TEMA 2
PHP CLASES. ERRORES Y EXCEPCIONES

1. 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 Ejemplo {
        private $atr1 = 10;          // privado con valor por defecto
        private $atr2;               // privado sin valor por defecto
        private static $atr3 = 0;    // estático con valor por defecto
....
}
```

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***, es decir **__**

Clase con atributos y constructor. No tiene métodos

```php
class Persona {
        private $DNI;
        private $nombre;
        private $apellido;
        function __construct($DNI, $nombre, $apellido) {
                $this->DNI = $DNI;
                $this->nombre = $nombre;
                $this->apellido = $apellido;
        }
}
```

Creación de un objeto sin contenido:

Se utiliza el operador ***new*** seguido del nombre de la clase

$0bj = new Clase();

Creación y uso de un objeto con contenido usando el CONSTRUCTOR:

Se usa el constructor, que es el nombre de la clase con todos los argumentos

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

// mostrarla, usa el método __toString()
echo $per . "<br>";
```

Creación y uso de un objeto SIN CONSTRUCTOR:

Asignamos los valores de atributos *A Mano*

Nota: Es necesario definir los atributos como ***public*** para que funcione

```php
class PersonaSinConstr {
        public $DNI;
        public $nombre;
        public $apellido;
}

// crear una persona SIN usar constructor
$per2 = new PersonaSinConstr();

// Asignar valores A Mano, sin Constructor
$per2->DNI='22222222B';
$per2->nombre='Pepe';
$per2->apellido='García';

// Visualizamos los atributos
echo $per2->DNI ."<br>";
echo $per2->nombre ."<br>";
echo $per2->apellido ."<br>";
```

Salida:

22222222B
Pepe
García

Visibilidad de atributos y métodos

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.

Acceso a los métodos de una clase

$objeto->método(argumentos);

Acceso a los atributos de una clase

$objeto->propiedad;

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

Creación y uso de un objeto CON CONSTRUCTOR y algún método:

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>";

    }
}
```

Salida:

Hola, soy Giuseppe Verdi y tengo 46 anyos.

Métodos Mágicos:

Los métodos mágicos son métodos especiales que **sobreescriben** acciones por defecto cuando se realizan ciertas acciones sobre un objeto.

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

Los siguientes nombres de métodos se consideran mágicos

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

Definición de la clase Persona con getters y sétters

```php
class Persona {
        private $DNI;
        private $nombre;
        private $apellido;
        function __construct($DNI, $nombre, $apellido) {
                $this->DNI = $DNI;
                $this->nombre = $nombre;
                $this->apellido = $apellido;
        }

        public function getNombre() {
                return $this->nombre;
        }
        public function getApellido() {
                return $this->apellido;
        }
        public function setNombre($nombre) {
                $this->nombre = $nombre;
        }

        public function setApellido($apellido) {
                $this->apellido = $apellido;
        }
        public function __toString() {
                return "Persona: " . $this->nombre . " ". $this->apellido;
        }
}
```

Nota:

En esta clase también se utiliza ***__toString()***, 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().***

Ejemplo: Manipulación de objeto de la clase Persona con setters y getters

```php
// crear una persona usando su constructor
$per = new Persona("1111111A", "Ana", "Puertas");

// mostrarla, usa el método __toString()
echo $per . "<br>";

// cambiar el apellido
$per->setApellido("Montes");
// volver a mostrar
echo $per . "<br>";
```

Salida:

Persona: Ana Puertas

Persona: Ana Montes

2. HERENCIA:

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.

Ejemplo1: Clase Cliente que hereda de Persona

- Incluye también los métodos getSaldo() y setSaldo( )
- Añade un atributo saldo
- 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.

Ejemplo: Definición de clase Cliente que hereda de la clase Persona

Clase Padre

```php
class Persona {
        private $DNI;
        private $nombre;
        private $apellido;
        function __construct($DNI, $nombre, $apellido) {
                $this->DNI = $DNI;
                $this->nombre = $nombre;
                $this->apellido = $apellido;
        }
        public function getNombre() {
                return $this->nombre;
        }
        public function getApellido() {
                return $this->apellido;
        }
        public function setNombre($nombre) {
                $this->nombre = $nombre;
        }
        public function setApellido($apellido) {
                $this->apellido = $apellido;
        }
        public function __toString() {
                return "Persona: " . $this->nombre . " ". $this->apellido;
        }
}
```

Clase Hija

```php
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  parent::getNombre().'<br>'.
                        parent::getApellido().'<br>'.
                        "Cliente: ". $this->getNombre().'<br>'.
                        ' Saldo: '.$this->getSaldo() ;
        }
}
```

Creación y uso de objjeto Cliente que hereda de Persona

```php
// crea un cliente
$cli = new Cliente("22222245A", "Pedro", "Sales", 100);
// modificar su saldo
$cli->setSaldo(2000);
// Mostrar datos
echo $cli . "<br>";
```

Salida

Nombre Cliente: Pedro
Apellido Cliente: Sales
Saldo: 2000

3. 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**.

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

Uso de interfaces

- 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.
- Especialmente importantes para la **arquitectura de aplicaciones complejas**.

Condiciones en el uso de interfaces:

- 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**

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

Ejemplo:

Declarar interface iVehiculo	Implementar clase auto	Implementar clase moto

`interface E11_iVehiculo{`
`  public function getTipo();`
`  public function getRuedas();`
`}`
	`require_once 'E11_iVehiculo.php';`
	
	`class E11_auto implements E11_iVehiculo{`
	`  public function getTipo(){`
	`  echo "Coche";`
	`  }`
	`  public function getRuedas(){`
	`  echo "4";`
	`  }`
	`}`
		`require_once 'E11_iVehiculo.php';`
		
		`class E11_moto implements E11_iVehiculo{`
		`  public function getTipo(){`
		`  echo "Moto";`
		`  }`
		`  public function getRuedas(){`
		`  echo "2";`
		`  }`
		`}`

Ejemplo: Uso del interfaz

```php
require_once 'E11_iVehiculo.php';
require_once 'E11_auto.php';
require_once 'E11_moto.php';

// Crear una instancia de la clase auto
$vehic1 = new E11_auto();
$vehic1->getTipo();

echo '<br>';
// Crear una instancia de la clase moto
$vehic2 = new E11_moto();
$vehic2->getTipo();
```

salida:

Coche
Moto

4. ERRORES Y EXCEPCIONES

Errores

Son **fallos** que ocurren durante la ejecución de un script y que impiden que se complete correctamente.

Los errores suelen generar mensajes en el registro de errores del servidor y, en algunos casos, pueden detener la ejecución del script.

Clasificación de errores:

- errores de *sintaxis*
- errores de *ejecución*
- errores en tiempo de compilación.

Excepciones

Son **eventos inesperados** que ocurren durante la ejecución de un script,

Las excepciones se pueden controlar y definir cómo se deben **manejar**.

Desde PHP 5 hay un sistema de excepciones similar al de Java y otros lenguajes utilizando bloques ***try/catch/finally***.

En PHP 7 aparecieron las excepciones de clase ***Error***.

Qué ocurre cuán se produce una excepción:

- Se **interrumpe** la ejecución del programa
- Se **almacena** el estado actual del código
- Se busca un bloque de código, **exception handler predefinido,** preparado para gestionar esa excepción.
- Ese exception handler **predefinido** (manejador) se puede cambiar mediante la función ***set_error_handler()***

4.1 ERRORES

Tipos de errores

Hay diferentes **tipos de errores**, cada uno asociado con un número y una constante predefinida.

Directivas PHP para control de comportamiento ante errores

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**.
    - 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.
- ***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.

CÓDIGOS DE ERROR. CONSTANTES PREDEFINIDAS Y MENSAJES DE ERROR

En la tabla siguiente se ven los **códigos,** el nombre de **constante predefinida** y los **mensajes** asociados

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 (definida por el usuario)*** para que se encargue de los errores.

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

La función definida por el usuario, *error_handler,* que se ocupe del error tendrá que tener la siguiente forma:

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

Ejemplo 1: Función propia (definida por el usuario) que sustituye el comportamiento de php ante un error determinado.

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 **no se detiene**, pues es un **tipo warning**:

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

4.2 EXCEPCIONES

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

El suario puede lanzar una excepción usando ***thrown***

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

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

Comportamiento:

- Cuando se lanza una excepción y no es capturada por un bloque ***catch***, la ejecución del programa se **detiene**.
- Si una excepción no es capturada, se emitirá un ***Error Fatal*** de PHP con un mensaje "***Uncaught Exception ...***" ("Excepción No Capturada"), a menos que se haya definido un manejador con set_exception_handler().
- 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 va al ***catch***, 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**.

EXCEPCIONES CONVENCIONALES (LANZADAS POR EL USUARIO)

Ejemplo 2: Excepciones convencionales

excepciones.php

```php
<?php
function dividir($a, $b){
                if ($b==0){
                        throw new Exception('El segundo argumento es 0');
                }
                return $a/$b;
        }

/* PROGRAMA PRINCIPAL 1 */

    try{
            $resul1 = dividir(5, 0);
            echo "Resul 1 $resul1". "<br>";
    }catch(Exception $e){
            echo "Excepción: ". $e->getMessage(). "<br>";
    }finally{
            echo "Primer finally<br>";
    }


/* PROGRAMA PRINCIPAL 2 */

    try{
            $resul2 = dividir(5, 2);
            echo "Resul 2 $resul2". "<br>";
    }catch(Exception $e){
            echo "Excepción: ". $e->getMessage(). "<br>";
    }finally{
            echo "Segundo finally";
    }
```

Salida:

Excepción: El segundo argumento es 0
Primer finally

Resultado2: 2.5
Segundo finally

Ejemplo: Múltiples excepciones

El fragmento contempla varios posibles casos de excepción

emailExcep.php

```php
// Ponemos un email no válido para forzar la excepción
$email = "ejemplo@ejemplo...com";
// Ponemos un email válido
$email = "ejemplo@ejemplo.com";

// Iniciamos el bloque try
try {
    // Comprobar si el email es válido
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        // Lanza una excepción si el email no es válido
       $cad="<b>$email</b> ".'No tiene formato válido<br>';

       throw new Exception($cad);
    }
    else{
        $cad="<b>$email</b> ".'Tiene formato válido<br>';
        throw new Exception($cad);
    }
}

catch (Exception $noValido) {
    echo $noValido->getMessage();
}
catch(Exception $valido) {
    echo $valido->getMessage();
}
finally{
    //echo "Algo mal";
}
```

Salída:

ejemplo@ejemplo...com No tiene formato válido

EXCEPCIONES TIPO ERROR

En PHP 7 aparecieron las excepciones de tipo ***Error***.

Error es la clase base para todos los **errores internos** de PHP.

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

- No heredan de la clase ***Exception***, así que para capturarlas hay que usar:

```
catch (Error $e){
    ...
}
```

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

```
catch (Throwable $e){
    ...
}
```

EXCEPCIONES ERROR PREDEFINIDAS

Algunas de las excepciones predefinidas se muestran en la tabla siguiente:

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 3: Excepción predefinida tipo *Arithmetic error* (con clase *Error*):

arithmeticError.php

```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 4: Excepción predefinida tipo *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

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.

Ejemplo típico de excepción definida por usuario:

customException.php

```php
class customException extends Exception {
    public function errorMessage() {
        // Mensaje de error
        $errorMsg =  '<b>'.$this->getMessage()
                .'</b><br>No es email válido';
        return $errorMsg;
    }
}
```

customException_usa.php

```php
include_once 'customException.php';

// Ponemos un email no válido para forzar la excepción
$email = "ejemplo@ejemplo/.com";


//$email = "ejemplo@ejemplo.com";

try {
    // Comprobar si el email es válido
    if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
        // Lanza una excepción si el email no es válido
        throw new customException($email);
    }
    else
    {
        echo $email."<br>Es email válido";
    }
}

catch (customException $e) {
    // Muestra el mensaje que hemos customizado en customException:
    echo $e->errorMessage();
}
```

Salida:

Caso email válido

ejemplo@ejemplo.com
Es email válido

Caso email No válido

ejemplo@ejemplo/.com
No es email válido

Ejemplo 5: Estructura Típica Recomendada para excepción definida por el usuario:

customExcepTipica.php

```php
class customExcepTipica 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<br>";
        // Mensaje de error
        $errorMsg =  '<b>'.$this->getMessage().'</b><br>No es válido';
        return $errorMsg;
    }
}
```

customExcepTipica_usa.php

```php
include_once 'customExcepTipica.php';

try {
    $valor="malo";
    //$valor="bueno";
    // Comprobar si el email es válido
    if($valor=="malo") {
        // Lanza una excepción
        throw new customExcepTipica($valor,0);
    }
    else
    {
        echo $valor."<br>Es Bueno";
    }
}

catch (customExcepTipica $e) {
    // Muestra el mensaje que hemos customizado
    // en customException:
    echo $e->customFunction();
}
```

Salida:

Caso bueno:

bueno
Es Bueno

Caso malo:

A Custom function for this type of exception
malo
No es válido

Ejemplo 6: De excepción de usuario para validar formulario

falta VALIDAR Y salida

excepcionValidation.php

```php
class exceptionValidation 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
include 'exceptionValidation.php';

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

        }

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

formBasico.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>
```

3- BIBLIOGRAFÍA

Desarrollo Web En entorno Servidor

Editorial Síntesis

Xabier Ganzábal García

isbn: 978-54-917183-6-9