

Guia PHP

PRINCIPIANTES

**JORGE MOYA
DELGADO**



Indice

1.	Introducción a PHP	2
1.1	¿Qué es php?	3
1.2	¿Para qué se usa PHP?	3
1.3	Ventajas en el uso de PHP	4
2.	Configuración del entorno de desarrollo	5
2.1	Instalación de XAMPP	5
2.2	Instalación de Visual Studio Code	6
2.3	Configuración de XAMPP	7
3.	Variables y tipos de datos	13
3.1	Variables en PHP	13
3.2	Tipos de datos en PHP	13
3.3	Conversión de tipos de datos	16
4.	Operadores	17
4.1	Operadores aritméticos.....	17
4.2	Operadores de comparación	18
4.3	Operadores lógicos.....	19
5.	Estructuras de control de flujo.....	20
5.1	Estructuras condicionales (if, else; elseif...)	20
5.2	Estructuras de repetición (While, do-while, foreach).....	22
5.3	Interrupción de estructuras de repetición (break, continue)	24
6.	Funciones.....	25

6.1	¿Qué son las funciones?	25
6.2	Definición y llamado de funciones	25
6.3	Parametros y argumentos de las funciones.....	26
7.	Arrays.....	27
7.1	¿Qué son los Arrays?	27
7.2	Creación y manipulación de arrays.....	27
7.3	Arrays asociativos.....	28
8.	Trabajo con formularios	34
8.1	Métodos GET y POST	34
8.2	Acceso a variables de formularios	36
8.3	Validación de datos de formularios.....	39
9.	Trabajo con archivos.....	43
9.1	Apertura y cierre de archivos	43
9.2	Lectura y escritura de archivos	44
9.3	Manipulación de archivos	46
10.	Introducción a la programación orientada a objetos	46
10.1	¿Qué es la POO?	46
10.2	Creación de clases y objetos.....	46
10.3	Métodos y propiedades de clases y objetos.....	46

1. Introducción a la guía de Jorge Moya

Hola, estoy redactando esta introducción habiendo finalizado casi por completo (siempre todo a medias hasta el último momento) éste manual.

He sentido la imperiosa necesidad de hacer saber a cualquier lector, aprendiz o maestro, en el noble arte de la programación, que éste manual no es parecido a nada que haya podido ver o leer antes.

Aquí se mezcla teoría (redactada por mí pero con un toque más serio), práctica (ejercicios basados en pensamientos que a veces no se aproximan a los típicos, pero que sirven para fijar y aprender a usar los conocimientos. Al final, el verdadero objetivo) y comentarios (realizados para amenizar aún más la experiencia que supone obtener los conocimientos necesarios para ese examen, puesto de trabajo o resolver ese gusanillo de curiosidad con la programación).

Al final de cada manual se incluirán (por eso aclaré que el manual estaba a medias, no aceptamos reclamaciones...) ejercicios para fijar los conocimientos aún más.

Esto simplemente es mi manera de acercar a la gente todos los conocimientos obtenidos por mí durante estos años, de una manera en la que a mí me hubiera gustado encontrarme las cosas y dejando a tú disposición aquellas herramientas u otras guías en las que alguna vez, yo también busqué refugio.

Agradecer a mi familia por el apoyo y a Christian, por afianzar en mi cabeza la idea de que unos simples PDFs que le enviaba, de propia cosecha, le ayudaban a estudiar, podían servir para aprender.

2. Introducción a PHP

2.1 ¿Qué es php?

PHP es un lenguaje de programación de código abierto utilizado principalmente para desarrollar aplicaciones web y páginas dinámicas.

Creado por Rasmus Lerdorf en 1994 como un conjunto de scripts para trabajar con formularios web, pero que ha evolucionado hasta convertirse en un completo y potente lenguaje de programación.

PHP es interpretado, lo que significa que el código fuente escrito por el desarrollador se compila en tiempo de ejecución. Esto hace que las páginas PHP sean muy dinámicas y flexibles, lo que a su vez las hace ideales para crear aplicaciones web interactivas.

2.2 ¿Para qué se usa PHP?

PHP se utiliza principalmente para el desarrollo de aplicaciones web dinámicas. Esto incluye todo, desde sitios web simples hasta aplicaciones web más complejas, como sistemas de administración de contenido (CMS), tiendas en línea, foros y redes sociales.

PHP se integra a la perfección con otras tecnologías web como HTML, CSS, JavaScript y bases de datos, lo que permite a los desarrolladores crear aplicaciones web interactivas y personalizadas para sus usuarios. Además, PHP es compatible con la mayoría de los servidores web, lo que lo hace muy popular entre los desarrolladores y las empresas que requieren aplicaciones web dinámicas y escalables.



2.3 Ventajas en el uso de PHP

- **Fácil de aprender**: PHP es un lenguaje de programación relativamente fácil de aprender para desarrolladores con experiencia en otros lenguajes de programación.
- **Flexibilidad**: Se puede utilizar en diversas aplicaciones web. Además, se puede integrar con otras tecnologías web como HTML, CSS, JavaScript y bases de datos, lo que permite a los desarrolladores crear aplicaciones web personalizadas y dinámicas.
- **Eficiencia**: PHP se ejecuta en el lado del servidor, lo que significa que el procesamiento se realiza en el lado del servidor, no en el navegador del usuario. Esto hace que las aplicaciones web sean más rápidas y eficientes.
- **Comunidad activa**: PHP tiene una gran comunidad de desarrolladores y usuarios que brindan soporte, herramientas y recursos para desarrollar aplicaciones web.
- **Coste**: PHP es gratuito y de código abierto, lo que lo convierte en una opción atractiva para empresas y desarrolladores que buscan una solución de desarrollo de aplicaciones web de bajo costo.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      cout << "Hola Mundo";
6      return 0;
7  }
```

Hola mundo en C++.

```
1  <?php
2  echo "Hola Mundo";
3  ?>
```

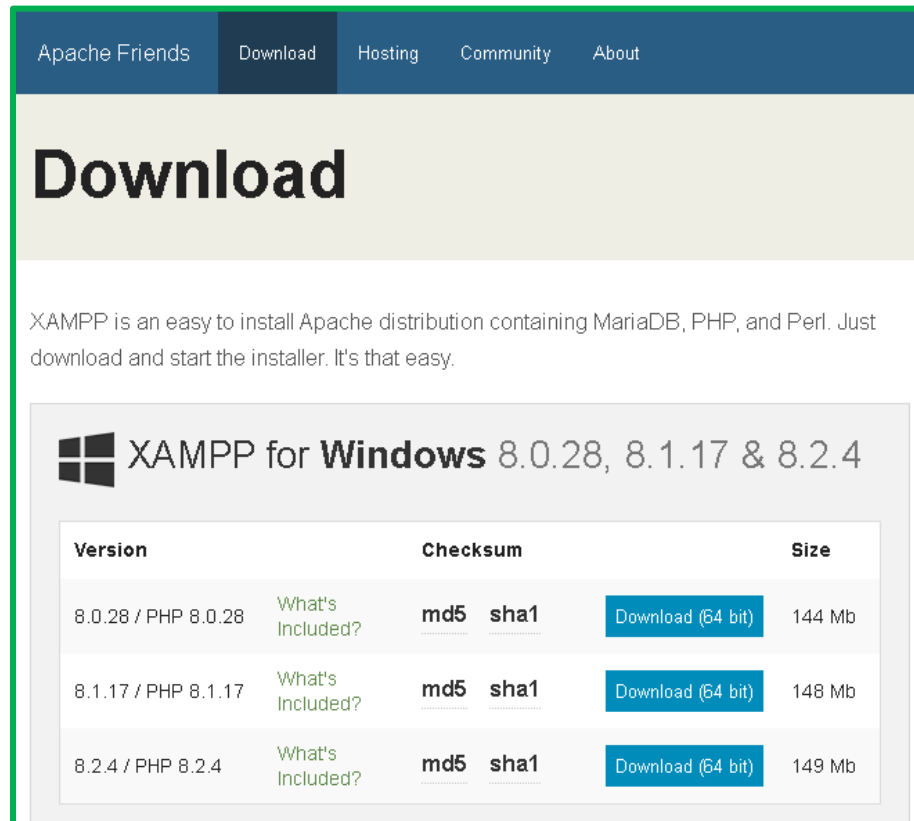
Hola mundo en PHP.

3. Configuración del entorno de desarrollo

3.1 Instalación de XAMPP

Para la instalación de XAMPP podemos seguir los primeros minutos del siguiente vídeo: [clic para ir al video](#).

El enlace a la página de [DESCARGA](#) de Xampp.

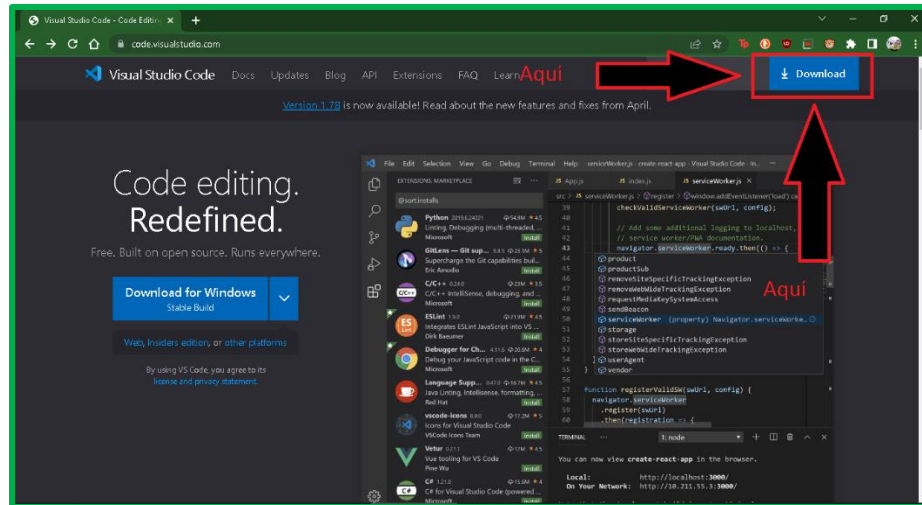


Version	Checksum	Size
8.0.28 / PHP 8.0.28	What's Included? md5 sha1	Download (64 bit) 144 Mb
8.1.17 / PHP 8.1.17	What's Included? md5 sha1	Download (64 bit) 148 Mb
8.2.4 / PHP 8.2.4	What's Included? md5 sha1	Download (64 bit) 149 Mb

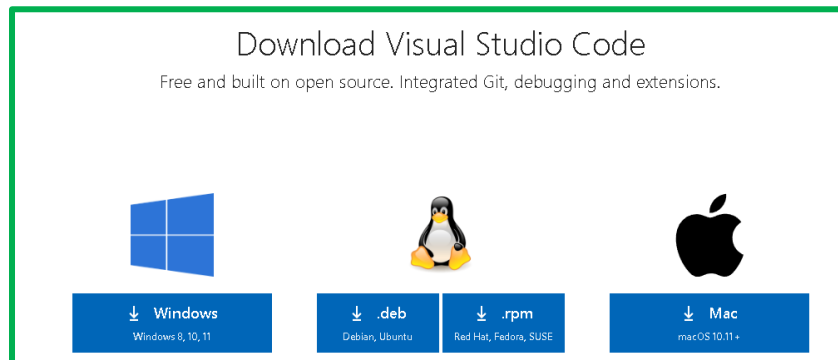
Dependiendo del SO que tengamos, buscamos la opción que más se acomode a nosotros.

3.2 Instalación de Visual Studio Code

Si nos dirigimos a la página de [Visual Studio Code](https://code.visualstudio.com) veremos que aparece la sección de descargas:



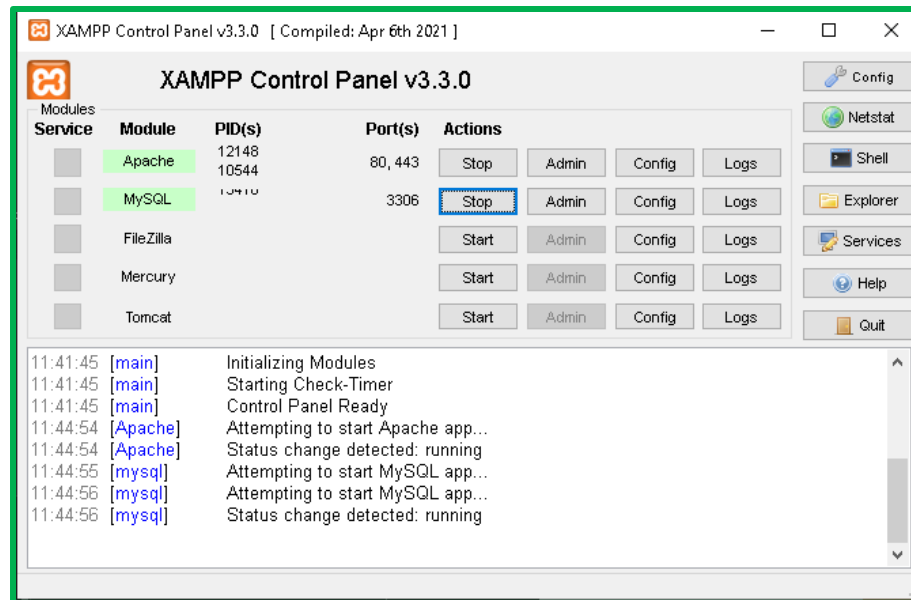
Al clicar nos aparecerá algo como esto:



Seleccionamos nuestro sistema operativo y ejecutamos el fichero que descarguemos, la instalación, como cualquier otra, dándole **si a todo** como si supiéramos lo que estamos haciendo.

3.3 Configuración de XAMPP

En esta parte mejor no toquetear mucho las cosas, si hemos seguido al chico del vídeo en la instalación, nuestro XAMPP debería verse así:



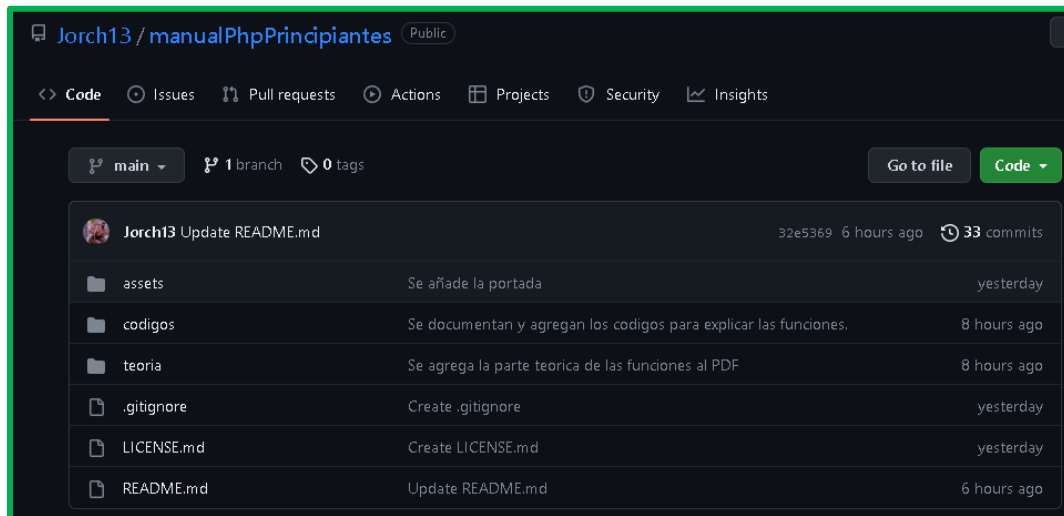
La ruta donde debemos crear nuestras webs dinámicas es: **C:\xampp\htdocs**

Una vez en ésta carpeta, podemos crear directamente nuestro Script o crear un directorio para cada web (lo mas recomendado si no queremos terminar con 231237 ficheros así: ejer1.php, ejer1Final.php, ejer1Final_finalisimo.php, ejer1-copia.php).

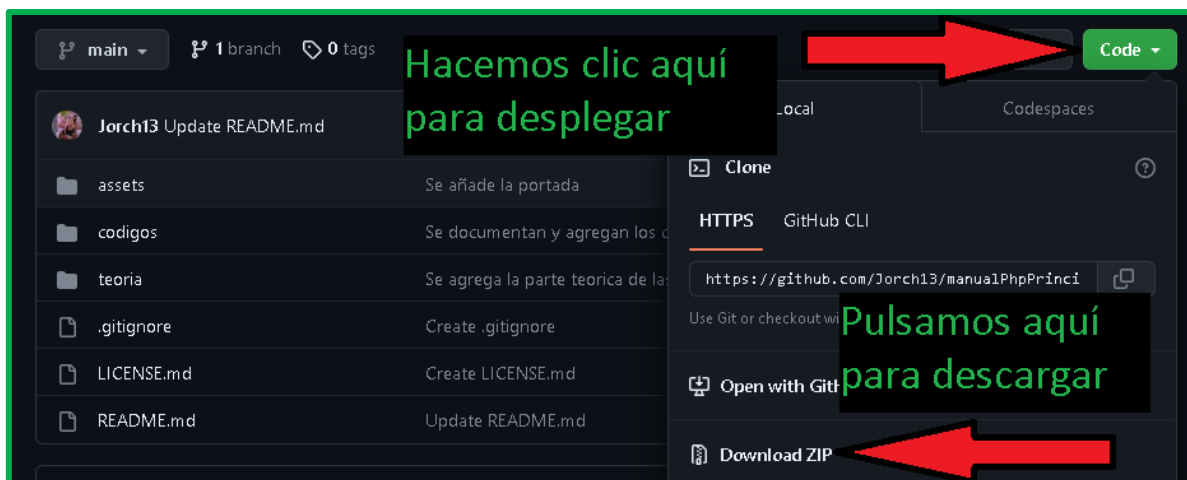
3.4 Usar códigos del manual y propios en PHP

Si ya sabes cómo funciona GitHub, simplemente descarga el repositorio en tu equipo local y salta al siguiente punto del manual.

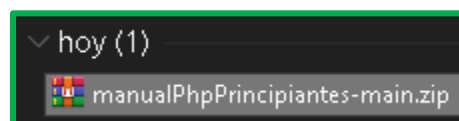
Ahora, para poder **usar los códigos** del manual, puedes descargar la carpeta **códigos** desde [aquí](#):



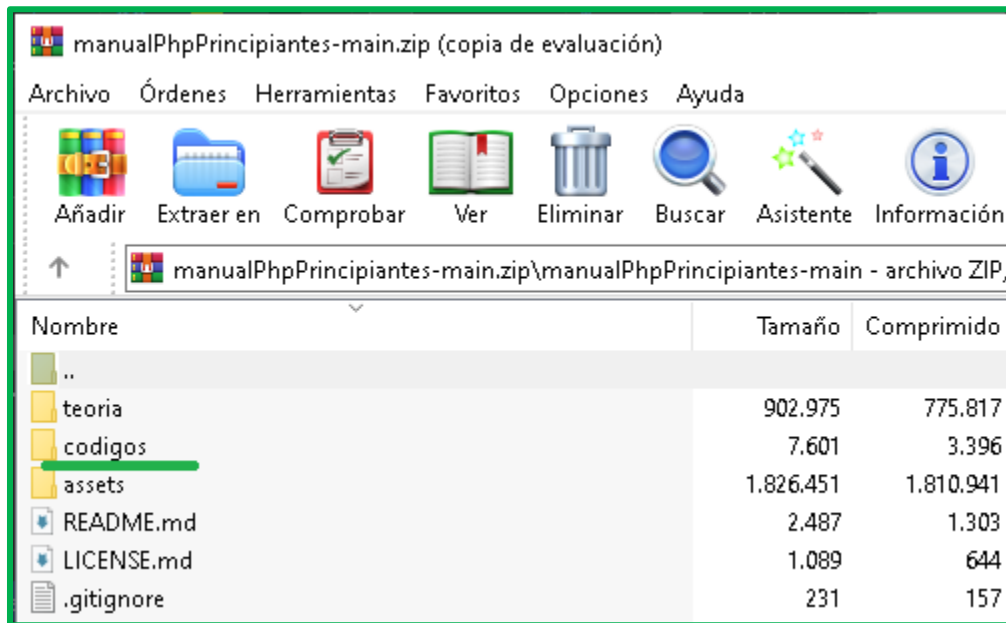
Si pulsamos en **Code** se abrirá un desplegable:



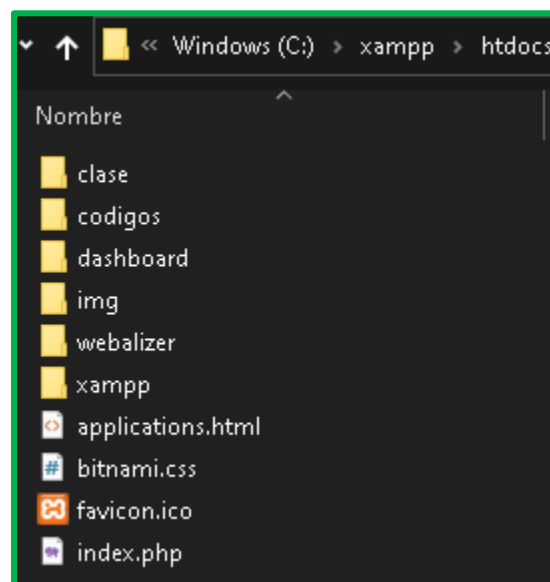
Se nos descargará un fichero comprimido en **.zip**



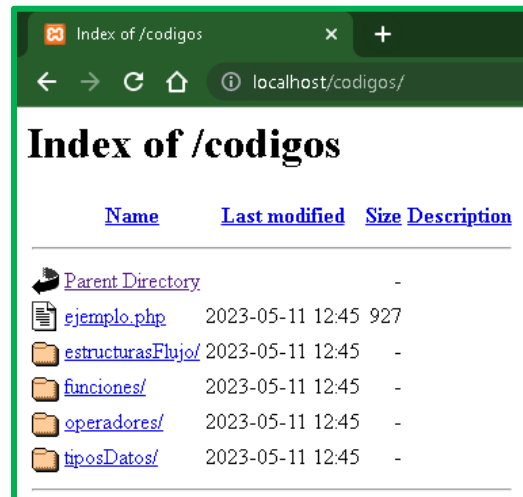
Si lo abrimos tendremos las carpetas del repositorio en nuestro ordenador



Simplemente copiamos la carpeta **códigos** en nuestra ruta **C:\xampp\htdocs**.
Quedando algo así:



Como vemos, la carpeta **códigos** ya está, para acceder a ella podemos usar: **127.0.0.1/codigos/** en la barra de búsqueda de nuestro navegador. También puede servirnos: **http://localhost/codigos/**

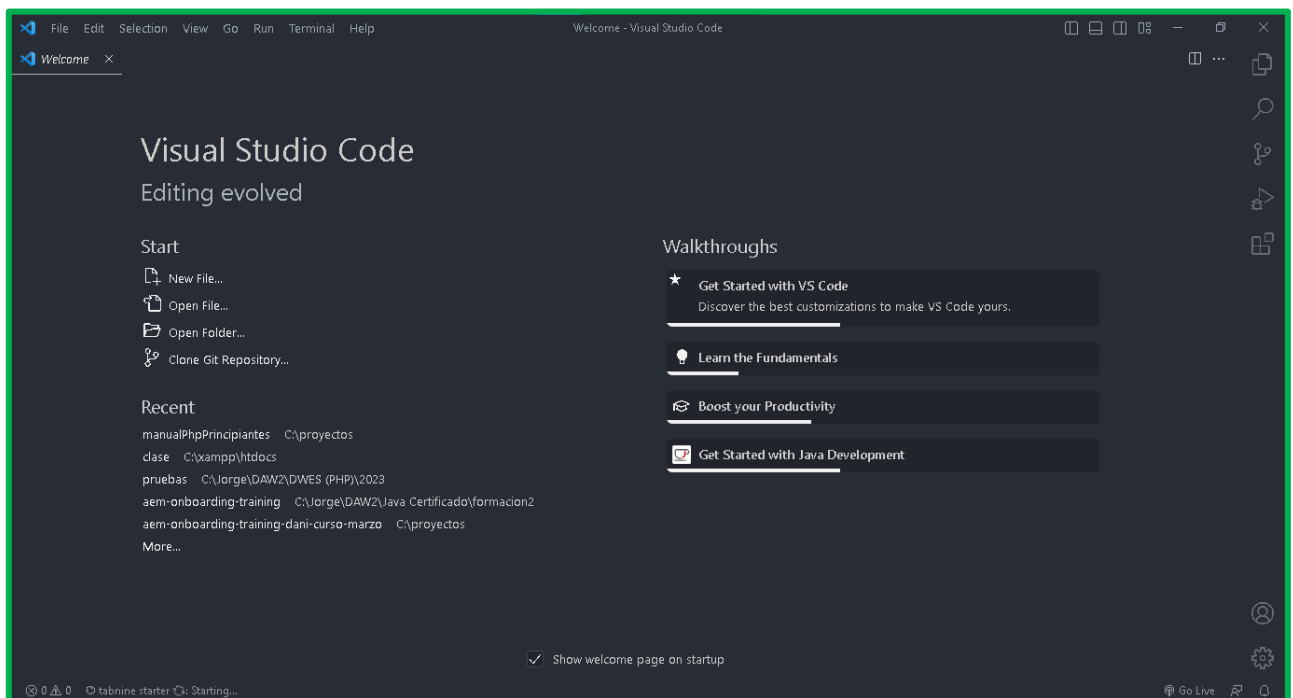


Si pulsamos en cualquiera de las carpetas, podemos acceder a los diferentes Scripts.

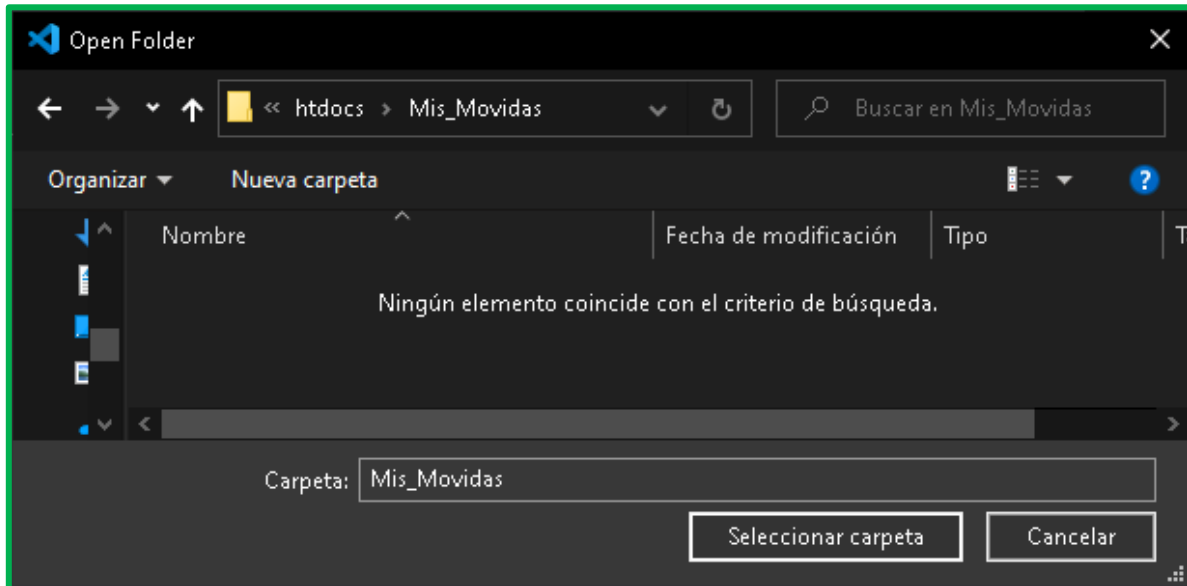
Ahora imagino que querrás crear tu propia carpeta, y meter dentro los códigos.

Para ello, desde el explorador de archivos creamos
C:\xampp\htdocs\Mis_Movidas

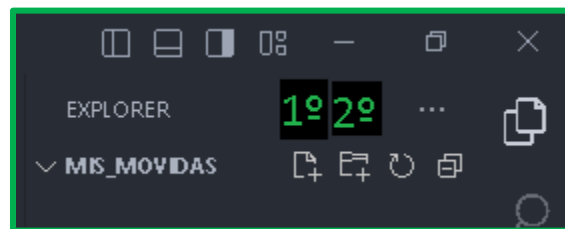
Ahora abrimos Visual Studio Code



Pulsamos en la opción **Open Folder...** y buscamos la carpeta **Mis_Movidas**



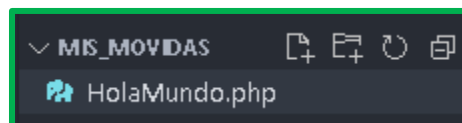
Le damos a seleccionar carpeta. Se nos abrirá en Visual Studio Code.



1º Con este botón podemos crear nuevos archivos en la carpeta

2º Con este botón podemos crear nuevas carpetas en la carpeta

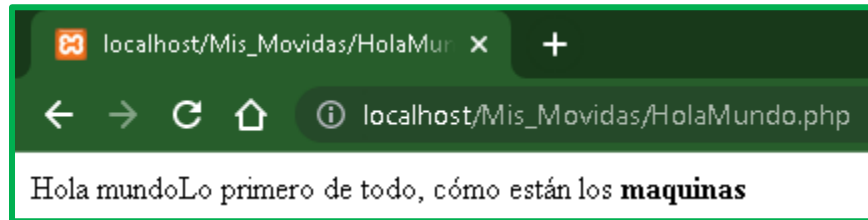
Usando el primer botón creamos **HolaMundo.php** y le damos a enter



Escribimos el ejemplo que veíamos anteriormente:

```
1 <?php
2 echo "Hola Mundo";
3 ?>
```

Si ahora buscamos en el navegador **localhost/Mis_Movidas/HolaMundo.php**



Esto, o algo así, debería aparecernos.



Cómo vemos, el maestro debe albergar siempre una lección más que el alumno.

Podemos hacer todo tipo de pruebas e ir usando lo que pillemos para probar, aunque lo más recomendable es que en tu carpeta Mis_Movidas, vayas creando otras carpetas para cada apartado de este manual o de tus proyectos propios.

Este manual no se focaliza en las etiquetas de html, se da por hecho que se comprende dicho lenguaje de marcas.

4. Variables y tipos de datos

4.1 Variables en PHP

Como ya hemos visto, bueno no se ve, pero está en el código. Hay diferentes variables en PHP ¿QUÉ ES UNA VARIABLE? Pues es una manera que tenemos de almacenar datos/info/cosas. Por ejemplo, imagina que tienes 12348 usuarios para una web de compra-venta de peonzas, cada uno de estos usuarios tendrá una edad, nombre, id (importante para las BD) una dirección, la cual nunca deberíamos de usar para fines ilícitos, esas cosillas.

Pues PHP nos permite guardar esos datos en variables, hay diferentes tipos y a continuación vamos a ver algunos de ellos, más adelante se profundiza en algunos de estos tipos de datos/variables.

4.2 Tipos de datos en PHP

Como toca, primero viene la parte teórica, estos son los tipos de datos más comunes que veremos en PHP:

- **Entero (int)**: es un número entero sin decimales, por ejemplo, 10, -5, 0, etc.
- **Flotante (float)**: es un número con decimales, por ejemplo, 3.14, -0.5, etc.
- **Cadena de caracteres (string)**: es un conjunto de caracteres alfanuméricos y especiales encerrados entre comillas simples o dobles, por ejemplo, "Hola Mundo", '123', etc.
- **Booleano (bool)**: es un tipo de dato que solo puede tener dos valores: verdadero (true) o falso (false).
- **Arreglo (array)**: es una colección ordenada de elementos que pueden ser de diferentes tipos de datos, por ejemplo, un arreglo que contiene números, cadenas de caracteres, booleanos, etc.
- **Objeto (object)**: es una instancia de una clase que contiene propiedades y métodos que pueden ser utilizados para manipular la información.
- **Nulo (null)**: es un valor especial que indica que una variable no tiene un valor asignado.

Bueno, lo normal si vienes con 0 conocimientos en lenguajes de programación, es que los booleanos, los array, los object y los null, te suenen a chino, PERO hay un truquillo y es: que, aunque tengas conocimientos, sabes que te va a tocar chapar un par de días hasta comprenderlos, como a todo KISKI.

Ahora vamos a mostrar en código un ejemplo de cada uno de estos datos para que podáis experimentar y ver qué hace cada uno. O en cristiano, copiar y pegar las cosas en vuestro Visual Studio Code (sí, durante todo el documento pienso escribirlo siempre entero, queda flama).

```
<?php
// Ejemplo de declaraciones de tipos de datos

// Tipo de datos escalares
$int_var = 10; // Entero
$float_var = 3.14; // Flotante
$string_var = "Hola mundo"; // Cadena o tring
$bool_var = true; // Booleano

// Tipo de datos compuestos
$array_var = array(1, 2, 3); // Array
$object_var = new stdClass(); // Objeto
```

Como podemos observar, la manera de crear una variable es la que sigue:

\$nombreVariable = valor;

El dólar \$ es NECESARIO, si no el Script reconocerá palabras sueltas y empezará a mandarnos Warnings o Failures, no queremos eso.

El nombre de la variable es simplemente eso, cómo va a llamarse. Mejor dicho, es la manera que tenemos de APUNTAR a esa variable dentro de nuestro Script.

El signo = es NECESARIO y nos sirve para darle un valor o valores a nuestra variable.

Ahora un poco en castellano para todos.

Imagina que tienes un html con 23329 líneas, y necesitas repetir el mismo cambio 70 veces, **OBVIAMENTE** no vas a ir buscando y cambiando cosas en el HTML, para ello, tienes implementadas ciertas variables con PHP que te ayudan a cambiar estas cosas.

Por ejemplo: Tenemos una web dinámica donde se realizan registros para que los usuarios registrados puedan acceder a su perfil, biblioteca, etc.

Podemos almacenar esos datos en variables y usarlos en el HTML:

```
// Variables de diferentes tipos
$name = "Juan";
$age = 15;
$mail = "juan@example.com";
$pass = "juan123";
$tel = 689120903;
$student = true;
```

Una vez tenemos esas variables las podemos usar en el html de la siguiente manera:

```
<!DOCTYPE html>
<html Lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Apuestas Recreativas</title> <!-- No es un error, es un chiste entre recreo y recreativas -->
  <style> ...
</style>
</head>
<body>
  <h1>Perfil de Usuario</h1>
  <?php
    print '<p>Nombre: ' . $name . '</p>';
    print '<p>Edad: ' . $age . '</p>';
    print '<p>Correo: ' . $mail . '</p>';
    print '<p>Telefono: ' . $tel . '</p>';
    print '<p>Es estudiante: ' . $student . '</p>';

  ?>
</body>
</html>
```

No te preocupes, encontrarás este fichero en el repositorio:

manualPhpPrincipiantes/codigos/tiposDatos/formUser.php

Más adelante ya veremos cómo se trabaja con formularios, con variables y el HTML. Ya que todos sabemos que no es recomendable mezclar php, html, css, JS, en un mismo fichero ahí todo apelotonado que ni sabemos lo que hay y eso que es “nuestro” código (agradecer estas comillas a [Stack Overflow](#)).

4.3 Conversión de tipos de datos

A veces, vamos a necesitar convertir ciertos datos a otro tipo para realizar operaciones o comparaciones dentro de nuestros Scripts, en PHP tenemos diferentes métodos para llevar a cabo estas conversiones:

- **Conversiones implícitas**: PHP realiza automáticamente conversiones implícitas de tipos de datos en ciertas situaciones. Por ejemplo, si se agrega un número entero a un flotante, PHP convertirá automáticamente el entero en un flotante antes de realizar la operación.
- **Conversiones explícitas**: puede convertir explícitamente un tipo de datos a otro utilizando los siguientes métodos:
 - **(int) o intval()**: convierte el valor en un número entero.
 - **(flotante) o floatval()**: Convierte el valor en un flotante.
 - **(cadena) o strval()**: convierte el valor en una cadena.
 - **(bool) o boolval()**: Convierte el valor a un valor booleano.
 - **(matriz) o settype()**: convierte el valor en una matriz.
 - **(objeto)**: Convierte el valor en un objeto.

Es importante tener en cuenta que las conversiones de tipos de datos pueden provocar la pérdida de precisión o información, por lo que es importante probar y verificar que los resultados sean los esperados.

```
// Conversión de escalares a otros tipos
$string_to_int = (int) $string_var; // Cadena a entero
$float_to_int = (int) $float_var; // Flotante a entero
$string_to_bool = (bool) $string_var; // Cadena a booleano

// Conversión de arrays a otros tipos
$array_to_object = (object) $array_var; // Array a objeto
$array_to_string = implode(',', $array_var); // Array a cadena

// Conversión de objetos a otros tipos
$object_to_array = (array) $object_var; // Objeto a array
$object_to_string = json_encode($object_var); // Objeto a cadena JSON
```

Una vez más, ha tocado meterse una pequeña chapita teórica, pero ahora continuamos con lo bueno, la práctica, el ensayo, la prueba y el error COMO ME GUSTA DIOOOOSSS.

No te preocupes si no comprendes cosas cómo **objeto**, **Array**, **JSON**, etc. Ya tendrás tiempo de desquiciarte con ello más adelante.

5. Operadores

5.1 ¿Qué es un operador?

Si, así de golpe operadores. No, los operadores no son los que te operan en el quirófano, esos son quiropractores...

Un operador, en PHP, suele ser un símbolo (+, -, *, etc) o palabra clave que usamos para realizar una operación `$factura1 + $factura2` (para sumar).

También es importante saber que el símbolo `=` es un operador y se usa para asignar valores, comparar `==` / `===`, etc.

5.2 Operadores aritméticos

- **Suma:** El operador `+` se usa para sumar dos valores.
- **Resta:** `-` el operador se utiliza para restar un valor de otro.
- **Multipliación:** El operador `*` se usa para multiplicar dos valores.
- **División:** El operador `/` se utiliza para dividir un valor por otro.
- **Módulo:** El operador `%` se usa para obtener el resto de dividir un valor por otro.
- **Incremento:** El operador `++` se utiliza para aumentar el valor de una variable en uno.
- **Decremento:** El operador `--` se utiliza para disminuir el valor de una variable en uno.

```
$resultadoSuma = $numero1 + $numero2;  
$resultadoResta = $numero1 - $numero2;  
$resultadoMultiplicacion = $numero1 * $numero2;  
$resultadoDivision = $numero1 / $numero2;  
$resultadoModulo = $numero1 % $numero2;
```

Cómo vemos, el símbolo `=` asigna el valor de la operación a `$resultadoSuma`, resta, etc... Cada una de estas operaciones usa un operador de los antes mencionados.

5.3 Operadores de comparación

Estos operadores se usan para comparar dos valores.

Los operadores de comparación devuelven un valor booleano (true o false) dependiendo del resultado de la comparación.

Estos son los diferentes operadores de comparación que tiene PHP:

```
<?php
$numero1 = 13;
$numero2 = 7;

> if($numero1 == $numero2){ ...
> } else { ...
> }
> if ($numero1 != $numero2) { ...
> } else { ...
> }
> if ($numero1 === $numero2){ ...
> } else { ...
> }
print "<p>";
> if($numero1 > $numero2){ ...
> } else { ...
> }
print "<p>";
> if($numero1 !== "10"){ ...
> } else { ...
> }

?>
```

- **Igualdad**: El operador **==** se utiliza para comprobar si dos valores son iguales.

- **Identidad**: El operador **===** se utiliza para comprobar si dos valores son iguales y del mismo tipo.

- **Desigualdad**: Los operadores **!=** o **<>** se utilizan para probar si dos valores son diferentes.

- **Sin Identidad**: El operador **!==** se utiliza para comprobar si dos valores son diferentes o no del mismo tipo.

- **Mayor que**: El operador **>** se utiliza para verificar si un valor es mayor que otro.

- **Mayor o igual que**: El operador **>=** se utiliza para comprobar si un valor es mayor o igual que otro valor.

- **Menor que**: el operador **<** se usa para verificar si un valor es menor que otro valor.

- **Menor o igual que**: El operador **<=** se utiliza para verificar si un valor es menor o igual que otro valor.

Más adelante veremos los **if** estos que estás viendo aquí, si no conoces estas estructuras es mejor no preocuparse aún por ellas. Al final de este punto se indica la ruta de los diferentes ficheros para entender los operadores de cada tipo.

5.4 Operadores lógicos

En PHP, hay tres operadores lógicos que se usan para combinar varias expresiones lógicas: and, or y not.

- **and (&&)**: Devuelve true si ambas expresiones son true.
- **or (||)**: Devuelve true si al menos una de las expresiones es true.
- **not (!)**: Devuelve true si la expresión es false.

```
<?php
$numero1 = 13;
$numero2 = 7;
$numero3 = 54;

if($numero1 > $numero2 and $numero1 > $numero3){ ...
} elseif($numero2 > $numero1 and $numero2 > $numero3){
} else { ...
}

print "<br>";
if($numero1 > $numero2 or $numero3 > $numero2){ ...
} else { ...
}

print "<br>";
if(!($numero1 == $numero2)){ ...
} else { ...
}

?>
```

Podemos encontrar los 3 ficheros para estos operadores en la siguiente ruta:

manualPhpPrincipiantes/codigos/operadores/Aritmeticos.php

manualPhpPrincipiantes/codigos/operadores/Comparacion.php

manualPhpPrincipiantes/codigos/operadores/Logicos.php

6. Estructuras de control de flujo

6.1 ¿Qué son las estructuras de control de flujo?

Una estructura de control de flujo nos permite ejecutar o no, ciertas partes de nuestro código. Esto resulta muy útil a la hora de realizar ciertas tareas cuando se cumplen determinadas condiciones, desde mostrar o no ciertos datos hasta asignar valores a variables de nuestros Scripts.

6.2 Estructuras condicionales (if, else; elseif...)

Las estructuras condicionales en PHP son una parte fundamental de la programación, ya que permiten que el programa tome decisiones y ejecute diferentes bloques de código en función de una o varias condiciones. En PHP, las estructuras condicionales se implementan a través de las siguientes palabras clave: if, else y elseif.

- **if**: Comprueba si una condición es verdadera o falsa, y ejecuta el código

```
$n1 = 13;
$n2 = 7;
$n3 = 54;

if ($n1 > $n2) {
    //Si se cumple hace esto:
    print 'Hasta aquí vas bien maquina';
}
?>
```

Esto nos sirve también en estas condiciones:

```
$palabra = 'Burrito';

if (!empty($palabra)) {
    print $palabra;
}
```

¡Como vemos comprobamos si \$palabra **NO!** está vacía.

Usando empty(\$variable) comprobamos si una variable está vacía, devuelve 'true' si está vacía y 'false' si contiene algo. ¡Luego con el símbolo ! indicamos que queremos lo contrario / distinto.

- **Else**; Se usa siempre junto al **if** para ejecutar una parte del código en caso de que la condición no se cumpla:

```
if ($n1 < $n2) {  
    //Si se cumple hace esto:  
    print 'Hasta aquí vas bien maquina';  
} else {  
    print 'Estas en el else';  
}  
?>
```

Tomando el ejemplo de antes y modificándolo un poco, vemos cómo esta vez entrará en la parte del `else {}` y ejecutará ese código.

- **Elseif**: Esta estructura se usa para comprobar más condiciones y ejecutar otros bloques de código en función de la que se cumple:

```
$usuario = 'Pedro';  
if ($usuario == 'Jorge') {  
    // Ejecuta si la condición1 es verdadera  
    print 'Hola ' . $usuario . ' has iniciado sesión';  
} elseif ($usuario == 'Pedro') {  
    // Ejecuta si la 1ªCondición es falsa y ésta es verdadera  
    print 'Hola ' . $usuario . ' has iniciado sesión';  
} elseif ($usuario == 'Hermenegilda') {  
    // Ejecuta si la 1ª y 2ªCondición son falsas y la 3ªCondición es verdadera  
    print 'Hola Hermen, tu tienes más flow que naide';  
} else {  
    // Ejecuta si todas las condiciones son falsas  
    print '¿Te quieres colar sin ser usuario?';  
}
```

Importante fijarse en que cada condición usa `==` para comprobar, si usásemos `=` le daríamos el valor en el primer `if` y siempre ejecutaría ese código. También podemos usar `===` y funcionaría, además comprobaría que la variable no solo tenga el valor, si no también el tipo, en este caso **String**.

6.3 Estructuras de repetición (While, do-while, foreach)

Este tipo de estructura nos resulta MUY útil ya que permite ejecutar VARIAS veces una parte del código mientras se cumpla una condición. Ya va sonando eso de “cumplir condiciones”.

- **While**: Siempre que una condición sea cierta, se ejecuta el código:

```
<?php
$numero = 1;

while ($numero <= 10) {
    print $numero;
    $numero++;
}
```

Ejemplo sencillo, simplemente pinta los números del 1 al 10, incluido el último gracias al símbolo <=.

- **Do-while**: Ejecutar un bloque de código **AL MENOS UNA VEZ**, y después seguir ejecutándolo siempre que una condición sea cierta:

```
$num2 = 13;
do {
    print 'JEJEJEJEJj 13 13 13';
} while ($num2 < 10);
```

Como podemos observar, al menos la primera vez, escribirá el Script. Ya que como vemos, \$num2 vale 13, que es mayor que 10, no menor, una vez compruebe el **while()** pararápapá.

- **Foreach:** Virgen de la macarena. Como este manual es para principiantes solo diré que usamos este bucle (así se llama también a las estructuras de repetición en mi barrio) para recorrer los elementos de un Array o de un Objeto:

```
$personas = array(  
    'Pepe', 'Manoli', 'Juanito', 'Rosalinda', 'Victor Manuel'  
);  
  
foreach ($personas as $nombre) {  
    print $nombre.', '  
}
```

Como vemos, la estructura es **foreach (\$Array as \$valor)**.

Lo que realiza es un recorrido por todo el contenido de **\$personas** y a cada cosa que encuentra le da el alias **\$nombre** una vez dentro de la estructura, pinta ese **\$nombre**.

Esto no es perfecto, ya que después de escribir Victor Manuel, nos pondrá una coma. Para evitarlo hay que usar más código que ahora no toca.

```
$tanques = array("Leopard", "M1A2 Abrams", "Challenger 2", "AMX-56 Leclerc");  
foreach ($tanques as $indice => $tanque) {  
    echo 'Top: ' . $indice . ', Tanque: ' . $tanque . '<br>';  
}
```

Aquí observamos otra forma de recorrer los Arrays con el bucle foreach. En esta ocasión también le damos un alias al índice.

6.4 Interrupción de estructuras de repetición (**break**, **continue**)

Las interrupciones en las estructuras de control son herramientas importantes en PHP que permiten controlar el flujo del programa mediante la interrupción de un bucle o la omisión de una iteración.

Las más comunes son **break** y **continue**.

- **Break**: Si se utiliza dentro de un bucle, **break** interrumpe la ejecución del bucle en el momento en que se encuentra:

```
$numeros = array(1, 3, 2, 5, 4, 6, 7, 9);

foreach ($numeros as $numero) {
    if ($numero % 2 == 0) {
        break;
    }
    echo $numero . '<br>';
}
```

Cómo se observa, el primer número que cumpla la condición del **if()** hará que el Script entre y ejecute el **break**, parando el programa.

- **Continue**: Se usa para saltar una iteración en un bucle si se cumple una determinada condición:

```
$numeros = array(1, 3, 2, 5, 4, 6, 7, 9);

foreach ($numeros as $numero) {
    if ($numero % 2 == 0) {
        continue;
    }
    echo $numero . ', ';
}
```

Cambiando un poco lo anterior, podemos hacer que solo se muestren los números impares.

7. Funciones

7.1 ¿Qué son las funciones?

Ya he usado varias veces la palabra “reutilizar” para referirme a partes del código que queremos repetir en determinadas condiciones.

Las funciones son eso, bloques de código que realizan cierta tarea y podemos reutilizarlas en nuestro Script (Script, código, ... Llámalo x llámalo y)

Muy fácil, los que **recibe** la función dentro de los paréntesis (), en este caso son ya variables \$precio, \$marca, \$modelo, a estas se les llama parámetros.

El nombre no necesita ser el mismo.

7.2 Definición y llamado de funciones

Para crear nuestra función basta con usar la palabra **function** seguido de **nombreFuncion (param1, param2, ...) {código de función...}**

```
function creaMotosierras($precio, $marca, $modelo)
{
    $motoSierra = array(
        "precio" => $precio,
        "marca" => $marca,
        "modelo" => $modelo
    );
    return $motoSierra;
}
```

Una vez tenemos definida nuestra función **creaMotosierras** podemos usarla mediante una llamada.

```
$Stihl = creaMotosierras(120, 'stihl', 'tira troncales');
$Husqvarna = creaMotosierras(150, 'husqvarna', 'parte piernas');
```

Como se puede ver, al llamar a la función nos va a devolver “algo”, entonces necesitamos almacenarlo y ¿Qué usamos para almacenar cosas? Efectivamente, VARIABLES.

Así que ahora mismo en **\$Stihl** tendremos un Array y en **\$Husqvarna** otro Array, cada uno con los datos de cada Motosierra.

7.3 Parámetros y argumentos de las funciones

Si, sé que ahora mismo todo eso de param1, param2 que has leído arriba te está comiendo el coco, observa la imagen detenidamente.

Tenemos la función **crearMotosierras** que nos va a devolver(**return**) un **array** cuyos índices son “precio”, “marca” y “modelo”, PERO ¿QUÉ VALORES VAN A TENER?

En PHP, los parámetros son valores que se pasan a una función cuando se llama. Los parámetros se definen en la definición de la función, dentro de los paréntesis después del nombre de la función. Los parámetros son variables que recibe una función para utilizarlas en la ejecución de esta. Una función puede recibir cualquier número de parámetros, incluso ninguno.

Y los argumentos ¿QUÉ SON LOS ARGUMENTOS ENTONCES? Pues son los valores que se usan al llamar a la función.

```
$Stihl = creaMotosierras(120, 'stihl', 'tira troncales');  
$Husqvarna = creaMotosierras(150, 'husqvarna', 'parte piernas');
```

Volviendo al ejemplo anterior, los argumentos serían 120, 'stihl' y 'tira troncales' para la 1ª llamada. ¿Sabrías identificar los argumentos de la 2ª llamada?

Es importante tener en cuenta que los argumentos se pasan por valor por defecto en PHP, lo que significa que se crea una copia del valor de la variable original. Por lo tanto, si se modifica el valor del argumento dentro de la función, no afectará a la variable original.

No vamos a tener problemas con esto si somos principiantes, pero en códigos extensos podemos perder el valor real de una variable por el camino, si queremos evitarlo podemos usar el paso por referencia.

```
//Paso por referencia usando &  
function duplicar(&$num) {  
    $num *= 2;  
}  
$numero = 5;  
duplicar($numero);  
echo $numero; // imprime "10"
```

Como digo, ahora no es necesario saber que es el paso por referencia, ya habrá tiempo.

8. Arrays

8.1 ¿Qué son los Arrays?

Igual debería haber metido esta parte más adelante, pero me parecía importante dedicarle un punto concreto a los Arrays en una guía para principiantes ya que es mi guía y a mí personalmente me costaron un poquito de entender en su momento. Así que quería darles su huequito especial.

Los arrays en PHP son un tipo de variable, eso ya lo hemos visto en esta guía, pero si seguís cayendo en el vicio de programar os enfrentareis mucho con ellos.

Básicamente es una colección de cosas, así que a fin de cuentas es una variable, estas colecciones pueden almacenar datos, otros arrays, objetos (en nivel principiante no se ven arrays de objetos porque es mi guía y punto).

8.2 Creación y manipulación de arrays

La primera es la forma clásica de hacerlo, con los corchetes [] y metiendo datos.

Estos datos pueden ser variables ya declaradas u otro tipo de cosas.

```
//Declarar los array de diferentes maneras
$arrayMarcasCoches = ['Nissan', 'Toyota', 'Opel', 'Mercedes'];
$marcasCoches = array('Nissan', 'Toyota', 'Opel', 'Mercedes');
```

Se puede ver que PHP nos proporciona el método **array()** que al ser llamado crea un Array. Esto nos sirve para crear Arrays vacíos, por ejemplo.

```
//Array vacio, puede usarse luego para almacenar datos
$mensajes = array();

$mensajes[0] = 'Este es el primer mensaje de tu array de mensajes';

print_r($mensajes);
```

8.3 Arrays asociativos

Al igual que los anteriores, son colecciones de datos o cosas que podemos almacenar con un **índice** y un **valor**.

PHP nos ofrece la posibilidad de usar nuestros **propios índices** para los arrays que creamos, de la siguiente manera:

```
//Arrays asociativos
$ventas = array(
    'Patatas' => 300,
    'Churros' => 200,
    'Porras' => 760
);

//En estos tenemos índices personalizables a los que podemos acceder
print $ventas['Patatas'].', '; //Mostrará 300
print $ventas['Churros'].', y '; //Mostrará 200
print $ventas['Porras']; //Mostrará 760
```

Es importante saber que existen muchas funciones (métodos) en PHP para trabajar con Arrays, para recorrerlos, modificarlos, mostrar determinados elementos, comparar, etc.

8.4 Métodos más comunes de Arrays

Ahora vamos a ver varios métodos, no es necesario saberlos todos de memoria, pero cuántos más y mejor manejeis, más sencillo resultará en un futuro trabajar con Arrays.

- **array()**: Crea un nuevo array, ya lo hemos visto
- **count()**: Cuenta los elementos en el array:

```
//Métodos de Arrays
$Frutas = array('Manzana', 'Pera', 'Papaya', 'Platano', 'Fresa');

// count() para contar los elementos del array
$contFrutas = count($Frutas);
print $contFrutas; //Devuelve 5
```

- **in_array()**: Comprueba si un valor está presente en el Array.

```
//in_array() devuelve true o false dependiendo si el valor existe en el Array
$hayValor = in_array('Pera', $Frutas);
print '<h4>Comprobar si existe un valor // in_array(): </h4>';
if ($hayValor === true) {
    print 'El valor se encuentra en el Array';
} else {
    print 'El valor no se encuentra en el Array';
}
```

Usamos las estructuras de control para ejecutar un código u otro en función de si nos devuelve true o false al realizar **in_array()** y almacenarlo en **\$hayValor**.

- **array_push()**: Añade un elemento al final del array:

```
//array_push() agrega un elemento al final del array
array_push($Frutas, 'Melon');
print_r($Frutas);
```

Cómo vimos que era una función y un parámetro, aquí vemos que el método **array_push()** recibe el array al que queremos agregar el elemento como 1º parámetro y el elemento como 2º parámetro. Se pueden agregar más elementos e irán introduciéndose en orden al final del array.

- **array_pop()**: Elimina y guarda el último elemento de un Array:

```
//array_pop() elimina el ultimo elemento del array y lo almacena
$ultimaFruta = array_pop($Frutas);
print '<p>Sacando el ultimo elemento de $Frutas con array_pop($Frutas): </p>';
print 'Ultima fruta: '.$ultimaFruta;
```

Una vez más, tenemos el método, al que le pasamos un parámetro y nos devuelve algo.

- **array_shift()**: Elimina y guarda el primer elemento de un Array:

```
//array_shift() elimina y almacena el primer elemento del Array
$primeraFruta = array_shift($Frutas);
print '<h4>Sacando el primer elemento // array_shift($Frutas): </h4>';
print ('Primera fruta: '.$primeraFruta);
```

- **array_unshift()**: Añade uno o varios elementos al principio del Array:

```
//array_unshift() añade elementos al principio del Array
array_unshift($Frutas, 'Mora', 'Uva', 'Naranja');
print '<h4>Añadir elementos al principio del Array // array_unshift(): </h4>';
var_dump($Frutas);
```

Al igual que ocurría con array_push(), podemos añadir más de un elemento.

- **sort()**: Ordena los elementos en orden ascendente.

```
//sort() ordena los elementos en orden ascendente
sort($Frutas);
print '<h4>Ordenando el Array en orden ascendente // sort(): </h4>';
var_dump ($Frutas);
```

Aquí no almacenamos el resultado de sort() ya que trabaja directamente sobre el Array.

- **rsort()**: Ordena los elementos en orden descendiente.

```
//rsort() ordena los elementos en orden Descendente
rsort($Frutas);
print '<h4>Ordenando el Array en orden Descendente // rsort(): </h4>';
var_dump($Frutas);
```

De igual manera, tampoco almacenamos el resultado de rsort()

Para los métodos de Arrays asociativos usaremos \$ventas, cuyos índices son el producto y los valores, el numero de ventas.

```
$ventas = array(  
    'Patatas' => 300,  
    'Churros' => 200,  
    'Porras' => 760,  
    'Melones' => 400,  
    'Cobre' => 300,  
    'Ajos' => 125  
);
```

- **asort()**: Ordena un array asociativo en orden ascendente, según el valor.

```
//asort() ordena un array asociativo en orden ascendente por los valores  
asort($ventas);  
print '<h4>Ordenando un Array asociativo por valores // asort(): </h4>';  
var_dump($ventas);
```

- **ksort()**: Ordena un array asociativo en orden ascendente, según la clave.

```
//ksort() ordena un array asociativo en orden ascendente por las claves  
ksort($ventas);  
print '<h4>Ordenando un Array asociativo por claves // ksort(): </h4>';  
var_dump($ventas);
```

- **array_key_exists()**: Comprueba si una llave está presente en el Array.

```
//array_key_exists() devuelve true o false si existe o no cierta clave en un array  
$existe = array_key_exists('Porras', $ventas);  
print '<h4>Comprobar si existe una clave en un Array // array_key_exists(): </h4>';  
if ($existe === true) {  
    print 'Si existe';  
} else {  
    print 'No existe';  
}
```

Aquí usamos las estructuras de control **if** y **else** para manejar el comportamiento de nuestro código dependiendo de si **\$existe** es *true* o *false*. Usamos el **operador de comparación ===** para comparar que el valor y el *tipo* son iguales.

- **array_flip()**: Intercambia las claves y valores de un Array.

```
//array_flip() para darle la vuelta a las claves y valores de un Array
$vuelta = array_flip($ventas);
print '<h4>Intercambiando las llaves por los valores // array_flip(): </h4>';
var_dump($vuelta);
```

Es importante comprobar qué métodos devuelven (**return**) cosas y cuales realizan una acción directa.

Estos son los métodos que, personalmente, considero útiles a nivel principiante en PHP, pero hay muchos otros.

Aquí tampoco se hace hincapié especial en la iteración (recorrer) de Arrays, ya que es un apartado para un mayor nivel.

A modo curiosidad para futuras guías más avanzadas, hay que saber que existen arrays que contienen otros arrays, esto tiene la finalidad de hacer los datos más accesibles para los desarrolladores:

```
//Podemos declarar arrays dentro de arrays si queremos
$arrayDosDimensiones = array(
    'Notas' => array(
        'Juan' => array(
            'Lengua' => 5,
            'Mate' => 7,
            'Economia' => 3,
            'Latin' => 8
        ),
        'Pedro' => array(
            'Lengua' => 9,
            'Mate' => 10,
            'Economia' => 8,
            'Latin' => 3
        )
    ),
    'Faltas' => array(
        'Juan' => array(
            'Lengua' => 2,
            'Mate' => 0,
            'Economia' => 9,
            'Latin' => 0
        ),
        'Pedro' => array(
            'Lengua' => 0,
            'Mate' => 0,
            'Economia' => 0,
            'Latin' => 9
        )
    )
);
```

Como puede verse, el 1ºArray es **\$arrayDosDimensiones** con los índices “Notas” y “Faltas”. Estos índices son cada uno **otro array**, el array **\$arrayDosDimensiones[‘Notas’]** tiene como índices “Juan” y “Pedro”, QUE A SU VEZ SON **CADA UNO OTRO ARRAY**, que ahora ya sí, son arrays asociativos con los índices personalizados de “lengua”, “mate”, etc y sus valores 2, 0, 9, ...

De esta manera, podemos acceder al dato que queramos.

9. Trabajo con formularios

9.1 Métodos GET y POST

Ya he mencionado que en este manual no se ven etiquetas html, se da por sabido.

Pero es importante recordar cómo funcionan los formularios:

```
<form action="" method="post">1º
  <label for="name">Nombre: </label> 2º
  <input type="text" name="name" id="name">

  <label for="age">Edad: </label>
  <input type="number" name="age" id="age">

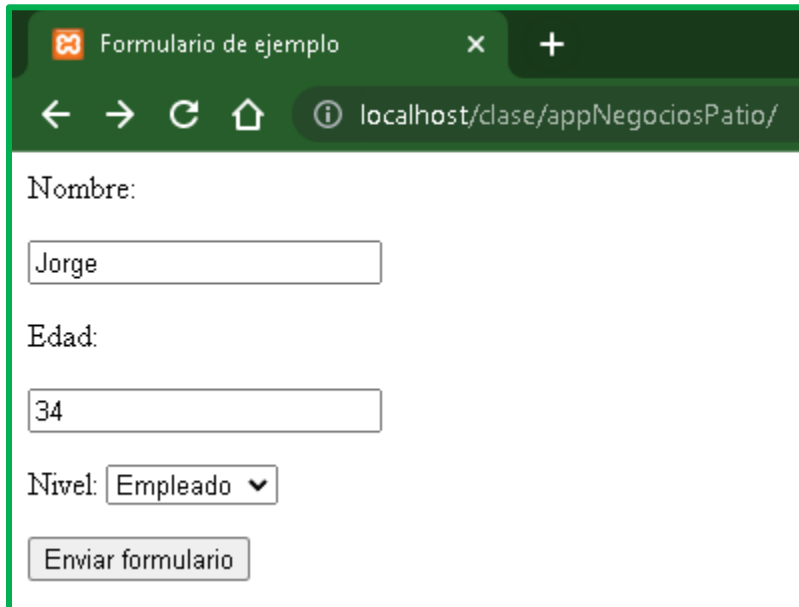
  <label for="level">Nivel: </label>
  <select name="level" id="level"> 3º
    <option value="student">Estudiante</option>
    <option value="employed">Empleado</option>
    <option value="teacher">Enseñador</option>
  </select>

  <input type="submit" value="Enviar formulario"> 4º
</form>
```

1º **<form action="" method="post">**: Aquí abrimos el formulario (cerrado con **</form>** en la última línea) El atributo **action** indica a donde se envían los datos una vez usemos el botón de Enviar, en este caso aún está vacío.

Method es otro atributo que indica a la pagina cómo enviar los valores. Hay dos posibles valores: **post** (los datos del formulario se envían en el cuerpo de la solicitud HTTP) y **get** (los datos del formulario se envían en la URL del navegador, lo que significa que los datos del formulario se pueden ver en la barra de direcciones del navegador).

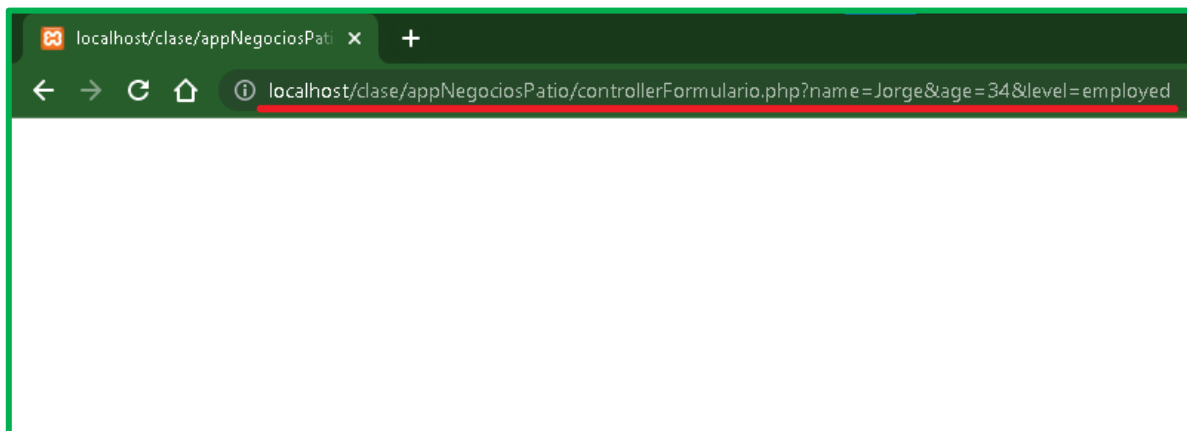
Para que se entienda:



Este es el formulario que muestra nuestro navegador.

```
<form action="controllerFormulario.php" method="get">
```

Cambiamos los valores, para enviar los datos y mostrarlos en **controllerFormulario.php** con el método **get**.

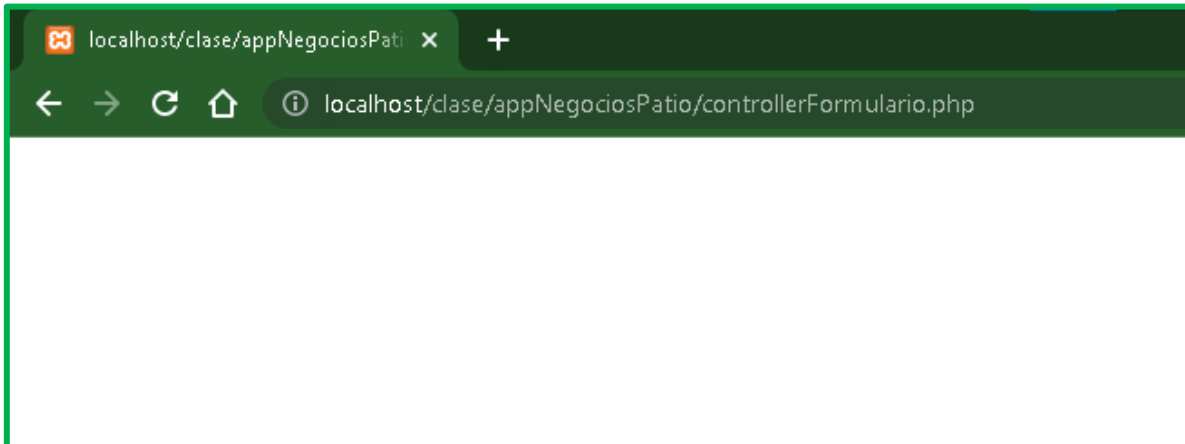


Si rellenamos el formulario y le damos a enviar, vemos que no muestra nada, ya que **controllerFormulario.php** está vacío, pero vemos que en la URL se muestran las etiquetas y los valores de nuestro formulario.

Vamos a ver ahora que ocurre si usamos el **method="post"**

```
<form action="controllerFormulario.php" method="post">
```

Cambiamos el valor de **method** a **post** y volvemos a enviar nuestro formulario



Cómo vemos, no muestra nada (**controllerFormulario.php** SIGUE VACIO) pero ya no están nuestros datos en la URL.

Esto es útil también para evitar que nos introduzcan datos fraudulentos en nuestra web. Imaginate si pudieras cambiar el valor de un formulario en Paypal desde la URL... Cuántas camisetas de Chayanne... Bueno sigamos para ver cómo mostramos y accedemos a estos datos, ahora que sabemos enviarlos por URL (**get**) o en el cuerpo de la petición http(**post**)

9.2 Acceso a variables de formularios

Si se ha hecho tanto hincapié en los Arrays y variables, aunque esto sea nivel principiante, es sólo para comprender mejor como accedemos a los datos y cómo los manejamos. Bien, nos dirigimos a **/Mis_Movidas/formularios/controllerFormulario.php** fichero que deberías haber creado ya y tener vacío.

Recuerda que en el formulario, cada campo tenía un atributo **name="valor"**

```
<label for="name">Nombre: </label>
<p><input type="text" name="name" id="name"></p>

<label for="age">Edad: </label>
<p><input type="number" name="age" id="age"></p>
```

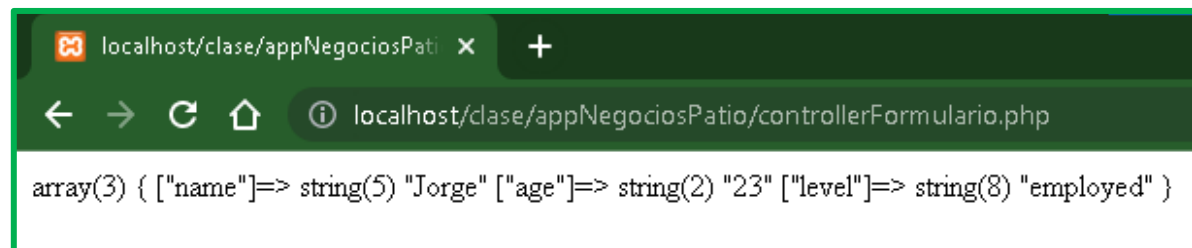
Pues usando eso **valores** (**name** y **age**) y un **Array global** que nos da PHP, podemos acceder y manejar los datos que envía el formulario.

Cómo se ha mencionado, PHP nos ofrece una serie de métodos para trabajar con Arrays, pero PHP también tiene más **palabras reservadas** para cosas, como en este caso, un Array con los datos de este formulario (bueno, son 3, pero el más recomendado es 1)

Para verlo fácil, vamos a usar algo que se ha visto en los códigos **var_dump()** que lo hemos usado para mostrar Arrays o variables en pantalla. Simplemente escribimos esto en **controllerFormulario.php**:

```
<?php  
var_dump($_POST);
```

Rellenamos el formulario y le damos a Enviar. Mostrará lo siguiente:



Efectivamente, un Array ¿Y cómo he dicho que se accede a las cosas en un Array? Pues con el nombre **\$_POST** seguido entre corchetes del índice del que queremos obtener el valor. Por ejemplo **\$_POST['age']** tendrá el valor **"23"** (importante ver que es **"23"** pero de tipo **String**, es decir, caracteres) forzamos la conversión si deseamos obtener el tipo **Int**.

```
$edad = $_POST['age'];  
$edad = (int) $edad;  
var_dump ($edad);
```

Hasta aquí todo más o menos "entendible", pero

¿Qué pasa si estoy recibiendo un formulario con el **method="get"**? Pues que **\$_POST** no te sirve, tienes que seguir el mismo procedimiento, pero usando **\$_GET** fácil.

¿Qué pasa si recibo un formulario y no sé qué **method** usa? Pues tenemos la **joya de la corona** que es el Array **\$_REQUEST**

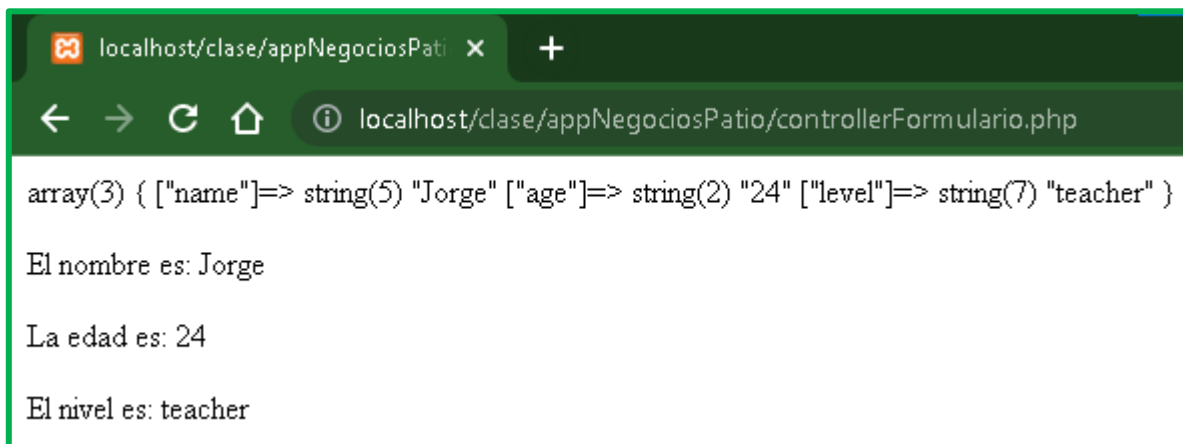
Con **\$_REQUEST** podemos acceder a los datos, se envíen éstos por la URL o en el cuerpo de la petición, es lo más recomendado de usar para acceder a los datos.

Escribimos entonces lo siguiente en **controllerFormulario.php**:

```
<?php
var_dump($_REQUEST);

$nombre = $_REQUEST['name'];
$edad = $_REQUEST['age'];
$nivel = $_REQUEST['level'];

print 'El nombre es: '.$nombre;
print 'La edad es: '.$edad;
print 'El nivel es: '.$nivel;
```



Como ves, el **var_dump()** muestra arriba el Array global **\$_REQUEST[]** y debajo se imprimen los **print** que usamos con las variables.

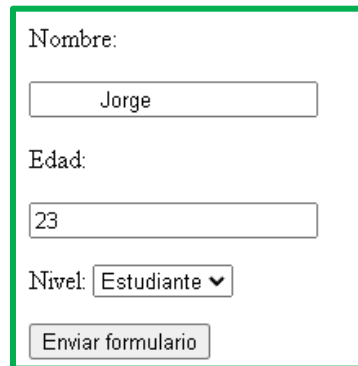
9.3 Validación de datos de formularios

Ahora, repasando todo lo aprendido hasta aquí, vamos a usar Arrays y funciones para validar los campos del formulario.

Es importante saber que la validación de datos va a depender de las condiciones del cliente, el ejercicio o proyecto que estemos realizando o cualquier tipo de restricción que se nos pida como desarrollador.

Así que simplemente usaré este apartado para enseñarte algunas funciones de PHP que se usan en una validación básica y la **eliminación de caracteres maliciosos**.

- **trim()**: Elimina los espacios en blanco al principio y final de una cadena de texto:

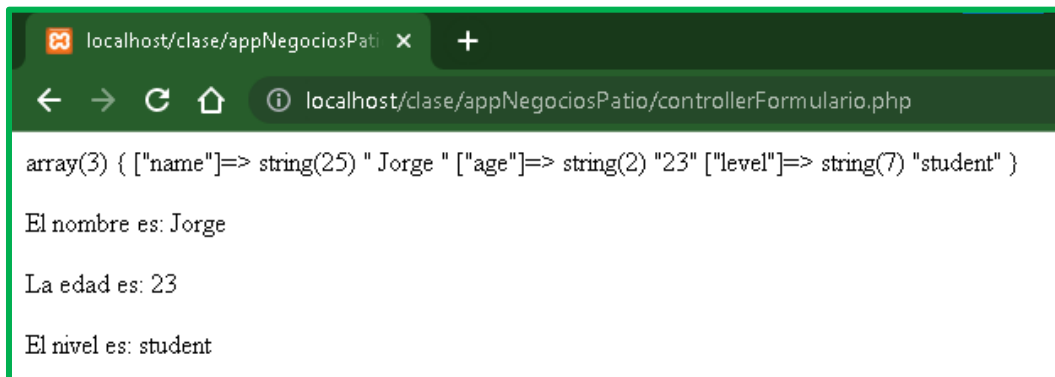


Nombre:

Edad:

Nivel:

Vamos a ver qué sucede si metemos 10 espacios en blanco antes y después del nombre:



```
array(3) { ["name"]=> string(25) " Jorge " ["age"]=> string(2) "23" ["level"]=> string(7) "student" }
```

El nombre es: Jorge

La edad es: 23

El nivel es: student

Cómo puedes ver, indica que el String que hay en `$_REQUEST["name"]`, tiene una longitud de `string(25)` "Jorge". Es porque está recogiendo 20 espacios en blanco + 5 caracteres de Jorge.

Con la función **trim()** eliminamos estos espacios:

```
$nombre = trim($_REQUEST['name']);  
var_dump($nombre);  
  
print '<p>El nombre es: '.$nombre.'</p>';
```

Nombre:

Edad:

Nivel:

Repetimos el proceso, pero esta vez, usando **trim()** en nuestro código:

```
string(5) "Jorge"  
  
El nombre es: Jorge  
  
La edad es: 50  
  
El nivel es: student
```

Cómo vemos, elimina los espacios y solamente deja los caracteres. Podemos usar **trim()** para que elimine otros caracteres que le indiquemos enviándolo como segundo parámetro:

```
$nombre = trim($_REQUEST['name'], 'Jge');  
var_dump($nombre);
```

```
string(2) "or"  
  
El nombre es: or  
  
La edad es: 60  
  
El nivel es: student
```

- **strip_tags()**: Elimina las etiquetas HTML y PHP de una cadena de texto:

Tomando el ejemplo anterior, vamos a **inyectar** código PHP en el campo nombre del formulario:

Nombre:

Edad:

Nivel:

string(38) "

Estoy saltandome la seguridad

"

El nombre es:

Para ello usamos una etiqueta <h1> y un texto largo, podemos ver que nuestro formulario es vulnerable, usemos **strip_tags()**:

```
$nombre = trim($_REQUEST['name'], 'Jge');  
var_dump($nombre);  
$nombre = strip_tags($nombre);  
var_dump($nombre);
```

```
string(38) "
```

Estoy saltandome la seguridad

```
" string(29) "Estoy saltandome la seguridad"
```

El nombre es: Estoy saltandome la seguridad

La edad es: 78

El nivel es: student

Observamos que el 1º `var_dump()` antes del **strip_tags** muestra el <h1> pero el 2º `var_dump()` muestra ya un texto normal, habiendo eliminado las etiquetas <h1>.

10. Trabajo con archivos

10.1 Apertura y cierre de archivos

Ahora que “comprendemos” mejor lo que es una función, vamos a usar `fopen()` para abrir un archivo:

```
<?php
$nuestroArchivo = fopen("archivo.txt", "r");
?>
```

La función recibe dos parametros, el primero el fichero y el segundo el modo de apertura (r: lectura; w: escritura; a: agregar datos al final del archivo; ...)

Al tratarse de una función, en este caso, nos devolverá “true” o “false” dependiendo del resultado de la operación. Por ello es importante comprobar:

```
if ($nuestroArchivo) {
    // Tratamos el fichero
} else {
}
?>
```

Es casi más importante aún cerrar el fichero después de usarlo para liberar recursos del sistema, usamos **`fclose()`** y le enviamos el identificador de archivo devuelto por **`fopen()`**.

```
<?php
$nuestroArchivo = fopen("archivo.txt", "r");

if ($nuestroArchivo) {
    // Tratamos el fichero

    // Cerramos el fichero
    fclose($nuestroArchivo);
} else {
    echo "No se pudo abrir el archivo";
}
?>
```

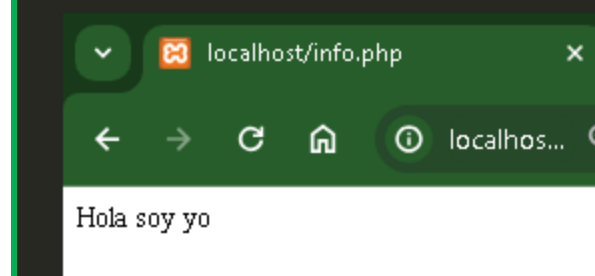
10.2 Lectura y escritura de archivos

Aquí empieza lo “complicadinho” y es que no tenemos una función mágica para escanear un archivo y devolver su contenido completo. Pero tenemos **fgets()**

Fgets() es una función de PHP que lee 1UNA(una) línea de texto desde un archivo.

```
// Tratamos el fichero
$linea = fgets($nuestroArchivo);
echo $linea;
```

```
1  Hola soy yo
2  fichero de ejemplo
3  Jorge Moya
```



Como vemos, no podríamos mostrar el fichero completo MAGICAMENTE. Pero si que podemos usar un bucle para leer la línea y mostrarla:

```
while (
    ($linea = fgets($nuestroArchivo)) !== false
) {
    echo $linea;
}
```

De esta forma, en cuanto **fgets()** devuelva false, parará de escribir líneas.

```
Hola soy yo, en un fichero de ejemplo, Jorge Moya
```

Ponemos las comas en el fichero de texto a mano.

A la hora de escribir en el archivo, recordad que debemos usar el 2º parámetro "w"

```
<?php
$nuestroArchivo = fopen("archivo.txt", "w");
```

Imaginemos que tenemos un Array de 10 ruletas en diferentes barrios y queremos pasarlas a un fichero:

```
$ruletas = array(
    "Calle Azar 777, Ciudad de la Suerte, Estado Ganador",
    "Avenida Apostador Dorado 123, Villa de la Fortuna, Condado de Riesgo",
    "Calle Ruleta 36, Barrio de la Esperanza, Ciudad del Juego",
    "Avenida Ganancia Instantánea 888, Pueblo del Azar, Provincia de Fortuna",
    "Calle Dados 555, Colina de los Apostadores, Municipio de la Suerte",
    "Avenida Jackpot 999, Barrio del Éxito, Ciudad de la Fortuna",
    "Calle Apuesta Alta 777, Pueblo de la Emoción, Condado del Riesgo",
    "Avenida Ruleta Rápida 101, Villa del Triunfo, Estado de la Suerte",
    "Calle Ganador 123, Ciudad del Apostador, Región de la Fortuna",
    "Avenida Gran Premio 666, Barrio de la Ganancia, Municipio del Juego"
);
```

Bueno, lo primero que se nos presenta es usar al compa **foreach** para recorrer el array e iterar (actuar) sobre cada elemento de éste.

```
if ($nuestroArchivo) {
    // Tratamos el fichero
    foreach ($ruletas as $direccionRuleta) {

    }
}
```

Y después escribimos usando **fwrite()**

```
foreach ($ruletas as $direccionRuleta) {
    fwrite($nuestroArchivo, $direccionRuleta . "\n");
}
```

- **fwrite()**: Función de PHP usada para escribir en un archivo. Toma dos parámetros: el identificador del archivo abierto y los datos que se van a escribir en el archivo.
- **\$nuestroArchivo**: Identificador del archivo en el que se va a escribir. **Debes haber abierto previamente el archivo utilizando la función fopen() y asignado su identificador a esta variable.**
- **\$direccionRuleta**: Esta es una variable que contiene la dirección que se desea escribir en el archivo.
- **"\n"**: Esto es un carácter de nueva línea. Cuando se escribe en el archivo, añade una nueva línea después de la dirección, lo que hace que la próxima escritura comience en una nueva línea en el archivo.

10.3 Manipulación de archivos

Ahora vamos a aprender a copiar, renombrar y eliminar archivos en PHP con **copy()**, **rename()** y **unlink()**.

A verificar la existencia de archivos con **file_exists()** y obtener información sobre archivos con **filesize()** y **filemtime()**.

11. Introducción a la programación orientada a objetos

11.1 ¿Qué es la POO?

11.2 Creación de clases y objetos

11.3 Métodos y propiedades de clases y objetos