PHP

Sitio: Aula Virtual do IES de Teis
Curso: Desarrollo web en entorno servidor 2023-24 (DAW-DUAL-A)
Libro: PHP

Impreso por: Abel Mahón Cortés
Data: Luns, 15 de Xaneiro de 2024, 13:28

Táboa de contidos

1. PHP

- Lenguaje de script ejecutado en el servidor
- Se suele encontrar incrustado en el código HTML
- Los bloques de código PHP se encuentran entre las etiquetas <?php ... ?>. En el caso de que la directiva *short_open_tag* true|false esté activa, se puede usar <? ... ?>, pero no se recomienda. Se puede encontrar en el fichero de configuración php.ini de la instalación de PHP (en nuestro caso: C:\xampp\php)
- El servidor web sabe distinguir cuándo le piden un archivo con extensión htm/html de un php.
- Toda la información que necesitemos saber sobre versión y configuración de PHP la obtendremos llamando a la función phpinfo();
- Requiere que las instrucciones terminen en punto y coma al final de cada sentencia. La etiqueta de cierre de un bloque de código de PHP ?> automáticamente implica un punto y coma. No es necesario usar un punto y coma para cerrar la última línea de un bloque de PHP. https://www.php.net/manual/es/language.basic-syntax.instruction-separation.php
- Para conocer la versión actual de PHP podemos utilizar la función phpversion()
- Documentación en línea en diferentes idiomas: https://www.php.net/docs.php

1.1. Variables

- Comienzan con un carácter $ seguido de una letra o un guion bajo. Son case sensitive (importan mayúsculas y minúsculas). No es lo mismo $variable que $Variable. Consultar https://www.php.net/manual/es/language.variables.basics.php
- Los tipos se asignan dinámicamente en función del valor asignado. Una misma variable podría tener varios tipos de datos distintos a lo largo de su vida
- Para saber si una variable está definida se utiliza **isset($var1[,$var2[,$var3...]])**, que devolverá un valor *true* **si la variable está definida y no es NULL.** Se le puede pasar un n.º variable de parámetros, y los evaluará ordenadamente finalizando si encuentra alguno no definido.

1.2. Tipos de datos

Para conocer los diferentes tipos de datos vamos a visitar la documentación oficial de PHP:

https://www.php.net/manual/es/language.types.php

Además PHP que nos será de ayuda para concer información sobre una variable:

- gettype — Obtener el tipo de una variable
- is_bool — Comprueba si una variable es de tipo booleano
- is_float — Comprueba si el tipo de una variable es float
- is_double — Alias de is_float
- is_int — Comprueba si el tipo de una variable es integer
- is_integer — Alias de is_int
- is_iterable — Verifica que el contenido de una variable es un valor iterable
- is_long — Alias de is_int
- is_null — Comprueba si una variable es null
- is_numeric — Comprueba si una variable es un número o un string numérico
- var_dump — Muestra información sobre una variable

1.3. Constantes en PHP

- Las constantes son un tipo especial de dato que se considera constante en la ejecución del programa. Un típico ejemplo es el valor Pi= 3,141592...
- Las constantes se crean con const NOMBRE_CONSTANTE = valor; (se recomienda con mayúsculas separadas por guiones bajos, aunque también es posible utilizando **define('nombreConstante', valor);** y posteriormente se utiliza directamente el nombre de la constante, que no lleva $ en ningún momento.)

1.4. Generación de HTML

- Para generar HTML desde PHP habitualmente se utiliza **echo** o **print**, a las que le pasamos la cadena que queremos generar en HTML resultado de la interpretación del código PHP. La diferencia entre ambas es que print devuelve 1, para ser utilizada en expresiones.
- printf. Permite pasar una cadena con los caracteres de formato adecuados antecedidos por un %, y tantos parámetros a continuación como caracteres de formato hayamos indicado. Los parámetros de formato pueden ser b(binario), c(carácter del nº indicado), d(entero), e(notación científica), f(nº real con punto decimal), o(octal), s(cadena), x(hexadecimal en minúsculas), X(hexadecimal en mayúsculas), etc.

1.5. Cadenas de caracteres

Las cadenas en PHP se pueden especificar de varias maneras distintas. Aquí veremos por el momento 2 de ellas:
● entre comillas simples: Todos los caracteres especificados encerrados entre comillas simples se evaluarán como dichos caracteres, excepto \' para una comilla simple y \\ para el propio carácter de backslash.
● entre comillas dobles: En las cadenas encerradas entre comillas dobles se evaluarán los caracteres de escape habituales (\n, \t, \\, \", etc.). Pero lo más importante es que las variables se expanden con su valor

```
$nombre = "Juan";
echo "<p>Me llamo $nombre </p>";
```

Esto generará una respuesta del servidor:
Me llamo Juan

Concatenación de cadenas de texto

A través del operador . (punto):
https://www.php.net/manual/es/language.operators.string.php

1.6. Operadores

Especialmente útiles al comienzo nos serán los operadores:

- aritméticos
- comparación
- incremento/decremento
- lógicos

Los operadores y sus precedencias pueden consultarse aquí: https://www.php.net/manual/es/language.operators.precedence.php

1.7. Estructuras de control

El teorema del programa estructurado indica que toda función que pueda ser realizada por un computador puede ser implementada en un lenguaje de programación que combine solo tres estructuras de control:

1. **Secuencial**: ejecución de una instrucción tras otra.
2. **Selección**l: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
3. **Iteración**: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

Veamos las siguientes secciones de Estructuras de control en PHP :

- Introducción
- if
- else
- elseif/else if
- while
- do-while
- for
- switch
- Sintaxis alternativa de estructuras de control
- break
- continue

1.8. Arrays

Un array es una estructura de datos que permite almacenar más de un dato.

Puede ayudar a entender qué es un array si lo concebimos como una especie de tabla donde en cada celda se almacena un valor. Para obtener el valor en una posición, hay que indicar en qué celda se encuentra.

clave	0	1	2
valor	DAM	DAW	ASIR

Para identificar la celda, se usa una *clave* que puede ser de dos tipos en PHP:

- Un **número** para obtener el dato de una posición. El primer dato está en la posición 0, el segundo dato en la posición 1 y así sucesivamente. Se dice que el array indexado numéricamente y la clave para obtener "DAM" es el 0, para obtener "DAW" es el 1, etc.

```
//Sintaxis larga array(...)
$ciclos = array(
    0 => "DAM",
    1 => "DAW",
    2 => "ASIR"
);
```

```
Array
(
    [0] => DAM
    [1] => DAW
    [2] => ASIR
)
```

- Una **cadena de texto** es la clave para obtener el valor. Se dice que son **arrays asociativos.**

```
//Sintaxis larga array(...)
$colores = array(
    "white"=> "0xFFFFFF",
    "black" => "0x000000",
    "red" => "0xFF0000"
);
```

```
Array
(
    [white] => 0xFFFFFF
    [black] => 0x000000
    [red] => 0xFF0000
)
```

Los valores almacenados pueden ser de cualquier tipo: enteros, bool, float, cadenas, objetos, etc.

La correspondencia entre clave valor hace que a menudo se les conozca como estructuras clave-valor.

Un mismo array puede tener índices numéricos y cadenas de texto como claves **:**

```
//Arrays con índices de dos tipos:

$mixto = ["white" => "0xFFFFFF"];
$mixto[] = "blanco";

print "<pre>";
print_r($mixto);
print "</pre>";
```

```
Array
(
    [white] => 0xFFFFFF
    [0] => blanco
)
```

Estos ejemplos los podemos encontrar en arrays.php

Para ver cómo trabajar con Arrays en PHP veremos este recurso:

https://www.php.net/manual/es/language.types.array.php

Las funciones relacionadas con arrays se pueden consultar aquí: https://www.php.net/manual/es/ref.array.php

Algunas destacables son:

- unset() — Para quitar una pareja clave/valor, se debe llamar a la función unset($ciclos[1]). Eliminará el elemento del array $ciclos. Para eliminar al variable $ciclos completamente, se puede usar unset($ciclos).
- is_array — Comprueba si una variable es un array
- sizeof — Alias de count (devuelve el número de elementos de un array)
- array_push — Inserta uno o más elementos al final de un array
- array_key_exists — Verifica si el índice o clave dada existe en el array
- array_keys — Devuelve todas las claves de un array o un subconjunto de claves de un array
- array_values — Devuelve todos los valores de un array
- array_merge — Combina dos o más arrays
- array_diff — Calcula la diferencia entre arrays
- explode — Divide un string en varios string
- implode — Une elementos de un array en un string
- print_r — Imprime información legible para humanos sobre una variable

Cómo recorrer un array
foreach

1.9. Formularios HTML

Recursos:

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form

Principales atributos de <form>

Para realizar el envío de un formulario, <form> deben conocerse al menos los siguientes atributos:

- **action**: La URL de la página o del recurso que procesará **en el servidor** la información enviada. Es un atributo opcional: Si está presente, será la URL de un fichero en el servidor. Si no lo está, se supondrá que será el mismo script donde se encuentra la etiqueta <form> en el lado del servidor.

- **method**: Indica el método del protocolo HTTP que se usará: Normalmente se usan:
  - ***GET***: Es un método de HTTP que permite recuperar un recurso o información del servidor. Los datos enviados aparecen en la URL del navegador, por lo que no se debe usar para enviar información sensible. Es el valor por defecto.
  - ***POST***: Es un método de HTTP que permite enviar información al servidor y que causa un cambio en el estado del servidor. Por ejemplo, crear un nuevo usuario o enviar un fichero. Los datos son enviados en el cuerpo de la petición HTTP en lugar de en la cabecera de la petición HTTP y no son visibles en la URL.

- **enctype**: Si el atributo method es **POST**, indica el tipo de contenido que se va a enviar a través de Media types o tipos de medios. Los Media Types son una cadena de texto que identifica el tipo de contenido enviado y ayuda a los navegadores a presentar dicho contenido. Por ejemplo, **su valor debe ser** `multipart/form-data` si el formulario tiene un <input type="file"> (un control para enviar el contenido de un fichero). Por defecto, su valor es la cadena de texto `application/x-www-form-urlencoded.`

- **name**: En caso de utilizar más de un formulario en un mismo documento HTML, debe ser único en el documento HTML y no puede ser una cadena vacía. Identificará al formulario **en el lado del servidor.**

1.10. Elementos de un formulario

Elementos <input>

Permiten al usuario introducir información. Existen multitud de tipos de elementos <input>y estos se especifican con el atributo ***type***. Si el atributo **type** no está presente, su valor por defecto es ***text***. Se trata de un elemento vacío o ***void element*** y no debe tener etiqueta de cierre.

Algunos tipos de elementos <input> son:

<input type="text">: Es el valor por defecto para el atributo *type*. Muestra una caja de texto que permite introducir texto. Cuenta con atributos para establecer la longitud máxima y mínima permitidas, un patrón con expresiones regulares para limitar las posibles entradas, el ancho de la caja de texto, si es de solo lectura, un placeholder o una pista del texto esperado, etc.

<input type="submit">: Muestra un botón que, una vez que el usuario haga clic sobre el mismo, enviará el contenido del formulario al servidor. Si existe un único elemento <button> dentro de un form, este será tratado como <input type="submit">.

Los diferentes tipos de <input> y sus atributos se pueden consultar aquí.

Atributos de todos los elementos <input>:

- **name**: El nombre **con el que se recibirá el dato en el servidor.** Debe ser **único** en el formulario y **debe estar presente para poder identificarlo en el servidor**. En PHP, en el caso de que se permitan enviar **múltiples valores**, como en una lista desplegable de selección múltiple o donde se permitan adjuntar múltiples ficheros, el **nombre deberá terminar con corchetes** para que ser reciba un array en el servidor:
  ```
  <input type="file" name="ficheros[]" id="fichero" multiple>
  ```

1.11. Arrays superglobales para obtener datos de formularios

Son arrays que están disponibles siempre en cualquier script PHP:

- *$_GET: Un array asociativo de variables pasado al script actual vía parámetros URL*

```
<h1>Formulario con GET</h1>
    <form method="get"  name="formulario-get">
        <label for="nombre">Introduzca su nombre: </label>
        <input type="text" name="nombre" id="nombre" required="required">
        <label for="email">Introduzca su email: </label>
        <input type="email" name="email" id="email" required="required">
        <input type="submit" value="Inicie sesión">
    </form>
```

```
En la URL aparecerá: http://localhost:3000/UD2/arrays/form.php?nombre=maria&email=maria%40c.com
```

En PHP, se podrá acceder a cada valor introducido por el usuario con:

```
$_GET["nombre"] y  $_GET["email"]
```

- *$_POST: Un array asociativo de variables pasadas al script actual a través del método POST de HTTP*

```
<h1>Formulario con POST</h1>
    <form method="post" name="formulario-post">
        <label for="nombre-post">Introduzca su nombre: </label>
        <input type="text" name="nombre-post" id="nombre-post" required="required">
        <label for="email">Introduzca su email: </label>
        <input type="email" name="email-post" id="email-post" required="required">
        <input type="submit" value="Inicie sesión">
    </form>
```

En PHP, se podrá acceder a cada valor introducido por el usuario con

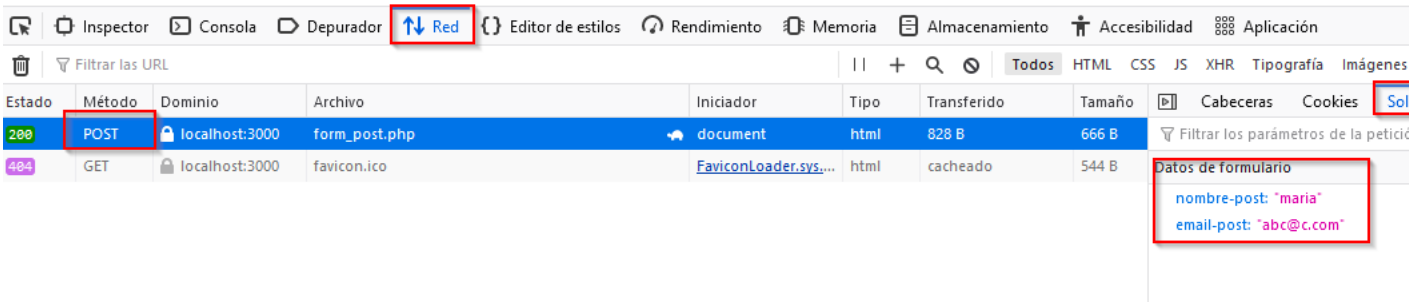
```
$_POST["nombre"] y  $_POST["email"]
```

```
En la URL aparecerá: http://localhost:3000/UD2/arrays/form.php
```

Se podrán visualizar los datos en el navegador con F12 o Herramientas del desarrollador en el propio navegador. En la imagen inferior se aprecia dónde se puede obtener el cuerpo de la petición con Google Chrome:

Formulario con POST

Introduzca su nombre: Introduzca su email: Inicie sesión



Si añadimos el código que procesará sendos formularios al html:

```
<h2> Estos son los datos recibidos de tu formulario:</h2>
<?php

//cambiar $_POST por $_GET en función del método HTTP utilizado
foreach ($_POST as $clave => $valor) {

    echo "<strong>$clave</strong>:";

    if (!is_array($valor)) {

        echo " $valor";
    } else {

        echo var_dump($valor);
    }

    echo "<br/>";
}
?>
```

1.12. Variables superglobales

Para ver las variables superglobales, recurrimos a la URL:

https://www.php.net/manual/es/language.variables.superglobals.php

Son accesibles desde cualquier lugar de cualquier script

Son arrays asociativos y sus claves serán, dependiendo del caso, cadenas predefinidas o nombres de otros elementos.

- $GLOBALS: Hace referencia a las variables disponibles en el ámbito global. Las entradas de este array son los nombres de las variables globales.
- $_SERVER: Información del entorno del servidor y de ejecución. Las entradas de este array podrían variar dependiendo del servidor.
- *$_GET: Sus claves se corresponden con los atributos **name** de los elementos del formulario enviados con el método GET (por defecto). Se accede a su valor a través de la clave: `$_GET["nombre"]`*
- *$_POST: Sus claves se corresponden con los atributos **name** de los elementos del formulario enviados con el método POST. Se accede a su valor a través de la clave: `$_POST["nombre"]`*
- $_REQUEST: Este array se nutre de los valores de otros arrays superglobales (generalmente $_GET,$_POST y $_COOKIE). Aunque pueda parecer cómodo utilizarlo, puede dar lugar a vulnerabilidades. Se recomienda acceder directamente al array destinado a cada uso. Realmente, la actualización automática del array $_REQUEST depende de la directiva request_order del php.ini (que veremos a continuación), a la que se le asigna una combinación de las iniciales EGPCS (de los arrays superglobales ENV, GET, POST, COOKIE y SERVER) para establecer qué datos de los arrays se copian y en qué orden. Un posible valor podría ser request_order="GP" con lo que se copiarían los datos de GET y los de POST.
- *$_COOKIE: Este array almacena las cookies como pares clave-valor. Estas cookies permiten transmitir datos que envía un servidor web al cliente web, y que este almacena, para enviarle de nuevo al servidor web en futuras conexiones. Se verá más en detalle en la UD4 (del centro educativo)*
- *$_SESSION: Este array almacena permite almacenar datos como pares clave-valor en el contexto de interacción de un cliente web con el servidor. Se verá también en la UD4 (del centro educativo).*
- $_ENV: Permite acceder a las variables de entorno. Hay que asegurarse de que la directiva variables_order permite tener acceso.
- $_FILES: Su clave es el atributo name del elemento <input type="file"...>. Su valor es, a su vez, un array asociativo con las claves name, type, size, tmp_name y error y un valor simple para cada clave. Si se envían múltiples ficheros, el *array asociativo con las claves name, type, size, tmp_name y error* tiene como valor un array para cada fichero enviado

La estructura del array que se crea cuando se adjuntan varios ficheros se puede consultar en la URL https://www.php.net/manual/es/features.file-upload.post-method.php
Es importante asegurarse de que el archivo se ha subido correctamente is_uploaded_file() y moverlo del directorio de almacenamiento temporal a una ubicación definitiva con *move_uploaded_file()*

1.13. Envío de ficheros

Siempre que se envíen ficheros en un formulario hay que añadir a la etiqueta <form>:

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

Al elemento <input type="file":

- Un atributo , por ejemplo: name="ficheros"
- Si se permite más adjuntar más de un fichero el atributo ***multiple*** *y, e*n PHP, añadimos unos corchetes al name: name="ficheros[]"

```
<input type="file" name="ficheros[]" id="ficheros" multiple >
```

1.14. Operador ternario "?:"

Permite resumir un if else sencillo con una sintaxis básica. Podemos ver un ejemplo en la documentación oficial:

https://www.php.net/manual/es/language.operators.comparison.php#language.operators.comparison.ternary

El **operador ternario** merece un comentario por tener una implementación diferente a cualquier otro lenguaje de programación conocido, pues **es asociativo a la izquierda**:

```
echo true ? 0 : true ? 77 : 99;
// Visuliza 99 (true ? 0 : true) ? 77 : 99
echo true ? 0 : (true ? 77 : 99);
// Aquí sí visualiza 0, posiblemente lo que esperábamos
echo true ? 1 : true ? 77 : 99;
// Visuliza 77 (true ? 1 : true) ? 77 : 99
```

Si en el operador ternario se deja en blanco el segundo operando, se convierte en un operador binario denominado **Elvis operator**:

```
$resultado = $dato ?: 'valor si se evalúa como falso';
```

1.15. Funciones definidas por el usuario

Una introducción a la creación de funciones definidas por el usuario se puede encontrar en la documentación oficial:

https://www.php.net/manual/es/functions.user-defined.php

- Se usa la palabra clave function seguida del nombre de la función.

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Función de ejemplo.\n";
    return $valor_devuelto;
}
?>
```

- El nombre de las funciones no es sensible a mayúsculas y minúsculas, pero es una buena práctica llamar a la función exactamente igual que en su definición. Un nombre de función válido comienza con una letra o guion bajo, seguido de cualquier número de letras, números o guiones bajos. Los nombres de función usan guion bajo entre palabras. es_correcto(). No se recomienda que empiecen por __ (guion guion) porque se podría confundir con elementos mágicos del lenguaje PHP. Normalmente no empiezan por guion.

- Tras el nombre de la función, se abren paréntesis y se indican los parámetros de entrada de la función. A continuación se cierran paréntesis y se abren llaves para indicar el cuerpo de la función.
- Se pueden proporcionar valores por defecto a los últimos parámetros, asignándole el valor en su declaración. https://www.php.net/manual/es/functions.arguments.php#functions.arguments.default. Si existen, deben ir **siempre al final.**
- PHP no admite la sobrecarga de funciones (funciones con mismo nombre de función, pero con diferentes tipos y / o número de parámetros. Ejemplos en Java se pueden encontrar aquí.)
- Para devolver el valor de la función se usa return, finalizando su ejecución inmediatamente. Si no se usa return, se entiende que devuelve NULL implícitamente.
- Dentro de la función puede ir cualquier código válido, incluso otras funciones y clases.
- Todas las funciones y clases de PHP tienen ámbito global. Se pueden llamar desde fuera de una función incluso si fueron definidas dentro, y viceversa.
- A partir de PHP 7 ya se pueden especificar los tipos bool, float,int,string e iterable tanto en los tipos de los parámetros como en el tipo devuelto: https://www.php.net/manual/es/functions.arguments.php#functions.arguments.type-declaration

1.16. Include y require

include y require: Expresiones que permiten estructurar código en diferentes ficheros

include y require permiten añadir el contenido de un fichero en otro: La diferencia fundamental es como tratan la no existencia del archivo incluido. El include genera un warning, y el require genera un error y finaliza la ejecución del script.

Ámbito de los ficheros incluidos

Cuando se incluye un archivo, el código que contiene hereda el ámbito de las variables de la línea en la cual ocurre la inclusión. Cualquier variable disponible en esa línea del archivo que hace la inclusión estará disponible en el archivo incluido, desde ese punto en adelante.

Sin embargo, todas las funciones y clases definidas en el archivo incluido tienen el ámbito global, salvo que se haga el include o require dentro de una función, en cuyo caso, todo el código contenido en el archivo incluido se comportará como si hubiera sido definido dentro de esa función

Cualquier código al interior del archivo objetivo que deba ser ejecutado como código PHP, tendrá que ser encerrado dentro de etiquetas válidas de comienzo y terminación de PHP.

Encadenación de include

Si un fichero A incluye un fichero B y este, a su vez, un fichero C, las rutas relativas que aparezcan en B se interpretarán partiendo del directorio del fichero A.

Para resolverlo, hay que usar una de estas alternativas que hacen uso de constantes mágicas:

- `include dirname(__FILE__)."/../C.php";`
- `include __DIR__."/../C.php";`

Para que el código PHP sea correctamente interpretado tanto en Windows como en Linux, en lugar de usar "/" o "\" en las rutas de directorios, se recomienda usar la constante DIRECTORY_SEPARATOR que se sustituirá por "/" o "\" dependiendo del Sistema Operativo donde se interprete el código PHP. Esto hará que nuestro código sea interoperable. Las opciones anteriores deberían usarse:

- `include dirname(__FILE__).DIRECTORY_SEPARATOR."..". DIRECTORY_SEPARATOR."C.php";`
- `include __DIR__.DIRECTORY_SEPARATOR."..".DIRECTORY_SEPARATOR."C.php";`

Se puede ver un ejemplo de uso en los ficheros adjuntos ejemplos_include.rar
También existen include_once y require_once que son iguales a los anteriores, pero se aseguran de que el fichero solo se incluya una vez en el script.

1.17. Ámbito de las variables

- El ámbito de las variables es el contexto dentro del que la variable está definida
- Dentro de las funciones definidas por el usuario se usa un ámbito local a la función. Si una variable se usa por primera vez dentro de una función, estará definida solo dentro de esa función.
- Si dentro de una función se desea usar variables inicializadas o declaradas con anterioridad, deberá usarse la palabra clave **global**
- Podemos ver ejemplos de estas diferencias en la documentación https://www.php.net/manual/en/language.variables.scope.php
- Ni los bucles ni las ramas if-else.. definen un ámbito local propio. Las variables definidas en los bucles para su recorrido siguen existiendo al terminar su ejecución. Veamos un ejemplo:

```
<?php

const MAX_ITERATIONS = 10;

for ($i = 0; $i < MAX_ITERATIONS; $i++) {
   echo "<p>$i</p>";
}

echo "<p> \$i fuera del for es:  $i";



echo "<p> \$i fuera del while es:  $i";

$array = [2, 4, 6];
foreach ($array as $key => $value) {
   echo "<p> en foreach: \$key es $key \$value es: $value</p>";
}

echo "<p> \$key-\$value fuera del foreach es:  $key-$value</p>";

//Ojo con definir variables en una rama alternativa, pues puede que no se inicialice la variable en función de las condiciones.
if (2 > 1) {
   $a = "a";
} else {
   //No se garantiza que $b haya sido asignada un valor en una estructura conficional
   $b = "b";
}

var_dump($a);
var_dump($b);
if (is_null($b)) {
   echo "b is null";
}
```

1.18. Paso por valor y referencia

Las Referencias en PHP son medios de acceder al mismo contenido de una variable **mediante diferentes nombres.**

Asignación por referencia

Se permite hacer que dos variables hagan referencia al mismo contenido. Es decir, cuando se hace:

```
<?php
$a =& $b;
?>
```

significa que *$a* y *$b* **apuntan al mismo contenido.**

```
<?php

$a = 1;
$b =& $a;

echo  " después de crear un alias con asignación por referencia \$b = $b, \$a=$a <br>";

$a = 5;
echo " después de asignar valor a \a:   \$b = $b, \$a=$a <br>"  ;
unset($b);
echo  " después de hacer unset de \$b = $b, \$a=$a";
```

Paso de argumentos de funciones por referencia
Antes debemos conocer las diferencias entre parámetros y argumentos en una función:

- Los parámetros son los nombres que aparecen en la definición de una función.
- Los argumentos son los valores que le pasamos (y que, por tanto, recibe) una función.

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.

Ver ejemplo 2 de la documentación https://www.php.net/manual/es/functions.arguments.php

1.19. Foreach y paso por referencia

Al recorrer un array con un foreach, es posible modificar los elementos del array si se utiliza una referencia.

Tenéis un ejemplo en la documentación https://www.php.net/manual/es/control-structures.foreach.php

Para entender el script de la documentación del foreach, vamos a intentar representar el $array en memoria.

```
<?php
$array = array(1, 2, 3, 4);
foreach ($array as &$valor) {
    $valor = $valor * 2;
}
// $array ahora es array(2, 4, 6, 8)

// sin unset($valor), $valor aún es una referencia al último elemento: $array[3]

foreach ($array as $clave => $valor) {
    // $array[3] se actualizará con cada valor de $array...
    echo "{$clave} => {$valor} ";
    print_r($array);
}
// ...hasta que finalmente el penúltimo valor se copia al último valor

// salida:
// 0 => 2 Array ( [0] => 2, [1] => 4, [2] => 6, [3] => 2 )
// 1 => 4 Array ( [0] => 2, [1] => 4, [2] => 6, [3] => 4 )
// 2 => 6 Array ( [0] => 2, [1] => 4, [2] => 6, [3] => 6 )
// 3 => 6 Array ( [0] => 2, [1] => 4, [2] => 6, [3] => 6 )
?>
```

Los valores que se almacenan en el array se almacenarán en la memoria en posiciones consecutivas que tienen su propia numeración. Por ejemplo, numeradas en hexadecimal en la columna de la derecha:

Clave array	Valores	Direcciones de memoria
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
		0x000000A1
		0x000000A2
		0x000000A3
		...

Cuando se usa un foreach como en el ejemplo que sigue (sin &), se crea una nueva variable $valor en otra posición de memoria y en esa posición (0x000000A1 en el ejemplo) se van **copiando los valores del array uno en cada iteración** al inicio de cada iteración. En este ejemplo, al final de cada iteración, se actualiza la variable $valor con el doble de cada posición del $array:

```
foreach ($array as $valor) {
    $valor = $valor * 2;
}
```

Al comienzo de la 1ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	1	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 1ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	2	0x000000A1
		0x000000A2
		0x000000A3
		...

Al comienzo de la 2ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	2	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 2ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	4	0x000000A1
		0x000000A2
		0x000000A3
		...

Al comienzo de la 3ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	3	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 3ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	6	0x000000A1
		0x000000A2
		0x000000A3
		...

Al comienzo de la 4ª iteración:		
Valores	**Direcciones de memoria**	
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	4	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 4ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	8	0x000000A1
		0x000000A2
		0x000000A3
		...

Nombre variable

El ámbito de la variable $valor excede el foreach, es decir, $valor tendrá el último valor asignado durante el foreach, al salir del mismo.

Cuando se usa el paso **por referencia (con &),** no se crea una copia del valor de cada elemento del array, sino que se guarda la dirección de memoria que ocupa cada elemento y **$valor pasa a ser una referencia a la posición de memoria de cada elemento del array**:

```
foreach ($array as &$valor) {
    $valor = $valor * 2;
}
```

Al comienzo de la 1ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	**0x0000000A**
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000A**	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 1ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	**0x0000000A**
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000A**	0x000000A1
		0x000000A2
		0x000000A3
		...

Al comienzo de la 2ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	**0x0000000B**
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000B**	0x000000A1
		0x000000A2
		0x000000A3
		...

Al final de la 2ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	**0x0000000B**
$array[2]	3	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000B**	0x000000A1
		0x000000A2
		0x000000A3
		...

Al comienzo de la 3ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	**0x0000000C**
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000C**	0x000000A1
		0x000000A2

Al final de la 3ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	**0x0000000C**
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000C**	0x000000A1
		0x000000A2

		0x000000A3
		...

Al comienzo de la 4ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	1	0x0000000A
$array[1]	2	0x0000000B
$array[2]	3	0x0000000C
$array[3]	4	**0x0000000D**
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000D**	0x000000A1
		0x000000A2
		0x000000A3
		...

		0x000000A3
		...

Al final de la 4ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	0x0000000C
$array[3]	8	**0x0000000D**
		...
		...
		0x000000A0
$valor	**lo que haya en la posición 0x0000000D**	0x000000A1
		0x000000A2
		0x000000A3
		...

Como el ámbito de la variable $valor excede el foreach, al finalizar el bucle anterior, $valor apunta a la dirección donde se almacena $array[3]. Si variamos $array[3], es lo mismo que variar $valor y viceversa: Si variamos el valor de $valor es lo mismo que variar $array[3]. Son una especie de alias para la misma posición de memoria.

Si a continuación, iteramos con un foreach por el array con la variable $valor (sin usar &, es decir, **por valor y no por referencia**), $valor va a ir tomando **los valores almacenados en cada posición del array en cada iteración** variando, en consecuencia, $array[3], pues **ambas variables son distintos nombres pero hacen referencia al mismo contenido.**

```
foreach ($array as $clave => $valor) {
    // $array[3] se actualizará con cada valor de $array...
    echo "{$clave} => {$valor} ";
    print_r($array);
}
```

En la 1ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	0x0000000C
$array[3]	2	**0x0000000D**
		...
		...
		0x000000A0
$valor	**cambiará lo que haya en la posición 0x0000000D en función del valor que vaya tomando**	0x000000A1
		0x000000A2
		0x000000A3
		...

La asignación $valor = 2 equivale a "lo que haya en la posición 0x0000000D =2"

En la 2ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	0x0000000C
$array[3]	4	0x0000000D
		...
		...
		0x000000A0
$valor	**cambiará lo que haya en la posición 0x0000000D en función del valor que vaya tomando**	0x000000A1
		0x000000A2
		0x000000A3
		...

La asignación $valor = 4 equivale a "lo que haya en la posición 0x0000000D =4"

En la 3ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	0x0000000C
$array[3]	6	0x0000000D
		...
		...
		0x000000A0
$valor	**cambiará lo que haya en la posición 0x0000000D en función del valor que vaya tomando**	0x000000A1
		0x000000A2
		0x000000A3
		...

La asignación $valor = 6 equivale a "lo que haya en la posición 0x0000000D =6"

En la 4ª iteración:		
Nombre variable	**Valores**	**Direcciones de memoria**
$array[0]	2	0x0000000A
$array[1]	4	0x0000000B
$array[2]	6	0x0000000C
$array[3]	6	**0x0000000D**
		...
		...
		0x000000A0
$valor	**cambiará lo que haya en la posición 0x0000000D en función del valor que vaya tomando**	0x000000A1
		0x000000A2
		0x000000A3
		...

La asignación $valor = 6 equivale a "lo que haya en la posición 0x0000000D =6"

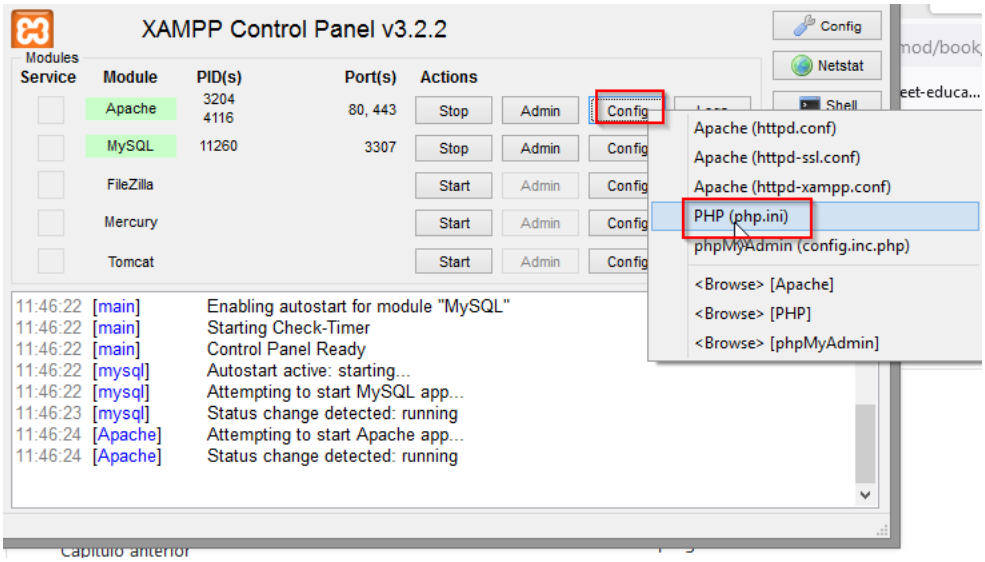
De ahí que cuando se use el paso por referencia se aconseje usar **unset($valor)** al terminar el foreach, para eliminar estos efectos indeseados.

1.20. Directivas de configuración

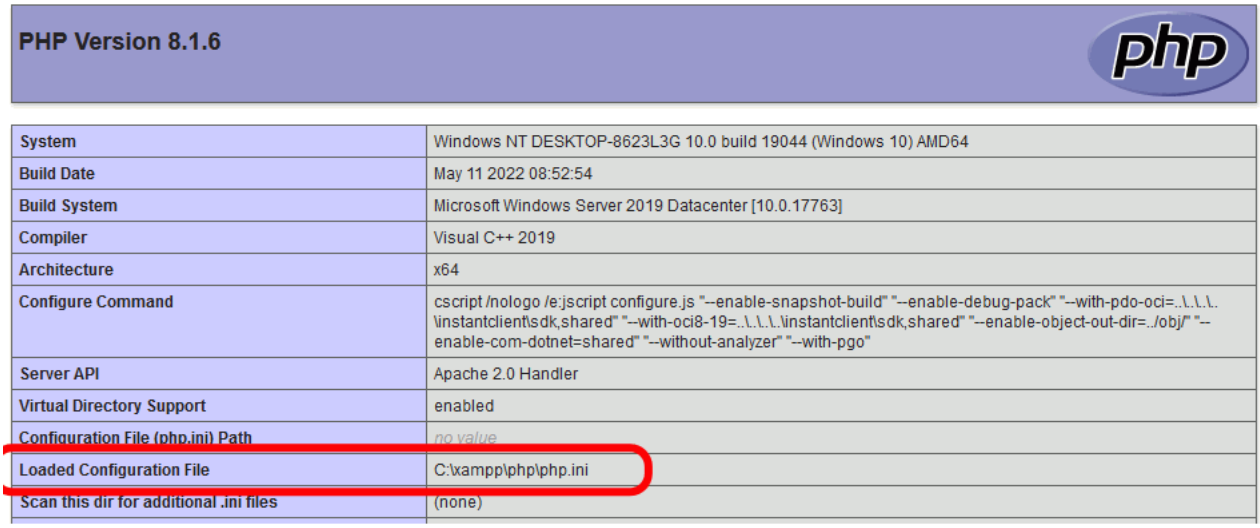
Directivas de configuración en el fichero php.ini

El archivo php.ini es el archivo de configuración de PHP.

Las directivas establecen la configuración del intérprete de PHP en la ejecución de un servidor.
En la infraestructura que instala XAMPP, el fichero php.ini se encuentra por defecto en C:\xampp\php\php.ini. También es accesible desde el Panel de control de XAMPP:



La ubicación del fichero php.ini puede variar. Se puede confirmar con la función phpinfo(); qué configuración se está cargando:



En php.ini podemos encontrar directivas de configuración que se aplican a diferentes aspectos del funcionamiento del intérprete de PHP.

Podemos cambiarlas editando directamente dicho archivo lo que afectará **a todos los scripts PHP del servidor**, o establecer su valor **durante la ejecución del script .php** utilizando la función **ini_set($directiva, $valor).** Aunque no todos los servidores lo permiten y no todas las directivas son accesibles desde esta función, puede ser útil cuando estamos en un servidor compartido donde no tengamos permisos para realizar cambios a los archivos de configuración. El nuevo valor se mantendrá durante la ejecución del script y se restaurará al finalizar.

El propio fichero de php.ini incluye documentación sobre su formato, cómo comentar líneas y los valores permitidos. Se recomienda una lectura del mismo por lo menos hasta la línea 60.

Entre las directivas, podemos destacar, por el momento:

1. short_open_tag
2. error_reporting
3. display_errors
4. default_mimetype
5. default_charset
6. upload_max_filesize
7. max_file_uploads

8. upload_tmp_dir
9. file_uploads
10. request_order

Si se modifica el fichero php.ini, tendremos que reiniciar el servidor web Apache para que se apliquen los cambios.

Veremos más directivas a medida que las vayamos utilizando. Un listado completo de las directivas se puede encontrar en la URL https://www.php.net/manual/es/ini.list.php

Para interpretar la columna ***Cambiable*** visita la URL:

https://www.php.net/manual/es/configuration.changes.modes.php

1.21. Comentarios en PHP

```
//Comentarios de una línea

# Comentario de una línea

/*

Comentarios de varias líneas

*/
```

Más información en https://www.php.net/manual/es/language.basic-syntax.comments.php

1.22. Manipulación de tipos

Para ver este apartado veremos el recurso online https://www.php.net/manual/es/language.types.type-juggling.php#language.types.typecasting

Se destacan algunos apartados:

Conversión a booleano

- Para especificar un literal de tipo boolean se emplean las constantes **`true`** o **`false`**. No son case-sensitive (podría usarse TrUE, FalSe, etc.)
- Para mostrar el valor de una variable booleana con valor False, echo no muestra nada. Podéis usar var_export o var_dump

Conversión a entero

- se puede emplear tanto `(int)` como `(integer)`
- También puede darse conversión automática en función del operador
- **`false`** producirá `0` (cero), y **`true`** producirá `1` (uno).
- Cuando se convierte un float a un integer, el número será redondeado *hacia cero*. Nunca se debe convertir una fracción desconocida a un integer, ya que a veces puede conducir a resultados inesperados. En su lugar, podéis utilizar la función `round()`

Conversión a número de punto flotante

Se rige de forma similar a las conversiones de string a int: la conversión es la misma que si el valor hubiese sido convertido primero a integer y luego a float.

Conversión a cadena

- `(string)` o mediante la función strval().
- La conversión es automática cuando se utilizan las funciones echo o print, o cuando se compara una variable con un string:
  - El valor **`true`** del tipo boolean es convertido al string `"1"`. El valor **`false`** del tipo boolean es convertido al string `""`
  - **`null`** siempre es convertido a un string vacío.
  - Los arrays siempre son convertidos al string `"Array"`. Debido a esto, echo y print no pueden por sí mismos mostrar el contenido de un array. print_r() y var_dump() pueden ser útiles para arrays

1.23. Funciones de cadenas

- strlen — Obtiene la longitud de un string

- str_contains — Determina si un string contiene
- str_starts_with — Determina si un string comienza con un substring dado
- str_ends_with — Determina si un string termina con un substring dado

- str_replace — Reemplaza todas las apariciones del string buscado con el string de reemplazo
- str_ireplace — Versión insensible a mayúsculas y minúsculas de str_replace

- strpos — Encuentra la posición de la primera ocurrencia de un substring en un string
- stripos — Encuentra la posición de la primera aparición de un substring en un string sin considerar mayúsculas ni minúsculas
- strrchr — Encuentra la última aparición de un caracter en un string

- strcmp — Comparación de string sensible a mayúsc. y minúsc.
- strcasecmp — Comparación de string segura a nivel binario e insensible a mayúsculas y minúsculas

- strtolower — Convierte un string a minúsculas
- strtoupper — Convierte un string a mayúsculas
- ucfirst — Convierte el primer caracter de una cadena a mayúsculas
- lcfirst — Pasa a minúscula el primer caracter de un string

- ltrim — Retira espacios en blanco (u otros caracteres) del inicio de un string
- trim() - Elimina espacio en blanco (u otro tipo de caracteres) del inicio y el final de la cadena
- rtrim() - Retira los espacios en blanco (u otros caracteres) del final de un string

- str_repeat — Repite un string

- strrev — Invierte una string

- strstr — Encuentra la primera aparición de un string
- strchr — Alias de strstr
- substr — Devuelve parte de una cadena

- implode — Une elementos de un array en un string
- join — Alias de implode

- str_split — Convierte un string en un array dependiendo de la longitud del fragmento
- explode — Devuelve un array de string. Divide un string en varios string dependiendo de un delimitador.

Una lista más extensa de las funciones de cadenas se puede consultar en la URL
https://www.php.net/manual/es/ref.strings.php