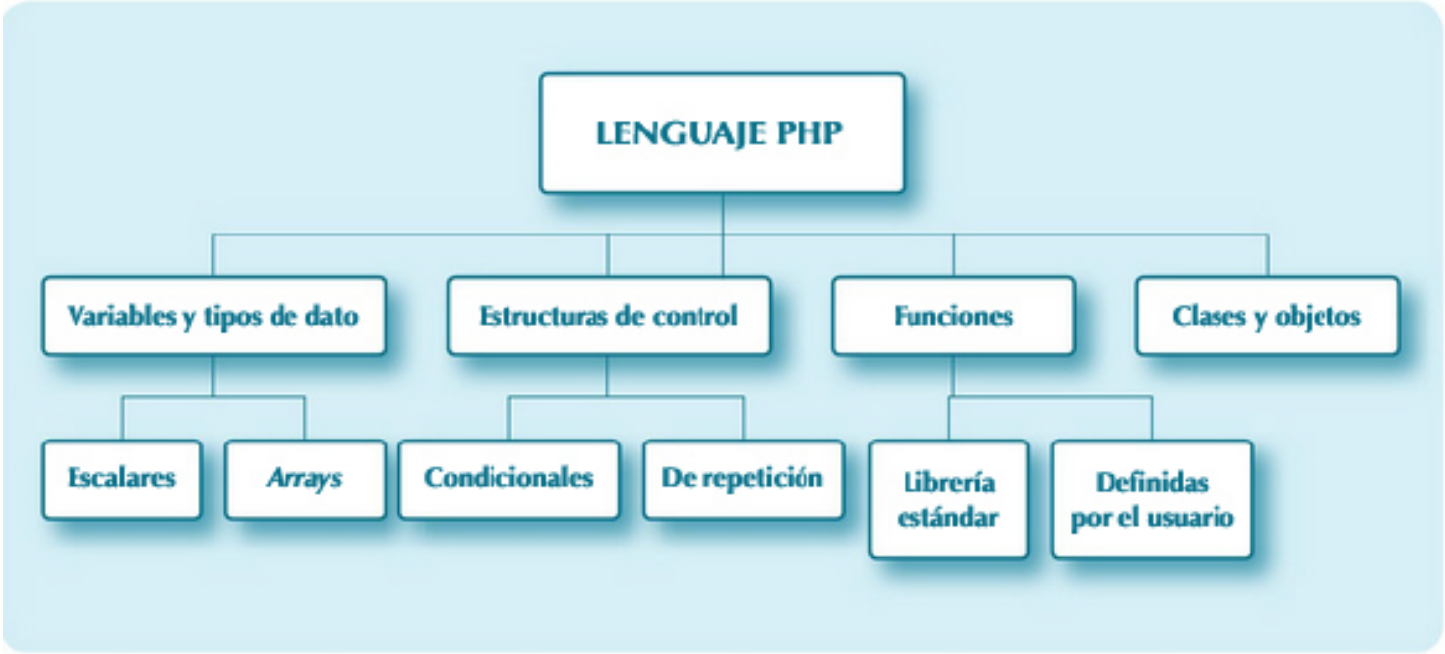
DWES DAW

TEMA 2

INTRODUCCIÓN AL LENGUAJE PHP

MAPA CONCEPTUAL



1. PHP Y HTML. CÓDIGO INCRUSTADO

2. SINTAXIS DE PHP

3. VARIABLES Y TIPOS DE DATOS

Tipos de Datos

Tipo	Descripción
integer	Números enteros
float	Números reales en coma flotante
string	Cadenas de caracteres
boolean	Booleanos, TRUE o FALSE
array	Colección de elementos identificados
object	Un objeto es una instancia de una clase
callable	Para las funciones de *callback*
null	Para representar variables no asignadas
resource	Representa recursos externos

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

Variables Predefinidas

En PHP hay muchas variables predefinidas disponibles. 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. Cada una de ellas guarda información de un tipo.

Por ejemplo, en $_SERVER hay información sobre el servidor en el que está alojada la página.

El script global ***server.php*** muestra algunos de los datos disponibles.

```php
<?php
    echo "Ruta dentro de htdocs: ". $_SERVER['PHP_SELF'];
    echo "Nombre del servidor: ". $_SERVER['SERVER_NAME'];
    echo "Software del servidor: ". $_SERVER['SERVER_SOFTWARE'];
    echo "Protocolo: ". $_SERVER['SERVER_PROTOCOL'];
    echo "Método de la petición: ". $_SERVER['REQUEST_METHOD'];
```

Las 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

4. COMENTARIOS

En PHP se pueden utilizar comentarios:

- De bloque, encerrados entre “/* y “*/.
- De línea, comenzando por “//” o por “#”

Ejemplo

```php
<?php
    //comentario de línea
    $a = 0; // comentario de línea
    /* comentario
    de bloque
    */
    # comentario de una sola línea
```

5. ESTRUCTURAS DE CONTROL

5.1. Estructuras condicionales

Simple

```php
<?php
    $var = 3;
    if($var < 0) echo "Es menor que cero";
    if ($var > 0){
        echo "Es mayor que cero";
    }
```

Doble

```php
<?php
    $var = 3;
    if($var < 0){
        echo "Es menor que cero";
    }else{
        echo "Es mayor o igual que cero";
    }
```

Anidado

```php
<?php
    $var = 3;
    if ($var == 1) {
        echo "Es un uno";
    }elseif ($var == 2) {
        echo "Es un dos";
    }elseif ($var == 3) {
        echo "Es un tres";
    }else{
        echo "No es un uno, ni un dos, ni un tres"
    }
```

Switch:

Ver breakSwitch.php

```php
<?php
    $var = 3;
    switch($var){
        case 1:
            echo "Es un 1";
        break;
        case 2:
            echo "Es un 2";
        break;
        case 3:
            echo "Es un 3";
        break;
        default:
            echo "No es un 1, ni un 2, ni un 3";
    }
```

Nota1:

Según el valor de $var, se ejecutará un case u otro, La sección ***default*** se ejecuta cuando no se da ninguno de los case.

Nota2:

Si no hay un ***break*** al final de un case, la ejecución continúa con el siguiente.

5.2. Estructuras de repetición

For:

```
<?php
    for($i = 0; i < 5; $i = $i + 1){
        echo "$i <br>";
    }
```

While:

```
<?php
    $i = 0;
    while($i < 5){
        echo "$i <br>";
        $i = $i +1;
    }
```

Do-while:

```
<?php
    $i = 0;
    do{
        echo "$i <br>";
        $i = $i +1;
    } while ($i < 5);
```

Otro Do-while:

```php
<?php
    $i = 0;
    do {
        echo "En el do-while: $i <br>";
        $i = $i + 1;
    } while ($i < 0);
    while ($i < 0) {
        echo "En el while: $i <br>";
        $i = $i + 1;
    }
```

Break:

Ver *break.php*

Dentro de un bucle es posible usar las sentencias break y continue. La primera sirve para salir del bucle o switeh en el que aparece. En el ejemplo break.php, el bucle acaba antes de llegar a cinco porque al llegar a tres se ejecuta el break.

```php
<?php
    $i = 0;
    while ($i < 5){
        echo "$i <br>";
        $i++; // es lo mismo que $i = $ i + 1;
        if ($i == 3){
            break;
        }
    }
```

Break con número:

En PHP la sentencia break admite un número, que representa el número de niveles de anidación de los que debe salir. Así se puede salir de un bucle anidado utilizando una única secuencia break como se puede ver a continuación.

```php
<?php
    echo "Primer for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
            if ($j == 1) {
                break; //es lo mismo que poner break 1
            }
        }
    }
    echo "Segundo for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
                if ($j == 1){
                    break 2;
            }
        }
    }
```

Break con número en Switch

Esto también se aplica a la sentencia condicional switch. Por ejemplo, si un switch está dentro de un bucle, utilizando break 2 se sale de ambos.

```php
<?php
    $i = 0;
    echo "Primer switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
    $i = 0;
    echo "Segundo switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break 2;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
```

Continue:

Ver continueFor.php

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.

En el ejemplo continue.php se puede observar el funcionamiento de continue.

```php
<?php
    for($i = 0; $i < 5; $i = $i + 1){
        if ($i == 3){
            continue;
        }
        echo "$i <br>";
    }
```

5.3. Otras estructuras de control

Bucles anidados:

Ver anidadoFor.php

Ya lo estudiaremos

Es posible incluir otros ficheros utilizando las sentencias include y require.
include “mifichero.php”;
require “mifichero.php”;

6. OPERADORES

Ver identico.php

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.

7. ARRAYS

Ya los estudiaremos más adelante

Ver arrays.php

Operadores para arrays

Operador	Descripción
$a1 === $a2	Idéntico. Verdadero si los dos *arrays* tienen las claves y valores iguales, en el mismo orden y del mismo tipo.
$a1 !== $a2	No idéntico.
$a1 == $a2	Igual. Verdadero si los dos *arrays* tienen las claves y valores iguales.
$a1 != $a2, $a1 <> $a2	No igual.
$a1 + $a2	Unión. Devuelve un *array* con los elementos de ambos.

8. FUNCIONES Y LIBRERÍAS

Ya las estudiaremos más adelante

Funciones predefinidas:

- *Is_null($var). Devuelve TRUE si $var es NULL, FALSE en otro caso.*
- *isset($var). Devuelve TRUE si svar ha sido inicializada y su valor no es NULL, FALSE en otro caso.*
- *unset(Svar). Elimina la variable. Ya no contará como inicializada.*
- *empty($var). Devuelve TRUE si svar no ha sido inicializada o su valor es FALSE, FALSE en otro caso.*
- *is int($var),is float(Svar),is bool(Svar), is array($var). Devuelven TRUE*
- *si $var es entero, float, booleano O array respectivamente, y FALSE en otro caso.*
- *print_r($var) y var_dump($var). Muestran información sobre $var*

Ver funciones.php

Ver funcionesVariables.php

Funciones de Variables

Funciones de variables	
isset($var)	TRUE si la variable está inicializada y no es NULL
is_null($var)	TRUE si la variable es NULL
empty($var)	TRUE si la variable no está inicializada o su valor es FALSE
is_int($var), is_float($var), is_bool($var), is_array($var)	Para comprobar el tipo de dato de $var
intval($var), floatval($var), boolvar($var), strval($var)	Para obtener el valor de $var como otro tipo de dato

Funciones de Cadenas

Funciones de cadenas	
strlen($cad)	Devuelve la longitud de $cad
explode($cad, $token)	Parte una cadena utilizando $token como separador. Devuelve un *array* de cadenas
implode($token, $array)	Crea una cadena larga a partir de un *array* de cadenas, entre cadena y cadena se introduce $token
strcmp($cad1, $cad2)	Compara las dos cadenas. Devuelve 0 si son iguales, -1 si $cad1 es menor y 1si $cad1 es mayor
strtolower($cad), strtoupper($cad)	Devuelven $cad en mayúsculas o minúsculas, respectivamente
str($cad1, $cad2)	Busca la primera ocurrencia de $cad2 en $cad1. Si no aparece devuelve FALSE, si aparece devuelve $cad1 desde donde comienza la ocurrencia

Funciones de Arrays

Funciones de *arrays*	
ksort($arr), krsort($arr)	Ordena el *array* por clave en orden ascendente o descendente
sort($arr), rsort($arr)	Ordena el *array* por valor en orden ascendente o descendente
array_values($arr)	Devuelve los valores de $arr
array_keys($arr)	Devuelve las claves de $arr
array_key_exists($arr, $cla)	Devuelve verdadero si algún elemento de $arr tiene clave $cla
count($arr)	Devuelve el número de elementos del *array*

Funciones definidas por el usuario:

```php
<?php
    function suma($a, $b)   {
        return $a + $b;
    }
    echo suma(4,8).'<br>';
    $var1 = 35;
    $var2 = 5;
    $var3 = suma($var1, $var2);
    echo $var3.'<br>';
```

Paso de parámetros por valor

Ya las estudiaremos más adelante

Paso de parámetros por referencia

Ya las estudiaremos más adelante

Funciones como argumentos

Ya las estudiaremos más adelante

```php
<?php
    function calculador($operacion, $numa, $numb){
        $resul = $operacion($numa, $numb);

       return $resul;
   }
   function sumar($a, $b){
       return $a + $b;
   }
   function multiplicar($a, $b){
       return $a * $b;
   }
   $a = 4;
   $b = 5;
   $r1 = calculador("multiplicar", $a, $b);
   echo "$r1 <br>";
   $r2 = calculador("sumar", $a, $b);
   echo "$r2 <br>";
```

9. EXCEPCIONES Y ERRORES

Error vs excepción:

Los **errores** en PHP son **fallos** que ocurren durante la ejecución de un script y que impiden que se complete correctamente. Estos errores pueden ser de diferentes tipos, como **errores de sintaxis**, **errores de ejecución** o **errores de tiempo de compilación**. Los errores suelen generar mensajes en el registro de errores del servidor y, en algunos casos, pueden detener la ejecución del script.

Por otro lado, las **excepciones** en PHP son **eventos inesperados** que ocurren durante la ejecución de un script, pero que se pueden controlar y **manejar** de manera programática. Las excepciones permiten al desarrollador anticiparse a situaciones problemáticas y definir cómo se deben manejar. Al lanzar una excepción, se interrumpe el flujo normal del programa y se busca un bloque de código preparado para capturar y gestionar esa excepción.

Desde PHP 5 hay un sistema de excepciones similar al de Java y otros lenguajes utilizando bloques try/catch/finally.
Finalmente, en PHP 7 aparecieron las excepciones de clase Error.

Ante determinadas situaciones PHP genera un error

> **NOTA:** Se verá en un tema posterior
>
> *Aunque PHP incluye un depurador a partir de la versión 5.6, lo habitual es usar un depurador externo integrado en un IDE. Los depuradores más extendidos son Xdebug y ZendDebugger*

Tipos de errores

Hay diferentes **tipos de errores**, cada uno asociado con un número y una constante predefinida. Se puede controlar cómo se comporta PHP antes los errores mediante tres directivas del fichero ***php.ini***:

- ***error_reporting***: indica qué errores deben reportarse. Lo normal es utilizar ***E_ALL***, es decir, todos.
- ***display errors***: señala si los mensajes de error deben aparecer en la salida del script. Esta opción es apropiada durante el desarrollo, pero no en producción.
- ***log errors***: indica si los mensajes de error deben almacenarse en un fichero. Es especialmente útil en producción, cuando no se muestran los errores en la salida.
- ***error log***: si la directiva anterior (*log_errors*) 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.

Se ven los **códigos** y mensajes asociados en la tabla siguiente:

Código	Constante	Descripción
1	E_ERROR	Error fatal en tiempo de ejecución. La ejecución del *script* se detiene
2	E_WARNING	Advertencia en tiempo de ejecución. El *script* no se detiene
4	E_PARSE	Error de sintaxis al compilar
8	E_NOTICE	Notificación. Puede indicar error o no
16	E_CORE_ERROR	Error fatal al iniciar PHP
32	E_CORE_WARNING	Advertencia al iniciar PHP
64	E_COMPILE_ERROR	Error fatal al compilar
128	E_COMPILE_WARNING	Advertencia fatal al compilar
256	E_USER_ERROR	Error generado por el usuario
512	E_USER_WARNING	Advertencia generada por el usuario
1024	E_USER_NOTICE	Notificación generada por el usuario
2048	E_STRICT	Sugerencias para mejorar la portabilidad
4096	E_RECOVERABLE_ERROR	Error fatal capturable
8192	E_DEPRECATED	Advertencia de código obsoleto
16384	E_USER_DEPRECATED	Como la anterior, generada por el usuario
32767	E_ALL	Todos los errores

Funciones relacionadas con manejo de errores

https://www.php.net/manual/es/ref.errorfunc.php

- ***error_reporting():***

La función *error_reporting()* permite cambiar el valor de la directiva *error_reporting* en tiempo de ejecución.

- ***set_error_handler():***

Esta función permite definir una función propia para que se encargue de los errores.

```
set_error_handler(callable $error_handler, int $error_types = E_ALL | E_STRICT): mixed
```

La función *error_handler* que se ocupe de los errores tendrá que tener la siguiente forma:

```
handler(
    int $errno,
    string $errstr,
    string $errfile = ?,
    int $errline = ?,
    array $errcontext = ?
): bool
```

Ejemplo:

manejadorErrores.php

El siguiente ejemplo muestra cómo utilizar *set_error_handler()* para manejar los errores con una función propia.

```php
function manejadorErrores($errno, $str, $file, $line)
{
    echo "Ocurrió el error: $errno";
}
set_error_handler ("manejadorErrores");
echo "Intenta asignar a una variable el valor de otra, pero la segunda"
. " no tiene valor asignado<br><br>";
$a = $b; // causa error, 5b no está inicializada
echo '<br>.....';
```

Salida:

Intentamos asignar a una variable el valor de otra, pero la segunda no tiene valor asignado

Ocurrió el error: 2
.....

Nota:

Ver tabla anterior. Devuelve error 2 y la ejecución del script nop se detiene, pues es un tipo warning:

2	E_WARNING	Advertencia en tiempo de ejecución. El *script* no se detiene

Excepciones

Otra opción para indicar un error es lanzar una **excepción**.

Para controlar las excepciones se utilizan bloques ***try/catch/finally***, como en Java.

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

Cuando se lanza una excepción y no es capturada por un bloque catch, la ejecución del programa se **detiene**.

Para capturar una excepción se introduce la instrucción que puede causarla dentro de un bloque try y se añade el bloque *catch* correspondiente.

Si es capturada, se ejecuta el código del bloque **correspondiente**.

Se puede añadir un bloque *finally*, que se ejecuta después del try/catch, haya habido excepción o no.

Ejemplo:

excepciones.php

En la primera llamada a dividir(), línea 12, se produce una excepción. Por tanto, el resto del bloque try, el echo, no se ejecuta. Si se ejecutan el catch y el *finally* correspondientes.

En la segunda llamada a dividir(), línea 23, no se produce excepción. En este caso se ejecutan el echo del try y a continuación el del bloque *finally*. El bloque catch no se ejecuta.

```
3       function dividir($a, $b)
4       {
5        if ($b==0){
6            throw new Exception('El segundo argumento es 0');
7        }
8            return $a/$b;
9       }
10      /*bloque principal*/
11       try{
12           $resul1 = dividir(5, 0);
13           echo "Resultado1:  $result".  "<br>";
14        }
15       catch (Exception $e){
16           echo "Excepción: ". $e->getMessage() . "<br>";
17       }
18       finally{
19            echo "Primer finally<br><br>";
20       }
21
22       try{
23            $resul2 = dividir(5, 2);
24            echo "Resultado2: $resul2". "<br>";
25       } catch (Exception $e){
26            echo "Excepción: ". $e->getMessage() . "<br>";
27       }
28       finally{
29           echo "Segundo finally";
30       }
```

Salida:

```
Excepción: El segundo argumento es 0
Primer finally

Resultado2: 2.5
Segundo finally
```

9.3. Excepciones error

Enlace:

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

En PHP 7 aparecieron las excepciones de tipo Error. No heredan de la clase *Exception*, 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){
    ...
}
```

Excepciones Error predefinidas (algunas)

Nombre	Descripción	Hereda de
Error	Clase base para las excepciones Error	
ArithmeticError	Error en operaciones matemáticas. Hereda del anterior	Error
DivisionByZeroError	Intento de división por cero. Hereda del anterior	ArithmeticError
AssertionError	Ocurre cuando falla una llamada a assert()	Error
ParseError	Error al compilar	Error
TypeError	Ocurre cuando una expresión no tiene el tipo de dato que se espera	Error
ArgumentCountError	Ocurre al llamar a una función con menos argumentos de los necesarios. Hereda del anterior	TypeError

Ver: https://www.php.net/manual/es/reserved.exceptions.php

Ejemplo arithmetic error (con clase *Error*):

```php
try {
    echo "Intentamos aplicar operador SHIFT con valor negativo<br><br>";
    $a = 10;
    $b = -3;
    $result = $a << $b;
}
catch (ArithmeticError $e) {
    echo $e->getMessage();
}
```

Salida:

Intentamos aplicar operador SHIFT con valor negativo

Bit shift by negative number

Ejemplo arithmetic error (con clase *Throwable*):

```php
try {
    echo "Intentamos aplicar operador SHIFT con valor negativo<br><br>";
    $a = 10;
    $b = -3;
    $result = $a << $b;
}
catch (Throwable $e) {
    echo 'El error es: <br>'.$e->getMessage();
}
```

Salida:

Similar al a anterior

9.4. Excepciones definidas por el usuario

Podemos definir excepciones personalizadas, heredando de la clase ***Exception***

Si generas con **throw** una excepción propia, esta será manejada por el primer bloque **catch** que la capture de forma específica

Si no existe un bloque catch específico de la excepción, ésta será manejada por el primer bloque catch que capture la excepción Exception

Es importante poner los bloques catch de las excepciones **específicas** antes que los de las **generales**
Si una clase se extiende de la clase Exception incorporada y redefine el constructor, es recomendable que llame *parent::__construct()* para asegurarse que todos los datos disponibles han sido asignados apropiadamente.

El método __toString() puede ser evitado para proveer una salida personalizada cuando el objeto es presentado como una cadena.

Estructura Recomendada:

excepcionUsuario.php

```php
class MyException extends Exception
{
    // Redefine el mensaje de la excepción
    public function __construct($message, $code = 0) {
        // some code

        // asegurar que hace todo lo que debe hacer
        parent::__construct($message, $code);
    }

    // Representación del objeto */
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }

    public function customFunction() {
        echo "A Custom function for this type of exception\n";
    }
}
```

Ejemplo:

falta VALIDAR Y salida

excepcionValidation.php

```php
class ValidationExcepcion extends Exception {
    public function __construct($campo, $valor)
    {
        if (empty($valor))
            $message = "El campo $campo está vacío";
        else $message = "El campo $campo no es correcto. "
                . "Valor actual: $valor";
        parent::__construct($message, 0, null);
    }

    public function __tostring() {
        return $this->getMessage();
    }
    public function RegistraError() {
        error_log($this->getMessage(), 0);
    }
}
```

excepcionValidation_usa.php

```php
try {
    if (isset($_POST['enviar']))
    {
        if (isset($_POST['usuario']))
        {
            if (empty($_POST['usuario']))
                throw new ValidationExcepcion("usuario", $_POST['usuario']);
            elseif($_POST['usuario'] !== "alex")
                throw new ValidationExcepcion("usuario", $_POST['usuario']);
            else
                echo "Validación OK"; }
            else
                throw new ValidationExcepcion("usuario", "");

        }

        }
        catch (ValidationExcepcion $e) {
            echo $e;
            $e->RegistraError();
    }
```

formBasico.php:

```php
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Formulario Básico</title>
    </head>
    <body>
        <form name="formBasico" method="post" action="exceptionValidation_usa.php">
            Usuario: <input type="text" name="usuario" value="" size="20" />
            <input type="submit" value="Envia" name="botEnvia" />
        </form>
    </body>
</html>
```

10. CLASES Y OBJETOS

https://www.php.net/manual/es/language.oop5.php

PHP tiene soporte completo para la **programación orientada a objetos (POO)**.

Permite definir *clases, herencia, interfaces* y los demás elementos habituales.

Para declarar una clase se utiliza la palabra reservada *class*.

Los **atributos** se declaran utilizando su nombre, los modificadores que pueda tener y opcionalmente un valor por defecto.

```
class Clase {
    private $att1 = 10; // con valor por defecto
    private $atr2; / sin valor por defecto
    private static $atr3 = 0; // estático
....
}
```

Los **métodos** son funciones definidas dentro de una clase.

Los **atributos** y **métodos** se pueden declarar como **estáticos**, de manera que no habrá uno por objeto, sino uno por clase.

Se utiliza la palabra reservada ***static***.

Se puede crear un constructor para la clase con el método ***__construct().*** Este método forma parte de los **métodos mágicos**, nombres reservados a tareas concretas y que comienzan por dos guiones bajos, "_.

Para crear un nuevo objeto se utiliza el operador **new** seguido del nombre de la clase:

$0bj = new Clase();

Para los atributos y métodos se pueden utilizar los siguientes modificadores de **visibilidad**:

- ***public***. Se pueden utilizar desde dentro y fuera de la clase.
- **private**. Pueden emplearse desde la propia clase.
- **protected**. Se pueden utilizar dentro de la propia clase, las derivadas y las antecesoras.

Normalmente los **atributos se declaran como privados** y se crean **métodos públicos** para acceder a ellos.

Para acceder a los métodos y atributos se utiliza:

$objeto->propiedad;

$objeto->método(argumentos);

En los métodos **no estáticos** se puede utilizar la palabra reservada ***this***, que representa el objeto desde el que se invoca el método.

Ejemplo:

clasePersona.php

```php
class clasePersona {

    public $name;
    public $surname;
    public $age;

    public function __construct($name, $surname, $age){

        $this->name = $name;
        $this->surname = $surname;
        $this->age = $age;

    }

    public function presentate( ){

        echo "Hola, soy $this->name $this->surname "
                . "y tengo $this->age anyos.<br>";

    }
}

$persona = new clasePersona('Giuseppe', 'Verdi', 56);

$persona->presentate( );
```

Métodos Mágicos:

https://www.php.net/manual/es/language.oop5.magic.php

- __construct()
- __destruct()
- __call()
- __callStatic()
- __get()
- __set()
- __isset()
- __unset()
- __sleep()
- __wakeup()
- __serialize()
- __unserialize()
- __toString()
- __invoke()
- __set_state()
- __clone()
- __debugInfo().

Ejemplo

El ejemplo *PersonayCliente.php* muestra cómo crear una clase y crear y manejar objetos de esta. Para empezar, declara la clase Persona, con atributos para nombre, apellido y DNI. Los tres son privados. La clase tiene un constructor, el método __construct (), que recibe valores para los tres atributos. A continuación, se declaran una serie de métodos públicos para leer y escribir los atributos, los llamados *getters* y setters (líneas 11-23).

```
1  <?php
2  class Persona {
3      private $DNI;
4      private $nombre;
5      private $apellido;
6      function __construct($DNI, $nombre, $apellido) {
7          $this->DNI = $DNI;
8          $this->nombre = $nombre;
9          $this->apellido = $apellido;
10     }
11     public function getNombre() {
12         return $this->nombre;
13     }
14     public function getApellido() {
15         return $this->apellido;
16     }
17     public function setNombre($nombre) {
18         $this->nombre = $nombre;
19     }
20
21     public function setApellido($apellido) {
22         $this->apellido = $apellido;
23     }
24     public function __toString() {
25         return "Persona: ".$this->nombre." ".$this->apellido;
26     }
27 }
```

En esta clase también se utiliza *__toString()* (líneas 24-26), otro método mágico que se usa para generar una cadena de texto con la información del objeto. Cuando se pasa un objeto a echo, se representa usando el valor de retorno de *__toString().*

En las líneas 46-52 se crea un objeto de la clase Persona y se manipula.

```
45 // crear una persona
46 $per = new Persona("1111111A", "Ana", "Puertas");
47 // mostrarla, usa el método __toString()
48 echo $per. "<br>";
49 // cambiar el apellido
50 $per->setApellido("Montes");

51 // volver a mostrar
52 echo $per. "<br>";
```

En text:

45 // crear una persona

46 $per = new Persona("1111111A", "Ana", "Puertas");

47 // mostrarla, usa el método _ toString()

48 echo Sper. "
";

49 // cambiar el apellido

50 Sper->setApellido(""Montes");

51 // volver a mostrar

52 echo S$per. "
";

La salida de este fragmento será:

Persona: Ana Puertas

Persona: Ana Montes

Para crear una clase que herede de otra, se utiliza la palabra clave *extends*. La clase derivada tendrá los mismos atributos y métodos que la clase base y podrá añadir nuevos o sobrescribirlos. En el ejemplo anterior se declara también la clase Cliente, que hereda de persona y añade un atributo saldo. Incluye también los métodos getSaldo() y setSaldo( ) y sobrescribe el método __ toString() de la clase base. También hay un constructor que incluye el muevo atributo y utiliza el constructor de la clase base.

```
class Cliente extends Persona{
    private $saldo = 0;
    function __construct($DNI, $nombre, $apellido, $saldo){
        parent::__construct($DNI, $nombre, $apellido);
        $this->$saldo = $saldo;
    }
    public function getSaldo(){
        return $this->saldo;
    }
    public function setSaldo($saldo){
        $this->saldo = $saldo;
    }
    public function __toString(){
        return "Cliente: ". $this->getNombre();
    }
}
```

En las líneas 54-56 se crea un objeto de la clase cliente y se muestra. La salida será “Cliente: Pedro”.

```
$cli = new Cliente("22222245A", "Pedro", "Sales", 100);
// lo muestra
echo $cli. "<br>";
```

INTERFACES

Las **interfaces** de objetos son contratos que han de cumplir las **clases** que las implementan.

Contienen **métodos vacíos** que obligan a una clase a emplearlos, promoviendo así un **estándar de desarrollo**.

Se utilizan cuando se tienen muchas clases que tienen en común un comportamiento, pudiendo asegurar así que ciertos métodos estén disponibles en cualquiera de los objetos que queramos crear. Son especialmente importantes para la **arquitectura de aplicaciones complejas**.

Para definir una interface se utiliza la parabra ***interface***, y para extenderla se utiliza ***implements***.

Si una clase implementa una ***interface***, está obligada a usar todos los métodos de la misma (y los mismos tipos de argumentos de los métodos), de lo contrario dará un **error fatal**.

Pueden emplearse **más de una interface en cada clase**, y pueden **extenderse** entre ellas mediante *extends*. Una interface puede extender una o más interfaces.

Todos los métodos declarados en una interface deben ser **públicos**.

Un interface puede incluir **constantes**, pero no atributos

Un interface puede heredar de otro utilizando **extends**

PHP tiene una serie de interfaces ya definidos, por ejemplo, el interface **Countable**

interface iVehiculo	clase auto	clase moto

`interface E12_iVehiculo{`
`  public function getTipo();`
`  public function getRuedas();`
`}`
	`require_once 'E12_iVehiculo.php';`
	
	`class E12_auto implements E12_iVehiculo{`
	`  public function getTipo(){`
	`  echo "Coche";`
	`  }`
	`  public function getRuedas(){`
	`  echo "4";`
	`  }`
	`}`
		`require_once 'E12_iVehiculo.php';`
		
		`class E12_moto implements E12_iVehiculo{`
		`  public function getTipo(){`
		`  echo "Moto";`
		`  }`
		`  public function getRuedas(){`
		`  echo "2";`
		`  }`
		`}`

11- BIBLIOGRAFÍA

Desarrollo Web En entorno Servidor

Editorial Síntesis

Xabier Ganzábal García

isbn: 978-54-917183-6-9