Apuntes Unidad 2

Sitio: Centros - Granada
Curso: Desarrollo web en entorno servidor
Libro: Apuntes Unidad 2

Imprimido por: Bailón Gracia, Ángel Javier
Día: jueves, 18 de noviembre de 2021, 19:39

Tabla de contenidos

1. ¿Qué es PHP?

PHP es un lenguaje de programación para desarrollo web en el lado del servidor. Es flexible y permite programar pequeños scripts con rapidez.

Se puede comparar con otros lenguajes de script que se ejecutan según el mismo principio: ASP (*Active Server Pages*), JSP (*Java Server Pages*) o PL/SQL Server Pages (PSP).

A diferencia de un lenguaje como JavaScript, donde el código se ejecuta del lado del cliente (en el navegador), el código PHP se ejecuta del lado del servidor. El resultado de esta ejecución se incrusta en la página HTML, que se envía al navegador. Este último no tiene conocimiento de la existencia del procesamiento que se ha llevado a cabo en el servidor.

Esta técnica permite realizar páginas web dinámicas cuyo contenido se puede generar total o parcialmente en el momento de la llamada de la página, gracias a la información que se recopila en un formulario o se extrae de una base de datos.

¿Qué puede hacer PHP?

2. Sintaxis de PHP

El elemento básico en PHP es el bloque. El bloque PHP está delimitado por las etiquetas correspondientes, y contiene una o varias sentencias separadas por punto y coma.

El código PHP se introduce dentro del HTML utilizando la etiqueta **<?php** para abrir el bloque de PHP y la etiqueta **?>** para cerrarlo.

```
<?php
    Sentencia1;
    SentenciaN;
?>
```

Además, cuando un fichero contiene solo PHP se recomienda no cerrar la etiqueta del último bloque. Puede dar problemas si la respuesta del servidor involucra a varios ficheros.

El servidor sustituye los bloques de PHP por su salida.

HTML y PHP

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hola mundo</title>
  </head>
  <body>
    <?php
     echo "Hola mundo";
    ?>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hola mundo</title>
  </head>
  <body>
    Hola mundo
  </body>
</html>
```

3. Comentarios

Se puede utilizar comentarios:

- De bloque, encerrados entre "/*" y "*/"
- De línea, comenzando por "//" o por "#"

```
<!DOCTYPE html>
<html>
<body>

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/

// This is a single-line comment

# This is also a single-line comment
?>

</body>
</html>
```

4. Variables y tipos de datos

Una de las características principales de PHP es que es un lenguaje no fuertemente tipado. Esto quiere decir que no es necesario indicar el tipo de dato al declarar una variable. De hecho, las variables no se declaran, se crean la primera vez que se les asigna un valor.

El tipo de dato depende del valor con que se inicialicen. Esto agiliza la escritura de programas, pero también tiene inconvenientes. Si no se presta atención, puede dar lugar a código de baja de calidad y, a medida que las aplicaciones crecen, pueden darse errores difíciles de depurar.

4.1. Declaración de variables

En PHP los identificadores de las variables van siempre precedidos por el carácter '$'. El identificador de la variable debe comenzar por una letra o un guion bajo ('_'), y puede estar formado por números, letras y guiones bajos. Para declarar una variable, solo hay que asignarle un valor:

```
$nombre = valor;
```

Por ejemplo, esta sentencia declara la variable $entero, que será de tipo integer porque se inicializa con un entero.

```
$entero = 4;
```

También es posible cambiar el tipo de dato de una variable simplemente asignándole un valor de otro tipo de dato, como se puede ver en el siguiente ejemplo (líneas 8-12). El ejemplo utiliza la función gettype(), que devuelve el tipo de dato de una variable.

```
1  <?php
2  /* declaración de variables */
3  $entero = 4; // tipo integer
4  $numero = 4.5; // tipo coma flotante
5  $cadena = "cadena"; // tipo cadena de caracteres
6  $bool = TRUE; //tipo booleano
7  /* cambio de tipo de una variable */
8  $a = 5; // entero
9  echo gettype($a); // imprime el tipo de dato de a
10 echo "<br>";
11 $a = "Hola"; // cambia a cadena
12 echo gettype($a); // se comprueba que ha cambiado
```

La salida del ejemplo confirma que la variable cambia de tipo de dato.

Si realizas una operación con variables de distintos tipos, ambas se convierten primero a un tipo común. Por ejemplo, si sumas un entero con un real, el entero se convierte a real antes de realizar la suma:

```
$mi_entero = 3;
$mi_real = 2.3;
$resultado = $mi_entero + $mi_real;
// La variable $resultado es de tipo real
```

Estas conversiones de tipo, que en el ejemplo anterior se lleva a cabo de forma automática, también se pueden realizar de forma forzada. Para hacerlo, escribiremos el nombre del tipo deseado entre paréntesis antes de al variable que se quiere forzar:

```
$mi_entero = 3;
$mi_real = 2.3;
$resultado = $mi_entero + (int) $mi_real;
// La variable $mi_real se convierte a entero (valor 2) antes de sumarse.
// La variable $resultado es de tipo entero (valor 5)
```

En la documentación de PHP se especifican las conversiones de tipo posibles y los resultados obtenidos con cada una:

Conversiones de tipos posibles y los resultados obtenidos

4.2. Asignación por copia y por referencia

En principio, la asignación de variables se realiza mediante copia. Es decir, si hacemos:

```
$a = $b;
```

se crea una nueva variable a y se le asigna el valor que tenga b. Las variables a y b representan posiciones diferentes de memoria, aunque tengan el mismo valor después de la asignación.

También es posible definir una referencia a una variable utilizando el operador ampersand:

```
$var2 = &$var1;
```

En este caso $var2 no es una nueva variable con el valor de $var1. Por el contrario, $var2 apunta a la misma dirección de memoria que $var1, de manera que $var1 y $var2 son en realidad dos nombres para el mismo dato.

4.3. Variables no inicializadas

Si se intenta utilizar una variable antes de asignarle un valor, se genera un error de tipo E_NOTICE, pero no se interrumpe la ejecución del script. Si una variable no inicializada aparece dentro de una expresión, dicha expresión se calcula tomando el valor por defecto para ese tipo de dato. En este ejemplo podemos ver lo que ocurre al utilizar una variable no inicializada dentro de una expresión.

```
1  <?php
2  $var1 = 100;
3  $var3 = 100 + $var2;//$var2 no existe, se toma como 0
4  echo "$var3 <br>"; // muestra 100
5  $var3 = 100 * $var2; // $var2 no existe, se toma como 0
6  echo "$var3 <br>"; // muestra 0
```

En la línea 3, se intenta sumar una variable no inicializada. Como el valor por defecto para un entero es 0, el resultado de la suma es 100. En la línea 5, se intenta multiplicar por una variable no inicializada y, al multiplicar por 0, el resultado es 0.

La salida muestra un mensaje de error por cada intento de utilización de una variable no inicializada.

4.4. Tipos de datos

Tipos de datos escalares:

- booleanos (boolean). Sus posibles valores son true y false. Además, cualquier número entero se considera como true, salvo el 0 que es false. Este tipo de dato se obtiene, entre otros casos, como resultado de los operadores de comparación y se utiliza en sentencias condicionales y bucles.

Cuando se espera un valor boolean y se recibe otro tipo de dato, se aplican reglas de conversión.

Conversión implícita a *boolean*

Tipo	Valor como *boolean*
integer	Si es 0 se toma FALSE, en otro caso como TRUE
float	Si es 0.0 se toma FALSE, en otro caso como TRUE
string	Si es una cadena vacía o "0", se toma como FALSE, en otro caso como TRUE
variables no inicializadas	FALSE
null	FALSE
array	Si no tiene elementos se toma FALSE, en otro caso como TRUE

- **entero** (integer). Cualquier número sin decimales.

  La conversión entre integer y float es automática. Aunque también se pueden usar los operadores de conversión (int) y (float).

- **cadena** (string). Conjuntos de caracteres delimitados por comillas simples o dobles. Para delimitar una cadena se puede utilizar las comillas simples o dobles.

- **real** (float). Cualquier número real. Se pueden representar también en notación científica. Los redondeos puedan dar sorpresas.

Si se usan las dobles (*comillas mágicas*), las variables que aparezcan dentro de la cadena se sustituirán por su valor. Las comillas dobles son muy prácticas, ya que es más rápido insertar directamente las variables que montar las cadenas con varias concatenaciones.

TOMA NOTA

A la hora de formatear la salida hay que tener en cuenta que esta va a ser procesada como HTML por un navegador web, es decir, los saltos de línea se ignoran y los espacios en blanco consecutivos colapsan. Los caracteres de escape como '\n', '\r' y '\t' son ignorados por el navegador. Para introducir un salto de línea la opción más sencilla es incluir la etiqueta de salto de línea de HTML ("
") en la salida, como se puede ver en los ejemplos anteriores.

Otros tipos de datos:

- **array**. Para representar colecciones de elementos.
- **object**, PHP tiene soporte completo para la programación orientada a objetos.
- **callable**. Un tipo de dato especial para representar funciones de callback, funciones que se pasan a otras funciones.
- **null**. El tipo de dato null representa una variable que no ha sido asignada. Solo puede tomar un único valor, NULL. Se considera que una variable es de tipo null si se le asigna el valor NULL o no está inicializada.
- **resource**. Este tipo de dato representa recursos externos, como una conexión a una base de datos

4.5. Ámbito de las variables

El ámbito de una variable es la parte del código en que esta visible. Una variable declarada en un fichero PHP está disponible en ese fichero y en los ficheros que se incluyan desde este.

Por otro lado, las funciones definen un ámbito local, de manera que las variables que se declaran en las misma solo son accesibles desde la propia función. Además, desde la función no se puede acceder a otras variables que no sean las locales o sus argumentos.

Para definir variables globales hay dos opciones. La palabra reservada **global** y la variable predefinida **$_GLOBALS**. Las variables globales son accesibles desde cualquier función o fichero de la aplicación.

Aquí puedes ver un ejemplo que ilustra su uso.

Para saber más sobre los ámbitos de las variables consulta la documentación de PHP.
http://www.php.net/manual/es/language.variables.scope.php

4.6. Variables predefinidas

Contienen información sobre el servidor, datos enviados por el cliente o variables de entorno.
Dentro de las variables predefinidas hay un grupo de ellas, las **superglobales**, que están disponibles en cualquier ámbito.

Las variables superglobales son relevantes para el desarrollo de aplicaciones web y las iremos conociendo a lo largo del curso.

Variables *superglobales*

Nombre	Descripción
$GLOBALS	Variables globales definidas en la aplicación
$_SERVER	Información sobre el servidor
$_GET	Parámetros enviados con el método GET (en la URL)
$_POST	Parámetros enviados con el método POST (formularios)
$_FILES	Ficheros subidos al servidor
$_COOKIE	*Cookies* enviadas por el cliente
$_SESSION	Información de sesión
$_REQUEST	Contiene la información de $_GET, $_POST y $_COOKIE
$_ENV	Variables de entorno

Para saber más acerca de ella consulta el manual de PHP

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

5. Constantes

Para definir constantes se utiliza la función define(), que recibe el nombre de la constante y el valor que queremos darle.

define(“LIMITE”,1000);

Es habitual utilizar identificadores en mayúsculas para las constantes. Los nombres de las constantes deben empezar por una letra o guión bajo y tener sólo letras, números y guiones bajos.

Las **constantes predefinidas** o mágicas, en realidad ,no son constantes puesto que cambian dependiendo de donde se emplean.

6. Operadores

Estos son los operadores más usados:

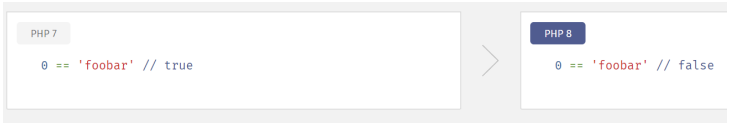
Operador	
Operadores de comparación	
e1 === e2	Idéntico. Verdadero si las dos expresiones son del mismo tipo y tienen el mismo valor.
e1 == e2	Igual. Verdadero si las dos expresiones son iguales tras la conversión de tipos, si es necesaria.
e1 !== e2	No idéntico.
e1 != e2, e1 <> e2	No igual.
e1 >= e2, e1 > e2, e1 <= e2, e1 < e2	Mayor o igual, mayor, menor o igual, menor.
e1 ?? e2 ?? e3	Comenzando por la izquierda, devuelve la primera expresión no nula.
Operadores aritméticos	
+e1	Como operador unario, sirve para convertir la expresión a *integer* o *float*, según corresponda.
–e1	Como operador unario, cambio de signo.
e1 + e2, e1 - e2, e1 * e2, e1 / e2	Suma, resta, multiplicación, división.
e1 % e2	Módulo.
e1 ** e2	Potencia.
Operadores lógicos	
e1 and e2, e1 && e2	Y. Verdadero si las dos expresiones se evalúan a TRUE.
e1 or e2, e1 || e2	O. Verdadero si una o las dos expresiones se evalúan a TRUE.
e1 xor e2	O exclusivo. Verdadero si solo una de las expresiones se evalúa a TRUE.
!e1	Devuelve TRUE si la expresión es FALSE y viceversa.
Operadores a nivel de bit	
e1 & e2, e1 | e2, e1 ^ e2, ~e1	Y, O, O exclusivo y negación.
$var» N	Desplaza N bits de $var hacia la izquierda.
$var «N	Desplaza N bits de $var hacia la derecha.
Operadores de asignación	
$var1 = e1	Asignación por valor.
$var2 = &$var2	Asignación por referencia.
$var += e1, $var -= e1, $var *= e1, $var /= e1	Equivalentes a $var = $var + e1, $var = $var – e1… Válido para cualquier operador binario aritmético, de cadenas o *arrays*.
Otros operadores	
$var++, $var--	Devuelve $var, luego le suma (resta) 1.
++$var, --$var	Suma (resta) 1 a $var, devuelve el valor actualizado.
$cad1. $cad2	Concatena dos cadenas.

Para saber más de los operadores consultaremos el manual.

7. Otros operadores

Comparadores inteligentes.

En PHP 8 hay comparaciones inteligentes entre strings y números.



En versiones anteriores a la 8 la evaluación de igualdad devuelve true aunque la cadena no contenga ningún carácter numérico.

Esto sucede porque las versiones antiguas del algoritmo interpretan la ausencia de caracteres numéricos en un string como equivalente a 0, devolviendo este valor en la comparación.
Este y otros fallos se corrigen con PHP8. Para saber más haz clic aquí

Operador nave espacial (<=>)

Este operador devolverá -1, 0 o 1 si la primera expresión es menor, igual o mayor que la segunda expresión. Es un operador binario que realiza tres comparaciones:

Ejemplo:

```
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
```

Es muy útil cuando se escriben funciones de comparación definidas por el usuario usando usort, uasort o uksort .

Antes de que apareciera este operador se requerían expresiones más elaboradas con estructuras de control para conseguir lo mismo.

Los objetos no se pueden comparar.

Operador ternario

El operador ternario es una forma abreviada de la sentencia if-else, por lo que volveremos a hablar de él cuando veamos las estructuras condicionales.

Operador null coalesce (fusión de null ??)

El operador de fusión de null (??) se ha añadido como aliciente sintáctico para el caso común de la necesidad de utilizar un operador ternario junto con isset(). Devuelve su primer operando si existe y no es null; de lo contrario devuelve su segundo operando. Volveremos a verlo más adelante cuando se explique isset()

Ejemplo:

```
echo $variable ?? 'No existe';
```

Si la variable existe y no es nula se muestra su valor. En caso contrario se mostraría el mensaje 'No existe'.

Operador nullsafe

Para comprobar en PHP 7 que una propiedad de un objeto o un método no devuelve null, necesitamos escribir varias sentencias if para comprobar si esa propiedad o método nos da null o no. Ahora con el operador nullsafe, si la primera propiedad, o el primer método, me devuelve null se anula la ejecución de toda la cadena.

Ejemplo en PHP clásico:

```php
$universidad =  null;

if ($session !== null) {

    $usuario = $session->usuario;

    if ($usuario !== null) {
        $estudios = $usuario->getEstudios();

        if ($estudios !== null) {
            $universidad = $estudios->universidad;
        }
    }
}
```

Ejemplo en PHP 8:

```php
$universidad = $session?->usuario?->getEstudios()?->universidad;
```

8. Estructuras de control

PHP cuenta con las estructuras de control habituales en la programación estructurada para realizar sentencias condicionales y de repetición.

8.1. Estructuras condicionales

- **if**
- **if-else**
- **if-elseif**

Operador ternario

Como ya vimos es la forma abreviada de la sentencia if- else.

Usarlo nos ayuda a crear código más limpio y fácil de entender.

Se llama ternario porque consta de 3 partes, la primera es la condición, la segunda el valor que nos devuelve si la condición es verdadera y el tercero es el valor que devuelve si la condición es false. Ambos valores para falso y verdadero se separan entre ellos con un signo `:` mientras en el signo `?` se usa para separar la condición de los posibles valores: falso y verdadero.

Ejemplo:

```php
$a = 2;
$b = 1;

$resultado = $a > $b ? true : false;

echo($resultado); //true
```

Este código equivale a escribir:

```php
$a = 2;
$b = 1;
 if ($a > $b) {
  $resultado = true;
} else {
  $resultado = false;
}

echo($resultado); // true
```

- **switch**
- **match**

8.2. Estructuras de repetición

- for
- while
- do-while

Dentro de un bucle se pueden usar las sentencias break y continue.

La primera sirve para salir del bucle o switch en el que aparece.

La sentencia continue fuerza una nueva iteración del bucle. Las sentencias que estén dentro del bucle, pero después de continue no se ejecutan y, si se cumple la condición del bucle, se ejecuta una nueva iteración. Si se trata de un bucle for, se ejecutan las instrucciones de autoincremento antes de evaluar la condición.

8.3. Otros estructuras de control

Podemos organizar el código en diferentes scripts PHP según nos convenga y reutilizar ese código en otros scripts.

Es conveniente por ejemplo para declarar funciones, crear la cabecera de una página, almacenar datos de configuración, para reutilizar código, etc.

Para incluir otros ficheros utilizaremos las sentencias include y require.

include “mifichero.php”

require “mifichero.php”

Se diferencian solo en el tratamiento de errores. Si no se encuentra el fichero especificado *include* genera un error del tipo E_NOTICE, pero *require* genera un error E_FATAL, lo que implica el fin de la ejecución del script. Si no hay errores en ambos casos el fichero especificado se ejecuta. Si el fichero es una librería con funciones, estas se declaran y quedan disponibles para el fichero que lo incluye. Si el fichero contine sentencias fuera de una función, estas se ejecutan.

También están disponibles las sentencias require_once e include_once, análogas a las anteriores, pero con la diferencia de que solo se ejecutan los ficheros especificados si no han sido incluido o requeridos con anterioridad.

La inclusión de ficheros tiene una diferencia con otros lenguajes. Supongamos que el fichero A incluye al fichero B. Las rutas relativas que aparezcan en B se interpretarán a partir del directorio de A.

Para solucionarlo, en el fichero B se utiliza dir_name(__FILE__) que devuelve la ruta del fichero:

```
include( dirname(__FILE__).“\”.’fichero.php’);
```

A la hora de usarlos require y require_once son más estrictos y generalmente más recomendables (dependerá de la situación).

Para acabar con las estructuras de control, la sentencia return finaliza la ejecución del fichero cuando se encuentra fuera de una función.

Si el script se estaba ejecutando mediante un require o include, la ejecución continúa el el fichero que lo incluyó. Si se pasa como argumento un número entero, este se devuelve como valor de retorno.

Si se encuentra dentro de una fusión, finaliza la ejecución de esta y la función devuelve el argumento del return. La ejecución continua en el fichero o función que llama a la función.

8.4. Sintaxis alternativa de las estructuras de control

Esta sintaxis alternativa de las estructuras de control se suele utilizar cuando estamos trabajando en las vistas de un programa de PHP. Si estamos poniendo código incrustado es recomendable usar este tipo de sintaxis para hacerlo más visual.

9. Variables superglobales $_GET y $_POST

Paso de parámetros

A menudo necesitaremos que se pueda establecer una comunicación entre el cliente y el servidor, probablemente interaccionando por medio de formularios cuyos campos tendrán que ser procesados para dar una respuesta adecuada.

El protocolo HTTP define varios métodos para ello. Los más habituales son GET y POST.

El método GET es el que se utiliza para solicitar página web a un servidor, por ejemplo, al seguir el vínculo o introducir una dirección a mano en la barra del navegador.

El método POST se utiliza sobre todo para enviar formularios al servidor.

Cuando se usa el método GET se pueden pasar parámetros al servidor en la URL.

Para ello, a la ruta normal para acceder a una página se le añade el carácter "?" como indicador de que empieza la lista de parámetros. Cada parámetro tiene su nombre, a la izquierda del igual, y un valor, a la derecha. Los argumentos están separados entre sí por el carácter ampersand. Por ejemplo, se puede acceder con el navegador a:

http://localhost/ejercicios-clase/saludo_personalizado.php?nombre=Ana$apellido=Prieto

Para acceder a los argumentos de la URL desde un bloque PHP se usa el array superglobal $_GET, que tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del array.

Para controlar si los parámetros se han pasado correctamente se pueden utilizar las funciones empty() o is_null(). Las dos devuelven TRUE cuando el parámetro no está presente en la URL. La diferencia está en los parámetros presentes, pero sin valor.

Para que los parámetros no aparezcan en la URL se usa el método POST.

En el script al que se envía el formulario los parámetros están disponibles en el array superglobal $_POST. La clave de cada argumento dentro del array es el atributo name del elemento correspondiente en el formulario.

10. Funciones

Una función es un conjunto de instrucciones que realiza una tarea concreta.

Las funciones pueden recibir argumentos, los datos que necesitan para llevar a cabo su contenido.

Las funciones pueden devolver o no un valor. En PHP no es necesario declarar el tipo de dato que devuelve una función, y, de hecho, puede devolver tipos de datos diferentes según el caso. Por ejemplo, muchas funciones devuelven FALSE en caso de error y no devuelven un *boolean* si no hay error.

Al desarrollar aplicaciones complejas, a menudo, hay instrucciones que se repiten. Podemos mejorar el código usando funciones como solución y así reutilizar el código.

10.1. Funciones definidas por el usuario

Su sintaxis:

```
function nombre(argumentos){
    <instrucciones>
}
```

La función puede tener o no argumentos que se declaran en la cabecera separados por comas.

10.2. Funciones predefinidas

Las más utilizadas están relacionadas con las variables. Como en PHP la variables no se declaran explícitamente hay una serie de funciones que permiten conocer el estado y el tipo de datos de una variable.

Entre las más importantes tenemos:

- is_null($var)
- isset($var)
- unset($var)
- empty($var)
- is_int($var)
- print_r($var)
- var_dump($var)

También tenemos funciones de cadenas como:

- strlen($cad)
- explod($cad,$token)
- implode($token,$array)
- strcmp($cad1,$cad2)
- strtolower($cad)
- strtoupper($cad)
- str($cad1,$cad2)

Funciones de arrays:

- ksort($arr),krsort($arr)
- sort($arr),rsort($arr)
- array_values($arr)
- array_keys($arr)
- array_key_exits($arr,$cla)
- count($arr)

10.3. Paso de argumentos por copia y por valor

En PHP los argumentos se pasan por copia.

Esto quiere decir que, cuando se llama a una función con una variable como argumento, se crea una variable local a la función en la que se copia el valor del argumento. Si la función modifica el argumentos, estos cambios no tienen efecto en la variable original.

10.4. Funciones como argumentos

En PHP se pueden pasar funciones como argumentos a otras funciones.

Se usa por ejemplo para las funciones de callback.

Sólo hay que pasar el nombre de la función ente comillas como argumento.

La función que lo recibe podrá utilizarla si le pasan los argumentos adecuados.

Funciones de retorno o callback

Cuando asignamos **una función como parámetro a otra función**, hablaremos de un *callback*. Las funciones *callback* suelen utilizarse mucho en bibliotecas y *frameworks*.

Lo normal es que se ejecuten únicamente después de que tenga lugar otro evento o circunstancia. Es decir, solo se invoca si ha tenido lugar otra acción claramente definida.

Un ejemplo de función *callback* son los controladores de eventos que se utilizan en elementos HTML como los botones. El evento podría ser un clic del ratón que hace que se ejecute el callback, y la función en sí misma podría provocar el redireccionamiento a otra página o transmitir un valor que se haya introducido en un formulario.

La principal diferencia entre una función normal y un *callback* es que una función normal se ejecuta directamente, mientras que una función ***callback* solo se define** y se llama y ejecuta únicamente cuando ocurre un evento concreto.

En PHP, los *callbacks* son un método muy utilizado para que las **funciones se comuniquen entre sí**. Suelen emplearse para implementar ***plugins* o módulos** de manera limpia y para garantizar su funcionalidad.

Este sería un ejemplo de una función clásica de *callback* en PHP:

```
function my_callback_function() {

        echo '¡Hola, mundo!';

}
```

Al llamar esta función, se genera el *string* "¡Hola, mundo!".

En PHP, las funciones *callback* también pueden ser **métodos de un objeto**, incluidos los métodos de clase estática.

Este sería el ejemplo de *callback* **como método:**

```
class MyClass {

        static function myCallbackMethod() {

                echo '¡Hola, mundo!';

        }

}
```

Ejemplo de uso de callback

10.5. Funciones anónimas

Las **funciones anónimas**, o closures, permiten crear funciones que no tienen un nombre específico.

De esta forma, podremos declarar una función sin nombre y almacenarla en una variable.

La sintaxis para definir un 'closure' es :

```
function(){
  //implementacion
}
```

En este caso para llamar a la función anónima utilizaremos la sintaxis de funciones variables, es decir nombre de la variable seguida de los paréntesis de apertura y cierre.

Ejemplo:

```php
$saludo = function() {
    return "Hola que tal";
};

echo $saludo(); // Devuelve: Hola que tal
```

Se utilizan, por ejemplo, cuando la función que definimos tiene un uso extremadamente puntual y no va a ser reutilizada. Por eso, la definimos en el mismo lugar donde se usa y de esta forma el código quedará más claro.

Es muy común usar las funciones anónimas en combinación con los callbacks,

Ejemplo: Usaremos una función anónima como callback para multiplicar los elementos de la misma posición de cada uno de los arrays.

```php
$multiplicador = function($a, $b) {
  return $a * $b;
};
$numeros = range(1, 10);
$numeros2 = range(1, 10);
$lista = array_map($multiplicador, $numeros,$numeros2);
echo implode(" ", $lista);
```

En este ejemplo hemos llamado a la función anónima desde otra función como argumento.

Las funciones anónimas son muy útiles en la POO.

Si quieres ver más ejemplos de funciones anónimas pincha aquí.

10.6. Funciones de flecha

Las funciones de flecha (short closures) son una forma más corta de escribir las funciones anónimas que ayuda a que el código sea más legible.

Su forma básica es : `fn (parámetros) => exp`

Veamos un ejemplo en que tenemos el mismo código representado mediante una función anónima y una función de flecha.

En este ejemplo se muestra los múltipos de 2,3 y 5 que hay en el array $numbers.

Usando una función anónima:

```php
<?php

$numbers = [12, 18, 5, 11, 10, 95, 3];

$factors = [2, 3, 5];

foreach($factors as $factor) {
    $multiples = array_filter($numbers, function($n) use ($factor) {
        return $n%$factor == 0;
    });
    echo "Multiples of $factor\n";
    print_r($multiples);
}
```

Usando la función de fecha:

```php
<?php

$numbers = [12, 18, 5, 11, 10, 95, 3];

$factors = [2, 3, 5];

foreach($factors as $factor) {
    $multiples = array_filter($numbers, fn($n) => $n%$factor == 0);
    echo "Multiples of $factor\n";
    print_r($multiples);
}
```

Debemos de tener en cuenta que:

- No se pueden usar las llaves {}.
- La expresión no lleva ';' al final.
- No se usa *return*.
- No necesitas el «use» para poder acceder a variables externas (como sí sucede en las funciones anónimas).

11. Arrays

Los arrays en PHP son una estructura muy flexible y potente. Unifica en un solo tipo lo que en otros lenguajes se consigue con arrays básicos, vectores, lista o diccionarios.

Los elementos de un array se identifican por una clave, que puede ser un entero o una cadena. Por tanto, las claves pueden ser numéricas o asociativas

Los elementos guardan un orden dentro del array. Este orden está determinado por el orden de los elementos al declarar el array o al añadir nuevos.

Para declarar un array se puede utilizar la función **array()**

```
$var = array (
clave1 => valor1,
…..
claven => valorn,
);
```

También es posible la notación entre corchetes

```
$arr = array [
clave1 => valor1,
…..
claven => valorn,
];
```

Para acceder a cada elemento del array lo hacemos utilizando su clave entre corchetes.

```
$arr[clave] = valor;
```

En PHP no es necesario que indiques el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array. Simplemente puedes comenzar a asignarle valores.

Para recorrer un array lo habitual es utilizar el bucle foreach.

```
foreach($arr as $valor){
....
}
```

El cuerpo del bucle se ejecuta una vez por cada elemento del array $arr. En cada iteración del bucle $valor almacena elemento correspondiente.

También se puede utilizar especificando una variable para la clave,

```
foreach($arr as $clave =>$valor){…}
```

En este caso, en cada iteración también se dispondrá de la clave del elemento correspondiente en la variable $clave.

Ejemplo:

```
<?php
    $capitalesamerica = array("Chile"=>"Santiago","Peru"=>"Lima","Ecuador"=>"Quito");
    foreach($capitalesamerica as $capi=>$valores){
           echo "La Capital del Pais de ".$capi." es: ".$valores;
           echo "<br />";
    }
?>
```

11.1. Arrays multidimensionales

En PHP puedes crear arrays de varias dimensiones.

Esto se consigue incluyendo un array entero como parte de otro, y así sucesivamente.

Ejemplo: Supongamos que queremos crear una matriz bidimensional con las calificaciones obtenidas por una alumna en las tres evaluaciones de Matemáticas, Lengua y Dibujo.

Podríamos hacerlo así:

```
$notas["Mat"]["Pri"] = "Suficiente";
$notas["Mat"]["Seg"] = "Bien";
$notas["Mat"]["Ter"] = "Notable";
$notas["Len"]["Pri"] = "Sobresaliente";
$notas["Len"]["Seg"] = "Sobresaliente";
$notas["Len"]["Ter"] = "Notable";
$notas["Dib"]["Pri"] = "Notable";
$notas["Dib"]["Seg"] = "Bien";
$notas["Dib"]["Ter"] = "Suficiente";
```

O bien, así:

```
$notas = array("Mat"=>array("Pri"=>"Suficiente",
                            "Seg"=>"Bien",
                            "Ter"=>"Notable"),
               "Len"=>array("Pri"=>"Sobresaliente",
                            "Seg"=>" Sobresaliente",
                            "Ter"=>"Notable"),
               "Dib"=>array("Pri"=>"Notable",
                            "Seg"=>"Bien",
                            "Ter"=>"Suficiente"));
```

Por supuesto también podíamos haber usado índices numéricos para las asignaturas y las calificaciones:

```
$notas = array(0=>array(0=>5,1=>6,2=>7),
               1=>array(0=>9,1=>9,2=>7),
               2=>array(0=>7,1=>6,2=>5));
```

Aunque de esta forma la declaración es más breve, sin embargo, en el futuro habrá que recordar a qué asignatura y a que evaluación corresponde cada índice.

Para hacer referencia a los elementos almacenados en un array multidimensional, debes indicar las claves para cada una de las dimensiones.

11.2. Funciones de arrays

Consultaremos las funciones de arrays en el manual de PHP.

11.3. Ejemplos arrays

Descarga aquí los ejemplos

12. Excepciones y errores

El sistema de control de errores de PHP ha ido evolucionando a lo largo de las versiones. En el sistema básico, se generan errores de diferentes tipos, representados por un número. A partir de PHP 5 se introdujo un sistema de excepciones usando los bloques try/catch/finally. Y finalmente, en PHP 7, aparecieron las excepciones de clase Error.

Hay que tener en cuenta que, aunque podemos controlar los errores, es imposible recuperarse de cierto tipo de errores y con ello se detiene la ejecución del programa. Sin embargo, podemos recuperarnos de las excepciones. Utilizaremos las excepciones para cambiar el flujo normal de un script si ocurre un error concreto dentro de una condición (llamada excepción).

Excepciones

Para indicar que se ha producido un fallo o error en nuestro código se puede lanzar una **excepción**.

Una excepción puede ser lanzada ("thrown") y opcionalmente capturada ("catched").

Para controlar las excepciones se utilizan bloque **try/catch/finally.**

- El código susceptible de producir algún error se introduce en un bloque **try**.
  - Cuando se produce algún error, se lanza una excepción utilizando la instrucción **throw.**
  - Después del bloque try debe haber como mínimo un bloque catch encargado de procesar el error.
  - Si una vez acabado el bloque try no se ha lanzado ninguna excepción, se continúa con la ejecución en la línea siguiente al bloque o bloques **catch**.
  - Se puede añadir un bloque **finally** que se ejecuta después del try/catch, haya habido excepción o no.

```
try{
instrucciones;
} catch(Exception e){
instrucciones;
}finally {
instrucciones;
}
```

Por ejemplo, para lanzar una excepción cuando se produce una división por cero podrías hacer:

```
try {
       if( $divisor == 0)
           throw new Exception("División por cero.");
    }
catch (Exception $e) {
        echo 'Se ha producido el siguiente error: ' . $e->getMessage();
    }
```

Una vez que una excepción es atrapada en el bloque catch, el objeto *Exception* contiene el mensaje de error que fue lanzado usando la palabra clave *throw*. La variable $e del ejemplo anterior es una instancia de la clase *Exception,* por lo que tiene acceso a todos los métodos de esa clase. En este bloque debes definir tu propia lógica para el manejo de excepciones, es decir, qué es lo que quieres hacer exactamente con el error que atrapaste.

PHP ofrece una clase base Exception para utilizar como manejador de excepciones. Para lanzar una excepción no es necesario indicar ningún parámetro, aunque de forma opcional se puede pasar un mensaje de error (como en el ejemplo anterior) y también un código de error.

Entre los métodos que puedes usar con los objetos de la clase Exception están:

- getMessage. Devuelve el mensaje, en caso de que se haya puesto alguno.
- getCode. Devuelve el código de error si existe.

Hay una forma de centralizar el manejo de todas las excepciones no capturadas de forma que puedas controlar su salida desde un sólo sitio. Para ello se utiliza set_exception_handler(), que establece una función de gestión de excepciones definida por el usuario.

Errores

En el sistema básico, ante determinadas condiciones (por ejemplo, utilizar una variable no inicializada) PHP genera un error. Hay diferentes tipos de errores, cada uno asociado con un número y una constante predefinida. Se puede controlar cómo se comporta PHO ante los errores mediante tres directivas del fichero php.ini:

- error_reporting: indica qué errores deben reportarse. Lo normal es utilizar E_ALL
- display_errors: señala si los mensajes de error deben aparecer en la salida del script (adecuado para desarrollo, pero no en producción)
- log_errors: indica si los mensajes de error deben almacenarse en un fichero (útil en producción)
- error_log: si la directiva anterior está activada, es la ruta en la que se guardan los mensajes de error

El valor de la directiva error_reporting es un número, pero para especificarlo lo habitual es utilizar las constantes predefinidas y el operador or a nivel de bit.

Errores y Registro

Valor	Constante	Descripción	Nota
1	**E_ERROR** (integer)	Errores Fatales en tiempo de ejecución. Éstos indican errores que no se pueden recuperar, tales como un problema de asignación de memoria. La ejecución del script se interrumpe.	
2	**E_WARNING** (integer)	Advertencias en tiempo de ejecución (errores no fatales). La ejecución del script no se interrumpe.	
4	**E_PARSE** (integer)	Errores de análisis en tiempo de compilación. Los errores de análisis deberían ser generados únicamente por el analizador.	
8	**E_NOTICE** (integer)	Avisos en tiempo de ejecución. Indican que el script encontró algo que podría señalar un error, pero que también podría ocurrir en el curso normal al ejecutar un script.	
16	**E_CORE_ERROR** (integer)	Errores fatales que ocurren durante el arranque incial de PHP. Son como un **E_ERROR**, excepto que son generados por el núcleo de PHP.	
32	**E_CORE_WARNING** (integer)	Advertencias (errores no fatales) que ocurren durante el arranque inicial de PHP. Son como un **E_WARNING**, excepto que son generados por el núcleo de PHP.	
64	**E_COMPILE_ERROR** (integer)	Errores fatales en tiempo de compilación. Son como un **E_ERROR**, excepto que son generados por Motor de Script Zend.	
128	**E_COMPILE_WARNING** (integer)	Advertencias en tiempo de compilación (errores no fatales). Son como un **E_WARNING**, excepto que son generados por Motor de Script Zend.	
256	**E_USER_ERROR** (integer)	Mensaje de error generado por el usuario. Es como un **E_ERROR**, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error().	
512	**E_USER_WARNING** (integer)	Mensaje de advertencia generado por el usuario. Es como un **E_WARNING**, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error().	
1024	**E_USER_NOTICE** (integer)	Mensaje de aviso generado por el usuario. Es como un **E_NOTICE**, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error().	
2048	**E_STRICT** (integer)	Habilítelo para que PHP sugiera cambios en su código, lo que asegurará la mejor interoperabilidad y compatibilidad con versiones posteriores de PHP de su código.	Desde PHP 5 pero no incluidoen **E_ALL** hasta PHP 5.4.0
4096	**E_RECOVERABLE_ERROR** (integer)	Error fatal capturable. Indica que ocurrió un error probablemente peligroso, pero no dejó al Motor en un estado inestable. Si no se captura el error mediante un gestor definido por el usuario (vea también set_error_handler()), la aplicación se abortará como si fuera un **E_ERROR**.	Desde PHP 5.2.0
8192	**E_DEPRECATED** (integer)	Avisos en tiempo de ejecución. Habilítelo para recibir avisos sobre código que no funcionará en futuras versiones.	Desde PHP 5.3.0
16384	**E_USER_DEPRECATED** (integer)	Mensajes de advertencia generados por el usuario. Son como un **E_DEPRECATED**, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger_error().	Desde PHP 5.3.0
32767	**E_ALL** (integer)	Todos los errores y advertencias soportados, excepto del nivel **E_STRICT** antes de PHP 5.4.0.	32767 en PHP 5.4.x, 30719 en PHP 5.3.x, 6143 en PHP 5.2.x, 2047 anteriormente

La función error_reporting() permite cambiar el valor de la directiva error_reporting en tiempo de ejecución. De esta forma no hay que acceder al archivo php.ini

También es posible definir una función propia (personalizada) para que se encargue de los errores utilizando set_error_handler().

Excepciones error

Las excepciones de tipo Error no heredan de la claseException, así que para capturarlas hay que usar:

catch(Error $e){

...}

o, alternativamente , la clase Throwable, de la que se derivan tanto Error como Exception:

catch(Throwable $e){

...}

Aquí puede encontrar las excepciones Error predefinidas.

12.1. Ejemplos excepciones

Descarga **aquí** los ejemplos

13. Mapa conceptual

Mapas conceptuales del curso de Desarrollo Web Entorno Servidor DAW José Luis Comesaña

Tema 2 - Características del lenguaje PHP