realizados:

```
<?php  
$nombreArchivo = "miarchivo.txt";  
$contenido = "Contenido que habrá dentro del fichero";  
// Abre el fichero en modo escritura (creará el archivo si no existe)  
$archivo = fopen($nombreArchivo, "w") or die("No se puede abrir el fichero");  
// Escribe el contenido del fichero  
fwrite($archivo, $contenido);  
// Cierra el fichero  
fclose($archivo);  
?>
```

- Primeramente se abre el fichero en modo escritura como con la función `fopen()` y el parámetro '`w`'. Se puede observar también el uso de la función `die()` para el control de errores en la apertura de un fichero.
- Se utiliza la función `fwrite()` para la escritura en el fichero. La función recibe como argumentos el fichero en el que escribir y el contenido a incluir en el fichero.
- Se cierra el fichero con la función `fclose()`.

A continuación, se puede observar un ejemplo de escritura en un fichero usando la función `file_put_contents()`.

```
<?php  
$nombreArchivo = "miarchivo.txt";  
$contenido = "Contenido que habrá dentro del fichero";  
// Escribe el contenido en el archivo (creará el fichero si no existe)  
file_put_contents($nombreArchivo, $contenido);  
?>
```

Como se puede observar, para la utilización de esta función no es necesario realizar la apertura y cierre del fichero. Únicamente basta con realizar la llamada a la función e incluir, como argumentos, el archivo en el que se desea escribir y el contenido a incluir.

Formularios



Qualentum Lab

Los formularios en PHP son fundamentales en las aplicaciones web. Permiten a los usuarios **interactuar** con un sitio web para recoger los datos que se necesiten como inputs y enviarlos al servidor para su procesamiento. En este epígrafe te voy a mostrar cómo se trabaja con los formularios.

El primer paso que debemos llevar a cabo para la definición de un formulario es la creación de un archivo HTML que contenga los inputs que se necesitan.

Si te fijas en el siguiente ejemplo, vemos un script HTML en el que se declaran, como inputs de un formulario, el nombre y un mensaje que un usuario pudiera introducir. Igualmente, podemos ver el uso de etiquetas para describir los campos y atributos. Estas etiquetas son de ayuda a la hora de identificar los campos cuando se envían los datos al servidor.

```
<!DOCTYPE html>
<head>
    <title>Formulario de contacto</title>
</head>
<body>
    <h1>Contacto</h1>
    <form method="post" action="index.php">
        <label for="nombre"> Nombre:</label>
        <input type="text" id="nombre" name="nombre" required/><br/><br/>

        <label for="mensaje"> Mensaje:</label>
        <textarea id="mensaje" name="mensaje" rows="4" required></textarea>

        <input type="submit" value="Enviar" />
    </form>
</body>
</html>
```

Adicionalmente, se crea un script que procesa los datos recogidos en el formulario y que imprime por pantalla dicha información. En el fragmento de código a continuación se puede observar esta función. Al inicio se verifica que el método de la petición es *POST* y, posteriormente, se obtienen los datos del formulario haciendo de la variable *\$_POST* y el nombre de la clave.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nombre = $_POST['nombre'];
    $mensaje = $_POST['mensaje'];
    echo "Gracias $nombre, por tu mensaje $mensaje";
}
?>
```

Anexos



Qualentum Lab

Por último, antes de finalizar este fastbook te comarto una serie de anexos que te ayudarán a dar tus primeros pasos en PHP, así como enlaces con información de interés.

Anexo I. Preparación de entorno PHP para cualquier sistema operativo

Para la programación en PHP es necesaria la configuración de un servidor web. A continuación, te detallo las pautas necesarias que pueden seguirse para configurar un servidor web en cualquier sistema operativo de forma sencilla. Para ello se hace uso de **Docker**.

1

¿Qué es Docker?

Docker es una plataforma de código abierto que nos facilita la creación, implementación y administración de aplicaciones en contenedores. Los contenedores son entornos de ejecución ligeros y portables que encapsulan a una aplicación. Docker facilita la distribución y ejecución de las diferentes aplicaciones independientemente del sistema operativo que tengas. Veamos a continuación sus principales **características**:

- Contenedores:** utiliza contenedores para empaquetar aplicaciones y sus dependencias en un único paquete que se puede ejecutar de manera consistente en cualquier entorno que admita Docker.
- Portabilidad:** los contenedores son portables, lo que significa que funcionan de manera consistente en diferentes SO.
- Distribución:** facilita la distribución a través de imágenes de contenedores. Las imágenes son instantáneas estáticas de una aplicación y su entorno.
- Automatización:** Docker proporciona herramientas de automatización y orquestación, como **Docker Compose** y **Kubernetes**, que permiten administrar y escalar dichas imágenes.
- Gestión de recursos:** Docker proporciona herramientas para gestionar recursos como GPU y memoria asignada a dicho contenedor.
- Seguridad:** incluye características de seguridad, como la separación de recursos y la capacidad de aislar contenedores, lo que ayuda a mantener las aplicaciones seguras y protegidas entre sí.

2

Instalación del entorno

Tienes **dos opciones** para realizar dicha instalación: Puedes descargar directamente desde su [página web](#) o **instalar Docker-compose**: esta herramienta se utiliza para la definición y ejecución de proyectos en diferentes plataformas. Permite simplificar el uso de Docker a partir de archivos YAML. De esta forma es más sencillo generar contenedores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc. Puede descargarla aquí, desde [Docker Desktop](#).

Como se ha indicado en el apartado anterior para utilizar Docker es necesario tener instalado **docker-compose**.

En la carpeta en la que se ubique el proyecto es necesario tener el fichero *Dockerfile* y un *docker-compose.yml*. Ambos ficheros contienen la configuración básica para el despliegue del servidor. Los pasos para poder levantar el servidor de Docker en los proyectos son los siguientes:

- Ubicarse en la carpeta del proyecto.
- Abrir un terminal y ejecutar el comando **docker-compose build**.
- Ejecutar el comando **docker-compose up**. Este comando levanta el contenedor de Docker y por lo tanto el servidor web.

Y una vez realizados estos pasos el servidor web estaría desplegado y ya podrías ejecutar el código PHP.

Anexo II. Enlaces de interés

Cuando trabajes con PHP te recomiendo que siempre tengas a mano...

- El manual oficial de PHP: <https://www.php.net/manual/es/intro-whatis.php>
- El acceso a la plataforma Docker: <https://www.docker.com/>

¡Enhорabuena! Fastbook superado



Qualentum.com