

# Apuntes de



**Marcos Pérez Fernández**

## Sumario

Objetivos.....	3
Introducción.....	3
Ciclo de vida de una aplicación web.....	4
Sintaxis básica.....	4
Variables.....	5
Tipos de datos.....	6
Strings.....	6
Integers.....	7
Constantes.....	8
Operadores.....	8
Condicionales.....	9
Bucles.....	9
Funciones.....	10
Arrays.....	11
Variables superglobales.....	13
Validación de formularios.....	14
Ejemplo básico.....	15
Ejemplo de validación.....	15
include y require.....	18
Ficheros.....	20
Ejemplos.....	21
Subir ficheros mediante formularios.....	23
Ejemplos.....	23
Cookies.....	25
Sesiones.....	27
Filtros.....	30
Filtros avanzados.....	31
JSON.....	31
Programación orientada a objetos.....	33
Clases y objetos.....	33
Constructores y destructores de objetos.....	34
Modificadores de acceso.....	37
Herencia y operador de ámbito.....	37
Clases abstractas.....	38
Traits.....	40
Métodos y atributos estáticos.....	40
Bases de datos.....	41
Conexión a BBDD.....	42
Creación de una BD.....	43
Creación de tablas.....	44
Inserción de datos.....	46
Inserción de múltiples valores.....	48
Sentencias preparadas.....	50
Select.....	52
Borrado.....	54
Actualización.....	55

## Objetivos

- Entender el funcionamiento de la programación del lado del servidor.
- Conocer la sintaxis básica de PHP.
- Aprender a crear webs páginas dinámicas.
- Aplicar POO en con PHP en entornos web.
- Consultar bases de datos desde PHP.
- Entender el patrón MVC.

## Introducción

PHP (Personal Home Page, originalmente. Hoy en día Hypertext Preprocessor), es un lenguaje de programación **interpretado**, **abierto**, del **lado del servidor**, que permite crear paginas **web dinámicas**, es decir, que cambien su contenido en función de valores variables, ya sean proporcionados por el usuario o por el propio sistema.

Al ser un lenguaje de programación del lado de servidor, **PHP necesita un servicio web** funcionando para poder ejecutarse, además de un **servidor de bases de datos** en caso de que se quiera acceder a datos. La mejor manera de conseguir todos los servicios que necesitamos para aprender PHP es a través del paquete de software, **XAMPP**, el cual incorpora, entre otros, un servidor web Apache y el gestor de bases de datos MariaDB.

Una vez instalado XAMPP y habiendo iniciado los servicios necesarios, podremos ejecutar nuestros programas PHP en cualquier navegador web de nuestro sistema.

Para probar que XAMPP esta funcionando correctamente, introduciremos la URL **<http://localhost>**, en un navegador y deberemos obtener una respuesta de bienvenida parecida a esta:



## Welcome to XAMPP for Linux 7.4.4

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

Los archivos PHP que creemos deberán guardarse en la carpeta **htdocs**, dentro del directorio de instalación de XAMPP.

## Ciclo de vida de una aplicación web



La imagen anterior se interpreta de la siguiente manera:

1. Un usuario, desde su navegador (cliente), solicita una web a través de la URL, al servidor.
2. El servidor busca dicha web entre todas las que alberga, y si la encuentra y es una web con extensión .php, envía dicha web a procesar al módulo de PHP que deberá tener instalado.
3. El procesador de PHP, interpreta el contenido del archivo .php, el cual puede estar compuesto de cualquier elemento web, HTML, CSS o JavaScript, junto con

bloques de código PHP, los cuales serán renderizados, esto es, se sustituirá el código PHP por valores estáticos HTML, de forma que, finalmente se obtenga una página web estática, HTML.

4. Esta web HTML, sin código PHP, será enviada al usuario que la solicitó y será mostrada en el navegador.

Esta forma de trabajar es válida para cualquier sistema que pueda tener el usuario final, de ahí que PHP sea un lenguaje multiplataforma.

## Sintaxis básica

- El código PHP va entre las etiquetas `<?php` y `?>`
- La extensión de un archivo PHP por defecto es `.php`
- PHP es sensible a mayúsculas.
- Comentarios de código:
  - `#` y `//`: comentarios de línea.
  - `/* */`: comentarios de bloque.
- En un archivo PHP veremos etiquetas HTML combinadas con código PHP.
- El siguiente ejemplo es un archivo llamado `prueba.php`, almacenado en el directorio `htdocs` de `xampp` y ejecutado en un navegador web a través de la URL <http://localhost/prueba.php>

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

<h1>Prueba inicial PHP</h1>

<?php
echo "Hola!";
?>

</body>
</html>
```

# Prueba inicial PHP

Hola!

## Variables

- Las variables se crean en el momento de su declaración. No existe un comando para su declaración explícita. Por ello, se dice que PHP es un lenguaje **débilmente tipado**.
- Si se trata de variables tipo cadena, su valor irá entre comillas dobles, si son numéricas, se les asigna un valor numérico directamente.
- Las variables en PHP comienzan por el símbolo '\$' y van seguidas del nombre de la variable, el cual podrá empezar por un guion bajo o una letra. A partir de ahí, podrá usarse cualquier combinación de letras, números y guiones bajos.
- el comando **echo** y **print** permiten mostrar el valor de una variable e información en general. La diferencia principal es que **echo** no devuelve ningún valor y **print** devuelve 1, por lo que puede ser usado en expresiones. Por ello, **echo** es más rápido. Ambos se pueden usar con o sin paréntesis.

<pre>&lt;?php \$x = 5; \$y = 7; \$suma = \$x + \$y; echo "&lt;h1&gt;Uso de echo&lt;/h1&gt;"; echo "La suma es: " . \$suma; echo ("&lt;br&gt; &lt;b&gt;Otra línea&lt;/b&gt; &lt;br&gt;"); echo "hola", "y", "adiós";  print "&lt;h1&gt;Uso de print&lt;/h1&gt;"; print "La suma es: " . \$suma; \$v = print ("&lt;br&gt; &lt;b&gt;Otra línea&lt;/b&gt; &lt;br&gt;"); echo \$v; ?&gt;</pre>	<h3>Uso de echo</h3> <p>La suma es: 12 Otra línea holayadiós</p> <h3>Uso de print</h3> <p>La suma es: 12 Otra línea 1</p>
---	---

## Tipos de datos

- PHP puede inferir y trabajar con los siguientes tipos de datos: cadenas, enteros, flotantes, booleanos, arrays, objetos y null.

<pre> &lt;?php \$c = "cadena"; var_dump(\$c); echo "&lt;br&gt;"; \$i = 7; var_dump(\$i); echo "&lt;br&gt;"; \$f = 10.555; var_dump(\$f); echo "&lt;br&gt;"; \$b = true; var_dump(\$b); echo "&lt;br&gt;"; \$a = array(1,2,3); var_dump(\$a); echo "&lt;br&gt;";  class clase {} \$o = new clase(); var_dump(\$o); echo "&lt;br&gt;";  \$n = null; var_dump(\$n); echo "&lt;br&gt;"; ?&gt; </pre>	<pre> string(6) "cadena" int(7) float(10.555) bool(true) array(3) { [0]=&gt; int(1) [1]=&gt; int(2) [2]=&gt; int(3) } object(clase)#1 (0) { } NULL </pre>
--	---

- La función **var\_dump(\$exp)**, nos muestra el tipo y valor de una expresión.

## Strings

Algunas de las muchas funciones para trabajar con strings en PHP:

- **strlen()**: devuelve la longitud de una cadena.
- **str\_word\_count()**: devuelve el numero de palabras de una cadena.
- **strrev()**: devuelve la cadena invertida.
- **strpos()**: busca un texto en una cadena y devuelve su posición.
- **str\_replace()**: busca un texto en una cadena y lo reemplaza por otro.

<pre> &lt;?php \$c = "Esto es una cadena";  echo "Longitud: " . strlen(\$c) . "&lt;br&gt;"; echo "Num Palabras: " . str_word_count(\$c) . "&lt;br&gt;"; echo "Al revés: " . strrev(\$c) . "&lt;br&gt;"; echo "Posición de 'una': " . strpos(\$c,"una") . "&lt;br&gt;"; echo "Reemplazo: " . str_replace("cadena","string",\$c); </pre>	<pre> Longitud: 18 Num Palabras: 4 Al revés: anedac anu se otsE Posición de 'una': 8 Reemplazo: Esto es una string </pre>
--	---

```
?>
```

## Integers

- PHP tiene detección automática de tipos, por lo que podremos usar una misma variable para alojar diferentes tipos de datos según nos convenga, aunque puede que no sea recomendable.
- Funciones relacionadas:
  - `is_int()`: permite comprobar si una variable almacena un entero.
  - `is_float()`: permite comprobar si una variable almacena un entero.
  - `is_finite()`: permite comprobar si el valor es menor que `PHP_FLOAT_MAX`.
  - `is_infinite()`: permite comprobar si el valor es mayor que `PHP_FLOAT_MAX`.
  - `is_nan()`: permite comprobar si el valor no es un número.
  - `is_numeric()`: permite comprobar si el valor es numérico incluso estando entre comillas.
- Para realizar un casting de tipos con enteros podremos usar `(int)`, `(integer)`.  
Para obtener la parte entera de un valor usaremos `intval()`.

```
<?php
echo var_dump(is_int((int) "12345")) . "<br>";
echo var_dump(is_float(987.114)) . "<br>";
echo PHP_FLOAT_MAX . "<br>";
echo var_dump(is_infinite(1.4e1000)) . "<br>";
echo var_dump(is_nan(0/0)) . "<br>";
echo var_dump(is_numeric("55")) . "<br>";
?>
```

```
bool(true)
bool(true)
1.7976931348623E+308
bool(true)

Warning: Division by zero
bool(true)
bool(true)
```

## Constantes

- `define()`: permite crear constantes. Son globales. El último parámetro indica si el nombre de la constante es sensible a mayúsculas.



```
<?php
define ("HOLA", "hola", true);
echo HOLA;
?>
```

hola

## Operadores

- **Aritméticos:** +, -, \*, /, %, \*\*
- **Asignación:** =, +=, -=, \*=, /=
- **Comparación:** ==, ===, !=, !==, >, <, <=, >=, <=>. Este último operador devuelve un entero menor o igual, igual o mayor que 0, dependiendo de si el primer valor es menor o igual, igual o mayor o igual que el segundo valor.
- **Incremento/decremento:** ++, --. Colocados antes que la variable, primero incrementan o decrementan y luego devuelven el valor, mientras que colocados después, primero devuelven el valor y luego incrementan o decrementan la variable.
- **Lógicos:** and, or, not, xor, &&, ||, !
- **De cadena:** ., .= Este último operador concatena el contenido de una variable a otra.
- **Asignación condicional:** ?:, ??

```
<?php
$x = 100;
$y = "100";
var_dump($x == $y); echo "<br>";
var_dump($x === $y); echo "<br>"; // falla el tipo

$x = 5;
$y = 10;
echo ($x <=> $y) . "<br>"; // devuelve -1

$x = 10;
$y = 10;
echo ($x <=> $y) . "<br>"; // devuelve 0

$x = 15;
$y = 10;
echo ($x <=> $y) . "<br>"; // devuelve +1

$i = ($x > $y) ? $x : $y; echo $i . "<br>";

// si x no existe o es null asigna $y
```

```
bool(true)
bool(false)
-1
0
1
15
10
```

```
$i = $xx ?? $y; echo $i . "<br>";  
?>
```

## Condicionales

```
<?php  
$h = date("H");  
  
if ($h > 8 && $h < 12) {  
    echo "good morning";  
} else if ($h >= 12 && $h < 16) {  
    echo "good afternoon";  
} else {  
    echo "good evening";  
}  
?>
```

```
<?php  
$color = "azul";  
  
switch ($color) {  
    case "rojo":  
        echo "red"; break;  
    case "azul":  
        echo "blue"; break;  
    case "amarillo":  
        echo "yellow"; break;  
    default:  
        echo "no conozco ese color";  
}  
?>
```

## Bucles

```
<?php  
$c = 0;  
while ($c < 5) {  
    echo $c++ . " ";  
}  
?>
```

```
<?php  
$c = 0;  
do {  
    echo $c++ . " ";  
}while ($c < 5);  
?>
```

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

- **foreach**: este tipo de bucle merece mención especial ya que está destinado únicamente a iterar arrays de forma asociativa, es decir, asociando a cada elemento del bucle a una variable en cada iteración. Sintaxis:

```
foreach ($array as $valor) {  
    CODIGO_A_EJECUTAR;  
}
```

<pre>&lt;?php \$ciudades = array( "Madrid", "Ferrol", "Valencia", "Vigo");  foreach (\$ciudades as \$ciudad){     echo \$ciudad . " "; } ?&gt;</pre>	<pre>&lt;?php \$ciudades = array( "Madrid" =&gt; "Centro",                   "Ferrol" =&gt; "Rías Altas",                   "Valencia" =&gt; "Levante",                   "Vigo" =&gt; "Rias Baixas");  foreach (\$ciudades as \$ciudad =&gt; \$zona){     echo \$ciudad . ", zona: " . \$zona .     "&lt;br&gt;"; } ?&gt;</pre>
--	--

## Funciones

- **strict\_types**: al ser PHP un lenguaje débilmente tipado, puede suceder que los parámetros de las funciones reciban datos que no sean del tipo que se espera y que la función devuelva resultados no esperados. Para forzar los tipos de datos, se puede usar **declare(strict\_types = 1)** y así, evitar problemas no previstos.

<pre>&lt;?php function sumar(int \$a, int \$b) {     return \$a + \$b; } echo sumar(5, "5 días"); ?&gt;</pre>	<p><b>Notice:</b> A non well formed numeric value</p> <p>10</p>
<pre>&lt;?php declare(strict_types = 1); function sumar(int \$a, int \$b) {     return \$a + \$b; } echo sumar(5, "5 días"); ?&gt;</pre>	<p><b>Fatal error:</b> Uncaught TypeError: Argument 2 passed</p> <p>trace: #0 /opt/lampp/htdocs/prueba02.php(6): sumar(</p>

- Se puede asignar a un parámetro un valor por defecto, que será usado en caso de que cuando se llame a la función, no se le pase ningún valor a dicho parámetro.
- Podemos forzar el tipo devuelto usando, junto con **strict**, el operador **':'**, después de la declaración de los argumentos de la función

```
<?php
declare(strict_types = 1);
function sumar(int $a = 1, int $b = 1) : int {
    echo "a = " . $a . " b = " . $b . "<br>";
    return $a + $b;
}
echo "suma = " . sumar() . "<br>";
echo "suma = " . sumar(5) . "<br>";
?>
```

```
a = 1 b = 1
suma = 2
a = 5 b = 1
suma = 6
```

## Arrays

- Tipos de arrays:
  - **Indexados**: se usa un índice numérico para acceder a los elementos del array.
  - **Asociativos**: se usa un nombre para acceder a los elementos del array.
- **array()**: se usa esta función para crear arrays.
- **count()**: devuelve el número de elementos de un array.

```
<?php
$elem = array("elem01", "elem02", "elem03");
$tam = count($elem);

for($i = 0; $i < $tam; $i++){
    echo $elem[$i] . " ";
}
?>
```

```
elem01 elem02 elem03
```

- En arrays asociativos en lugar de índices numéricos se usaran nombres

```
<?php
$lugares = array("Vigo" => "Pontevedra",
                "Ferrol" => "A Coruña",
                "Ribadeo" => "Lugo");

echo "Vigo está en: " . $lugares['Vigo'] . "<br>";
echo "Ferrol está en: " . $lugares['Ferrol'] . "<br>";
echo "Ribadeo está en: " . $lugares['Ribadeo'] . "<br>";

$lugares['Cartagena'] = "Murcia";

echo "Cartagena está en: " . $lugares['Cartagena'] .
"<br>";
?>
```

```
Vigo está en: Pontevedra
Ferrol está en: A Coruña
Ribadeo está en: Lugo
Cartagena está en: Murcia
```

- El bucle **foreach** está pensado para recorrer los arrays asociativos. En el siguiente ejemplo, el array **\$lugares** se recorrerá usando como clave **\$ciudades** y como valor **\$provincias**.

```
<?php
$lugares = array( "Vigo" => "Pontevedra",
                  "Ferrol" => "A Coruña",
                  "Ribadeo" => "Lugo");

$lugares[ 'Cartagena' ] = "Murcia";

foreach ( $lugares as $ciudad => $provincia ) {
    echo $ciudad . " esta en " . $provincia . "<br>";
}
?>
```

Vigo esta en Pontevedra  
 Ferrol esta en A Coruña  
 Ribadeo esta en Lugo  
 Cartagena esta en Murcia

- Podemos usar más de una dimensión en arrays

```
<?php
$coches = array (
    array( "Seat", 20),
    array( "BMW", 14),
    array( "Fiat", 7)
);
$filas = count($coches);
$cols = count($coches[0]);

for ($i = 0; $i < $filas; $i++){
    echo "<p><b>Fila $i</b></p>";
    echo "<ul>";
    for ($j = 0; $j < $cols; $j++) {
        echo "<li>" . $coches[$i][$j] . "</li>";
    }
    echo "</ul>";
}
?>
```

#### Fila 0

- Seat
- 20

#### Fila 1

- BMW
- 14

#### Fila 2

- Fiat
- 7

- Funciones para ordenar arrays:
  - **sort()** : ordena el array ascendentemente.
  - **rsort()** : ordena el array descendientemente.
  - **asort()** : ordena el array asociativo ascendentemente, en función del valor.
  - **arsort()** : ordena el array asociativo descendientemente, en función del valor.

- **ksort()**: ordena el array asociativo ascendentemente, en función de la clave.
- **krsort()**: ordena el array asociativo descendientemente, en función de la clave.

```
<?php
$personas = array ( "Pepe" => 34,
                    "Ana" => 40,
                    "Miguel" => 25);
asort($personas);

echo "<h2>Orden por valor</h2>";
foreach($personas as $nombre => $edad){
    echo "Clave: $nombre; Valor: $edad <br>";
}
echo "<h2>Orden por clave</h2>";
ksort($personas);
foreach($personas as $nombre => $edad){
    echo "Clave: $nombre; Valor: $edad <br>";
}
?>
```

## Orden por valor

Clave: Miguel; Valor: 25  
Clave: Pepe; Valor: 34  
Clave: Ana; Valor: 40

## Orden por clave

Clave: Ana; Valor: 40  
Clave: Miguel; Valor: 25  
Clave: Pepe; Valor: 34

## Variables superglobales

- Las variables superglobales son **variables predefinidas** de PHP, las cuales están siempre disponibles, sin importar de qué bloque de código se trate. En este punto las describiremos y, a medida que vayamos avanzando las usaremos en su contexto adecuado:
  - **\$GLOBALS**: PHP almacena las variables globales en este array asociativo, de forma que puedan ser accedidas desde cualquier ámbito usando el nombre de la variable global como clave.
  - **\$\_SERVER**: es un array asociativo que almacena información sobre cabeceras, rutas y ubicaciones del servidor.
  - **\$\_REQUEST**: es array asociativo que recoge datos después de haber enviado un formulario HTML.
  - **\$\_POST**: array asociativo que permite recoger los datos de un formulario.

- **`$_GET`**: array asociativo que permite recoger los datos de un formulario a través de la URL.
- **`$_FILES`**: array asociativo de elementos subidos mediante el método **`post`**.
- **`$_ENV`**: array asociativo de variables pasadas al código mediante el método del entorno.
- **`$_COOKIE`**: array asociativo con las cookies pasadas al código.
- **`$_SESSION`**: array asociativo con las variables del código actual.

## Validación de formularios

Antes de ver un ejemplo sobre cómo validar formularios, vamos a mostrar los elementos más importantes que entran en juego:

- **Archivo HTML con el formulario**: este es el archivo que contiene todos los elementos del formulario. Destacamos los siguientes elementos clave:
  - **atributo `action`**: indica el fichero PHP al que se le enviarán los datos del formulario para ser procesados
  - **atributo `method`**: indica la forma en la que serán pasados los datos del formulario al archivo PHP que los procesará. Las formas posibles son "**`GET`**" y "**`POST`**".
  - **botón de envío**: es el botón que ejecuta la acción de enviar los datos del formulario al archivo PHP que los procesa.
- **Archivo PHP de validación**: es el archivo indicado en el atributo **`action`** del formulario. Se deberán llevar a cabo las siguientes tareas:
  - **`$_SERVER["REQUEST_METHOD"]`**: será necesario comprobar antes de acceder a los datos, la forma en la que estos han sido enviados. Accediendo a esta variable sabremos si ha sido mediante **`GET`** o **`POST`**.

- Ahora, hay que acceder a cada una de las variables recibidas a través de `$_GET` o `$_POST` y realizar y ejecutar sobre ellas las siguientes funciones (si son necesarias):
  - **htmlspecialchars**: aporta un nivel de seguridad ya convierte etiquetas HTML en entidades especiales HTML, de forma que no si se intentan insertar a través de formularios scripts, a traves de la etiqueta HTML `<script>`, estos no funcionarán ya que esta se transformará en `&ls;scrip&gt`
  - **trim**: elimina todos los caracteres innecesarios como espacios extra, tabulaciones, etc.
  - **stripslashes**: elimina los backslashes `'\'`.
- **Array \$errores**: los errores que se puedan detectar durante la fase de validación se almacenarán en este array, de forma que puedan mostrarse una vez ejecutado todo el código de validación.

## Ejemplo básico

Fichero uno.php

```
<html>
<body>
<form action="dos.php" method="get">
  Nombre: <input type="text"
name="nombre" />
  <br />
  Edad: <input type="text"
name="edad" />
  <br />
  <input type="submit"
value="aceptar" />
</form>
</body>
</html>
```

Fichero dos.php

```
<html>
<body>
<?php
echo $_GET["nombre"] . " ".
$_GET["edad"];
?>
</body>
</html>
```

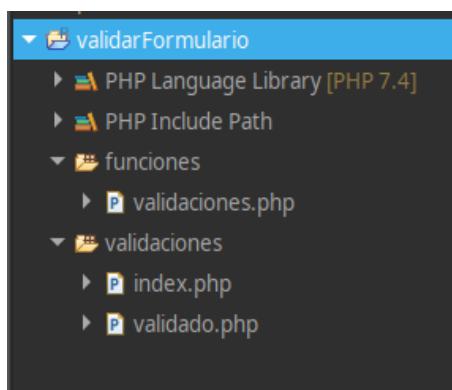


Nombre: <input type="text" value="Pepe"/> Edad: <input type="text" value="25"/> <input type="button" value="aceptar"/>	Pepe 25  <a href="http://localhost/validar01/dos.php?nombre=Pepe&amp;edad=25">http://localhost/validar01/dos.php?nombre=Pepe&amp;edad=25</a>
--	--

- Los datos recogidos en el formulario del archivo **uno.php** se pasan por **GET** a **dos.php**. No se realiza ninguna validación.

## Ejemplo de validación

A continuación se muestra el código de una aplicación web formada que valida un formulario. La aplicación tiene la siguiente estructura:



ARCHIVO index.php CON FORMULARIO

```
<!DOCTYPE>
<html>
<head>
  <title>Validar un formulario</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
  <form action="index.php" method="post">
    <label>Nombre </label> <br />
    <input type="text" name="nombre" /> <br />
    <label>Edad </label>
    <input type="text" name="edad" size=3 /> <br />
    <label>E-mail </label>
    <input type="text" name="email" /> <br />
    <input type="submit" value="Enviar" />
  </form>
```

```

<?php
    require_once '../funciones/validaciones.php';

    $errs = (array)$GLOBALS["errores"];
    echo "<ul>";
    if ($errs)
        foreach($errs as $error){
            echo "<li>$error</li>";
        }
    echo "</ul>";
?>

</body>

</html>

```

ARCHIVO CON LA LÓGICA DE VALIDACIÓN

```

<?php

$errores = array();

function validaRequerido($valor){
    if(trim($valor) == ''){
        return false;
    }else{
        return true;
    }
}

function validarEntero($valor, $opciones=null){
    if(filter_var($valor, FILTER_VALIDATE_INT, $opciones) === FALSE){
        return false;
    }else{
        return true;
    }
}

function validaEmail($valor){
    if(filter_var($valor, FILTER_VALIDATE_EMAIL) === FALSE){
        return false;
    }else{
        return true;
    }
}

if ($_SERVER['REQUEST_METHOD'] == 'POST'){

    $nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
    $edad = isset($_POST['edad']) ? $_POST['edad'] : null;
    $email = isset($_POST['email']) ? $_POST['email'] : null;

    if (!validaRequerido($nombre)){
        array_push($errores, 'El campo nombre es incorrecto');
    }

    $opciones_edad = array(

```

```

        'options' => array(
            'min_range' => 3,
            'max_range' => 130
        )
    );

    if(!validarEntero($edad,$opciones_edad)) {
        array_push($errores, 'El campo edad es incorrecto');
    }

    if(!validaEmail($email)) {
        array_push($errores, 'El campo email es incorrecto');
    }

    if(!$errores){
        header('location: validado.php');
        exit;
    }
}
?>

```

ARCHIVO CON MENSAJE DE VALIDACIÓN CORRECTA

```

<!DOCTYPE>
<html>
<head>
<title> Formulario </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
    <strong> Sus datos han sido enviados correctamente </strong>
    <?php header('refresh:5; url=index.php') ?>
</body>
</html>

```

## include y require

- Ambos permiten insertar todo el código de un fichero en el punto donde se especifiquen.
- En caso de fallo, **include** emite un **warning** y permite seguir al programa mientras que **require**, emite un **error** y para la ejecución.
- Se puede usar **include\_once** y **require\_once** de forma que si el fichero ya ha sido incluido en un momento anterior, este no se volverá a incluir.

ARCHIVO header.php

```
<?php
require 'variables.php';
echo 'Hoy es ' . $fecha_hoy . " " . $fecha_dia;
echo '<h1>Bienvenidos!!!</h1>';
?>
```

ARCHIVO nav.php

```
<?php
echo '
<a href="">Home</a> -
<a href="">Sección 1</a> -
<a href="">Sección 1</a> -
<a href="">Sección 1</a> -
<a href="">About</a>';
?>
```

ARCHIVO main.php

```
<?php
echo '
<p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Vestibulum sit amet sodales nisl.
Vestibulum ante ipsum primis in faucibus orci
luctus et ultrices posuere cubilia curae;
Integer in neque ac enim aliquam rutrum eu nec enim.</p>';
?>
```

ARCHIVO footer.php

```
<?php
require_once 'variables.php';
echo '<p>Copyright &copy; 1999 - ' . $año . ' miempresa.com';
?>
```

ARCHIVO variables.php

```
<?php
$fecha_hoy = date("d/m/Y");
$fecha_dia = date("l");
$año = date("Y");
?>
```

ARCHIVO index.php

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style=max-width:500px>
<?php require_once 'variables.php' ?>
<header><?php include 'header.php'; ?></header>
<hr />
<nav><?php include 'nav.php'; ?></nav>
```

```
<hr />
<main>
<article><?php require 'main.php'; ?></article>
<article><?php require_once 'main.php'; ?></article>
</main>
<hr />
<footer><?php require 'footer.php' ?></footer>
</div>
</body>
</html>
```

Hoy es 04/05/2020 Monday

## Bienvenidos!!!

---

[Home>](#) - [Sección 1](#) - [Sección 1](#) - [Sección 1](#) - [About](#)

---

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum sit amet sodales nisl. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Integer in neque ac enim aliquam rutrum eu nec enim.

---

Copyright © 1999 - 2020 miempresa.com;

## Ficheros

- **readfile()**: lectura directa de ficheros. Devuelve en número de bytes leídos.
- **fopen()**: apertura de ficheros. Más potente y recomendada que readfile(). Modos de apertura:
  - **r**: apertura de solo lectura. Se posiciona al principio del fichero.
  - **w**: apertura de solo escritura. Si el fichero existe, borra su contenido. Se posiciona al principio del fichero.
  - **a**: apertura para añadir. Abre para escritura no destructiva. Se posiciona al final del fichero.

- **x**: apertura de solo escritura. Si el fichero existe, devuelve FALSE y un error.
  - **r+**: apertura para lectura/escritura. Se posiciona al principio del fichero.
  - **w+**: apertura para lectura/escritura. Si existe el fichero lo borra y si no, lo crea. Se posiciona al principio.
  - **a+**: apertura para lectura/escritura. Si existe el fichero no lo borra y si no existe, lo crea. Se posiciona al final del fichero.
  - **x+**: crea un fichero para lectura/ escritura. Devuelve FALSE y un error si el fichero existe.
- **fread()**: lee desde un fichero abierto. Se le pasa el tamaño máximo de bytes que se leerán.
  - **fclose()**: cierra un fichero abierto.
  - **fgets()**: lee una línea de un fichero.
  - **feof()**: comprueba si se ha alcanzado el fin de fichero (EOF).
  - **fgetc()**: lee un carácter de un fichero.
  - **fwrite()**: escribe en un fichero.

## Ejemplos

```
<!DOCTYPE html>
<html>
<body>
<?php
echo readfile("fichero.txt");
?>
</body>
</html>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper. Morbi dui mauris, sollicitudin non diam non, viverra sollicitudin mi. Pellentesque eleifend mauris a suscipit egestas. Phasellus consectetur est eu mattis malesuada. Nunc purus lacus, gravida a convallis ut, gravida sed enim. Morbi vel nunc sed est tempus lobortis vel eu sem. Vivamus convallis nulla et tempor dictum. Curabitur eget ipsum turpis. Sed ut fringilla leo, ut sodales ipsum. Curabitur pharetra neque eu maximus sagittis. **536**

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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper. Morbi dui mauris, sollicitudin non diam

```

<?php
$mi_fichero = fopen("fichero.txt", "r")
or die("No se puede abrir el
fichero!!!");
echo "<pre>" .
fread($mi_fichero, filesize("fichero.tx
t")) . "</pre>";
fclose($mi_fichero);
?>

</body>
</html>

```

non, viverra sollicitudin mi. Pellentesque eleifend mauris a suscipit egestas. Phasellus consectetur est eu mattis malesuada. Nunc purus lacus, gravida a convallis ut, gravida sed enim.

Morbi vel nunc sed est tempus lobortis vel eu sem. Vivamus convallis nulla et tempor dictum. Curabitur eget ipsum turpis. Sed ut fringilla leo, ut sodales ipsum. Curabitur pharetra neque eu maximus sagittis.

```

<!DOCTYPE html>
<html>
<body>

<?php
$mi_fichero = fopen("fichero.txt", "r")
or die("No se puede abrir el
fichero!!!");
echo "<pre>" .
fgets($mi_fichero) . "</pre>";
fclose($mi_fichero);
?>

</body>
</html>

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper.

```

<!DOCTYPE html>
<html>
<body>

<?php
$mi_fichero = fopen("fichero.txt", "r")
or die("No se puede abrir el
fichero!!!");

while(!feof($mi_fichero)) {
    echo fgets($mi_fichero);
}
fclose($mi_fichero);
?>

</body>
</html>

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper. Morbi dui mauris, sollicitudin non diam non, viverra sollicitudin mi. Pellentesque eleifend mauris a suscipit egestas. Phasellus consectetur est eu mattis malesuada. Nunc purus lacus, gravida a convallis ut, gravida sed enim. Morbi vel nunc sed est tempus lobortis vel eu sem. Vivamus convallis nulla et tempor dictum. Curabitur eget ipsum turpis. Sed ut fringilla leo, ut sodales ipsum. Curabitur pharetra neque eu maximus sagittis.

<pre> &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;?php \$mi_fichero = fopen("fichero.txt", "r") or die("No se puede abrir el fichero!!!");  while(!feof(\$mi_fichero)) {     echo fgetc(\$mi_fichero); } fclose(\$mi_fichero); ?&gt;  &lt;/body&gt; &lt;/html&gt; </pre>	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper. Morbi dui mauris, sollicitudin non diam non, viverra sollicitudin mi. Pellentesque eleifend mauris a suscipit egestas. Phasellus consectetur est eu mattis malesuada. Nunc purus lacus, gravida a convallis ut, gravida sed enim. Morbi vel nunc sed est tempus lobortis vel eu sem. Vivamus convallis nulla et tempor dictum. Curabitur eget ipsum turpis. Sed ut fringilla leo, ut sodales ipsum. Curabitur pharetra neque eu maximus sagittis.</p>
---	--

<pre> &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;?php \$mi_fichero = fopen("fichero.txt", "a") or die("No se puede abrir el fichero!!!"); \$nuevo_texto = "ESTE TEXTO SE AÑADIRA AL FICHERO"; fwrite(\$mi_fichero, \$nuevo_texto); fclose(\$mi_fichero);  echo readfile("fichero.txt"); ?&gt;  &lt;/body&gt; &lt;/html&gt; </pre>	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ullamcorper scelerisque semper. Morbi dui mauris, sollicitudin non diam non, viverra sollicitudin mi. Pellentesque eleifend mauris a suscipit egestas. Phasellus consectetur est eu mattis malesuada. Nunc purus lacus, gravida a convallis ut, gravida sed enim. Morbi vel nunc sed est tempus lobortis vel eu sem. Vivamus convallis nulla et tempor dictum. Curabitur eget ipsum turpis. Sed ut fringilla leo, ut sodales ipsum. Curabitur pharetra neque eu maximus sagittis. ESTE TEXTO SE AÑADIRA AL FICHERO569</p>
---	--

## Subir ficheros mediante formularios

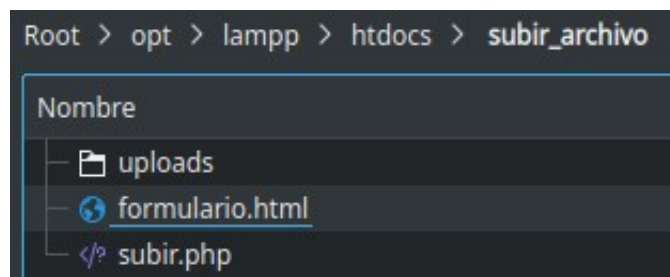
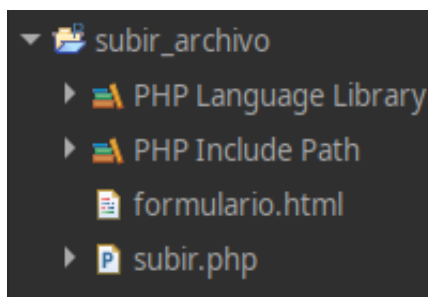
- Para poder subir ficheros, el archivo de configuración **php.ini** deberá tener la directiva **file\_uploads=on**.
- El formulario que se use para subir archivos deberá tener configurado el atributo **enctype="multipart/form-data"**.
- El método de transferencia será **POST**.



- Habrá un `<input>` con el atributo `type="file"`, que abrirá un buscador de archivos.
- Además del archivo del formulario, habrá otro archivo que será el que lo procese.

## Ejemplos

- Estructura del proyecto y de directorios:



- Archivos:

### ARCHIVO `formulario.php`

```
<!DOCTYPE>
<html>
<body>
<form action="subir.php" method="post" enctype="multipart/form-data">
  Selecciona una imagen:<br />
  <input type="file" name="archivo_a_subir" /><br />
  <input type="submit" value="Enviar" name="enviar" />
</form>
</body>
</html>
```

### ARCHIVO `subir.php`

```
<?php
$directorio_destino = "uploads/";
$ruta_absoluta = $directorio_destino . basename($_FILES["archivo_a_subir"]
["name"]);
$tipo_imagen = strtolower(pathinfo($ruta_absoluta, PATHINFO_EXTENSION));
$procesoOK = true;

if (file_exists($ruta_absoluta)) {
    echo "El archivo ya existe <br />";
    $procesoOK = false;
}

if ($_FILES["archivo_a_subir"]["size"] > 5000000) {
```

1- Se crea un directorio para los archivos  
2- Se guarda la ruta absoluta y extensión del archivo.  
3- Se crea una variable de comprobación

Se realizan todas las comprobaciones oportunas

```

    echo "El archivo es demasiado grande <br />";
    $procesoOK = false;
}

if ($tipo_imagen != "jpg" && $tipo_imagen != "png") {
    echo "Solo se permiten archivos JPG y PNG <br />";
    $procesoOK = false;
}

if ($procesoOK == false) {
    echo "El archivo no se ha subido.";
} else {
    if (move_uploaded_file($_FILES["archivo_a_subir"]["tmp_name"],
    $ruta_absoluta)) {
        echo "El archivo <b>" . basename($_FILES["archivo_a_subir"]["name"]).
        "</b> se ha subido";
    } else {
        echo "El archivo no se ha subido.";
    }
}
?>

```

Si todo hay ido bien, se sube el archivo

Selecciona una imagen: <input type="text" value="Seleccionar archivo"/> 3a Ev Dept Informática.pdf <input type="button" value="Enviar"/>	Solo se permiten archivos JPG y PNG El archivo no se ha subido.
Selecciona una imagen: <input type="text" value="Seleccionar archivo"/> foto01.png <input type="button" value="Enviar"/>	El archivo <b>foto01.png</b> se ha subido
Selecciona una imagen: <input type="text" value="Seleccionar archivo"/> foto01.png <input type="button" value="Enviar"/>	El archivo ya existe El archivo no se ha subido.

## Cookies

- Las cookies son pequeños archivos que el servidor guarda en el equipo del cliente, de forma que cada vez que el cliente pide la misma página al servidor, este puede solicitar la cookie y obtener la información que hay en ella, habitualmente información sobre alguna preferencia del cliente e identificación.

- **setcookie()**: permite crear, modificar y leer una cookie. Esta función irá siempre antes de la etiqueta **<html>**.
- Por orden, los parámetros que usaremos son:
  - **Nombre** de la cookie.
  - **Valor** que almacena.
  - **Tiempo** de vida.
  - **Ámbito** para el cual estará disponible. '/' representa a todo el sitio web.
- Para borrar una cookie basta con configurarle a **setcookie()** un tiempo de vida pasado.
- Podemos saber si el navegador tiene las cookies activadas si cuando al intentar crear una cookie el contador de la variable **\$\_COOKIE** es mayor que 0;

```
<?php
$nombre = "usuario";
$valor = "Pepe Perez";
$duracion = time() + (86400 * 30); //86400 = 1 dia
$ambito = "/";

// CREACIÓN DE LA COOKIE
setcookie($nombre, $valor, $duracion, $ambito);
if (count($_COOKIE) > 0) {
    echo "Las cookies están activadas<br /><br />";
} else {
    echo "Las cookies NO están activadas<br /><br />";
}
?>
<html>
<body>
    <?php
        // LECTURA DEL VALOR DE LA COOKIE
        if (!isset($_COOKIE[$nombre])) {
            echo "La cookie " . $nombre . " no está creada";
        } else {
            echo "La cookie " . $nombre . " está creada" . "<br />";
            echo "Su valor es: " . $_COOKIE[$nombre];
        }

        // MODIFICACION DE LA COOKIE
        echo "<br /><br />Modificamos la cookie $nombre a un nuevo valor...<br />";
        $valor = "Manuel Rodriguez";
```

```

    setcookie($nombre, $valor, $duracion, $ambito);
    echo "La cookie " . $nombre . " tiene el valor: " . $_COOKIE[$nombre];

    // BORRRADO DE LA COOKIE
    echo "<br /><br />Borramos la cookie $nombre...<br />";
    setcookie($nombre, $valor, time() - 3600, $ambito);
    echo "La cookie " . $nombre . " tiene el valor: " . $_COOKIE[$nombre];
    ?>

</body>
</html>

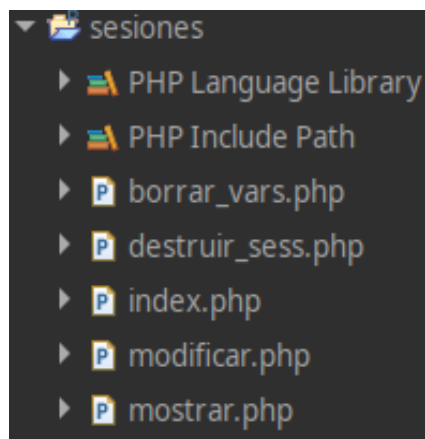
```

- Para ver el funcionamiento del ejemplo anterior habrá que ir realizando varias ejecuciones del código, comentando adecuadamente en cada una de ellas, las líneas donde aparezca **setcookies**, para ver como se crea, se modifica y se borra la cookie con la que se está trabajando.

## Sesiones

- La sesión es una forma de almacenar información en variables que serán usadas entre diferentes páginas del sitio web.
- Esta información se guarda en el servidor.
- Debido a que el protocolo HTTP no mantiene el estado de una sesión de usuario, es decir, al cambiar de una página a otra, HTTP no sabe de qué página se viene, se necesita una forma para poder mantener esta información.
- La sesión, permite guardar información de cualquier tipo, ya sea preferencias de usuario, nombre de usuario, etc. entre múltiples páginas y no perderlas hasta que el usuario cierra el navegador.
- El servidor puede mantener información de sesión para cualquier número de usuarios que se conecten al sitio.

- `session_start()`: crea la sesión. Irá siempre antes de la zona HTML del documento y estará presente en todas las páginas que formen el sitio web y necesiten usar variables de sesión.
- `$_SESSION`: almacena las variables de sesión.
- `session_unset()`: elimina todas las variables de sesión almacenadas en `$_SESSION`.
- `session_destroy()`: elimina la sesión.



index.php

```
<?php
session_start();

$_SESSION["nombre"] = "Pepe";
$_SESSION["edad"] = 25;
echo "Variables de sesión, creadas!<br />";
?>

</body>
<a href="mostrar.php">Mostar variables en otra pagina</a>
</html>
```

mostrar.php

```

<?php
session_start();
?>

<html>
<body>

<?php
echo "Nombre: " . $_SESSION["nombre"] . "<br>";
echo "Edad: " . $_SESSION["edad"] . "<br>";
?>

</body>
<a href="modificar.php">Modificar los valores....</a><br />
<a href="borrar_vars.php">Borrar variables de sesion....</a><br />
<a href="destruir_sess.php">Destruir la sesion....</a>
</html>

```

#### modificar.php

```

<?php
session_start();
?>

<html>
<body>

<?php
$_SESSION["nombre"] = "Manuel";
$_SESSION["edad"] = 40;
echo "Variables de sesión, modificadas<br />";
?>

</body>
<a href="mostrar.php">Mostar variables en otra pagina</a>
</html>

```

#### borrar\_vars.php

```

<?php
session_start();
?>

<html>
<body>

<?php
session_unset();
echo "Variables de sesión, modificadas<br />";
?>

</body>

```

```
<a href="mostrar.php">Mostar variables en otra pagina</a>
</html>
```

destruir\_sess.php

```
<?php
session_start();
?>

<html>
<body>

<?php
session_destroy();
echo "Sesion destruida<br />";
?>

</body>
<a href="mostrar.php">Mostar variables en otra pagina</a>
</html>
```

## Filtros

- Los **filtros** facilitan la comprobación de la entrada de usuario. Permiten verificar que los datos están en el formato correcto (**validate**) y eliminar cualquier carácter ilegal de los mismos (**sanitize**).
- **filter\_var()**: permite validar y sanear los datos. Un parámetro es la variable a comprobar y otro el filtro que se va a usar.

```
<?php
$cadena = "<h1><i>Holaaaa</i></h1>";
$cadena= filter_var($cadena, FILTER_SANITIZE_STRING);
echo $cadena;

$entero = 0;
if ( filter_var($entero, FILTER_VALIDATE_INT) === 0 ||
    filter_var($entero, FILTER_VALIDATE_INT) ) {
    echo "<br />" . "Entero validado!";
} else {
    echo "<br />" . "Entero NO validado!";
}
```

Holaaaa  
Entero validado!  
IP validada!  
email validado!  
URL validada!

```

$IP = "127.0.0.1";
if (filter_var($IP, FILTER_VALIDATE_IP)) {
    echo "<br />" . "IP validada!";
} else {
    echo "<br />" . "IP NO validada!";
}

$email = "pepe@email.com";
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "<br />" . "email validado!";
} else {
    echo "<br />" . "email NO validada!";
}

$url = "http://www.misitio.com";
$url = filter_var($url, FILTER_SANITIZE_URL);
if (filter_var($url, FILTER_VALIDATE_URL)) {
    echo "<br />" . "URL validada!";
} else {
    echo "<br />" . "URL NO validada!";
}

```

## Filtros avanzados

```

<?php
$entero = 50;
$min = 20;
$max = 100;

if (filter_var($entero, FILTER_VALIDATE_INT,
    array("options" => array(
        "min_range" => $min,
        "max_range" => $max)))) {
    echo "Variable dentro del rango";
} else {
    echo "Variable fuera del rango";
}

$IPv6 = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";
if (filter_var($IPv6, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6))
{
    echo "<br />" . "IPv6 valida!";
} else {
    echo "<br />" . "IPv6 NO valida!";
}

$cadena = "niño";
$cadena = filter_var($cadena, FILTER_SANITIZE_STRING,
    FILTER_FLAG_STRIP_HIGH);
echo "<br />" . $cadena;

```

Variable dentro del rango  
IPv6 valida!  
nio



## JSON

- Javascript Object Notation (JSON), es un formato de almacenamiento e intercambio de datos, en modo texto.
- Permite fácilmente intercambiar información con el servidor y procesar esa información a través de un lenguaje de programación.
- **json\_encode()**: se usa para codificar valores a formato JSON.
- **json\_decode()**: se usa para decodificar un objeto JSON y pasar sus valores a un array asociativo.

```
<?php
$personas = array( "Pepe" => 25,
                  "Maria" => 40,
                  "Manuel" => 14);

$personas_json = json_encode($personas);

echo "<b>codificacion json</b>: " . $personas_json . "<br />";

echo "<b>json_decode devuelve un objeto: </b>";
var_dump(json_decode($personas_json));

echo "<br />" . "<b>json_decode devuelve un array asociativo: </b>";
var_dump(json_decode($personas_json, true));
```

**codificacion json:** {"Pepe":25,"Maria":40,"Manuel":14}

**json\_decode devuelve un objeto:** object(stdClass)#1 (3) { ["Pepe"]=> int(25) ["Maria"]=> int(40) ["Manuel"]=> int(14) }

**json\_decode devuelve un array asociativo:** array(3) { ["Pepe"]=> int(25) ["Maria"]=> int(40) ["Manuel"]=> int(14) }

<pre> &lt;?php \$personas = array( "Pepe" =&gt; 25,                   "Maria" =&gt; 40,                   "Manuel" =&gt; 14);  \$personas_json = json_encode(\$personas);  \$objeto = json_decode(\$personas_json); echo \$objeto-&gt;Pepe . "&lt;br /&gt;"; echo \$objeto-&gt;Maria . "&lt;br /&gt;"; echo \$objeto-&gt;Manuel . "&lt;br /&gt;";  foreach (\$objeto as \$nombre =&gt; \$edad){     echo \$nombre . " tienen " . \$edad . " años" . "&lt;br /&gt;"; }  \$arr_asociativo = json_decode(\$personas_json, true); echo \$arr_asociativo["Pepe"] . "&lt;br /&gt;"; echo \$arr_asociativo["Maria"] . "&lt;br /&gt;"; echo \$arr_asociativo["Manuel"] . "&lt;br /&gt;";  foreach (\$arr_asociativo as \$nombre =&gt; \$edad){     echo \$nombre . " tienen " . \$edad . " años" . "&lt;br /&gt;"; } </pre>	<pre> 25 40 14 Pepe tienen 25 años Maria tienen 40 años Manuel tienen 14 años 25 40 14 Pepe tienen 25 años Maria tienen 40 años Manuel tienen 14 años </pre>
---	--

## Programación orientada a objetos

- PHP permite programar bajo el paradigma orientado a objetos lo cual implica el uso de:
  - **Clases:** plantillas para crear objetos.
  - **Objetos:** instancia de una clase.
  - **Métodos:** funciones de los objetos.
  - **Atributos:** propiedades de los objetos.
  - **Constructores:** métodos que se llaman al destruir un objeto.
  - **Destruyores:** métodos que permiten destruir objetos.
  - **Herencia:** mecanismo por el cual un objeto tiene propiedades de otro.
  - **Interfaces:** forma de definir una clase sin implementarla.
  - **Clases abstractas:** clases parcialmente implementadas.

- En los siguientes apartados veremos como implementa PHP todos estos conceptos de POO, entre otros.

## Clases y objetos

```
<?php
class Persona {
    // Atributos
    private $nombre;
    private $edad;

    // Metodos
    function set_nombre($nombre){
        $this->nombre = $nombre;
    }

    function get_nombre(){
        return $this->nombre;
    }

    function set_edad($edad){
        $this->edad = $edad;
    }

    function get_edad(){
        return $this->edad;
    }
}

// Creacion de objetos
$amigo01 = new Persona();
$amigo02 = new Persona();

// Asignacion de valores
$amigo01->set_nombre("Pepe");
$amigo01->set_edad(35);

$amigo02->set_nombre("Manuel");
$amigo02->set_edad(40);

// Obtencio de valores
echo "Mis amigos son : " . "<br />";
echo $amigo01->get_nombre() . ", " .
    $amigo01->get_edad() . " años<br />";
echo $amigo02->get_nombre() . ", " .
    $amigo02->get_edad() . " años<br />";

var_dump($amigo01 instanceof Persona);
```

Mis amigos son :  
Pepe, 35 años  
Manuel, 40 años  
bool(true)

- `$this` es una variable que referencia al propio objeto y permite acceder a las variables de el mismo, de forma que elimine ambigüedades en caso de que un parámetro tenga el mismo nombre que un atributo de clase.
- El operador `'->'` permite acceder a los atributos y métodos de un objeto.
- Los atributos precedidos por `$this->` no llevarán el símbolo `'$'`.
- `instanceof` permite comprobar si un objeto pertenece a una clase.

## Constructores y destructores de objetos

```
<?php
class Persona {
    // Atributos
    private $nombre;
    private $edad;

    // Constructor
    function __construct($nombre, $edad){
        $this->nombre = $nombre;
        $this->edad = $edad;
    }

    // Destructor
    function __destruct() {
        echo "Persona: $this->nombre, $this->edad ";
    }
}

$amigo01 = new Persona("Pepe",35);
```

Persona: Pepe, 35

- Los constructores y destructores comienzan por doble un guion bajo `"__"` seguido por la palabra reservada `"construct"` y `"destruct"`, respectivamente.
- En PHP **no existe** de forma nativa la **sobrecarga de constructores**, por lo que no podrá haber más de un método con el mismo nombre. Para tener más de un constructor, habrá que **simular** dicha sobrecarga **a través** de la **sobrecarga de métodos**.
- Para simular la sobrecarga de constructores se creará un método genérico sin parámetros para posteriormente, crear métodos específicos para cada combinación de parámetros deseada.

```

<?php
class Jugador {
    private $nombre;
    private $equipo;

    function __construct() {
        // obtención de parámetros recibidos
        $params = func_get_args();
        // obtención del número de parámetros
        $num_params = func_num_args();
        // comprobación y llamada al constructor
        $funcion_constructor = "__construct" . $num_params;
        if (method_exists($this,$funcion_constructor)){
            call_user_func_array(array($this,$funcion_constructor),$params);
        }
    }

    function __construct0(){
        $this->__construct1("Sin nombre");
    }

    function __construct1($nombre){
        $this->__construct2($nombre, "Sin equipo");
    }

    function __construct2($nombre,$equipo){
        $this->nombre = $nombre;
        $this->equipo = $equipo;
    }

    public function getNombre()
    {
        return $this->nombre;
    }

    public function getEquipo()
    {
        return $this->equipo;
    }
}

$jugador = new Jugador();
echo "Jugador: " . $jugador->getNombre() . "<br />";
echo "Jugador: " . $jugador->getEquipo() . "<br />";

$jugador = new Jugador("Pepe");
echo "Jugador: " . $jugador->getNombre() . "<br />";
echo "Jugador: " . $jugador->getEquipo() . "<br />";

$jugador = new Jugador("Pepe","Racing de Ferrol");
echo "Jugador: " . $jugador->getNombre() . "<br />";
echo "Jugador: " . $jugador->getEquipo() . "<br />";

```

```
Jugador: Sin nombre  
Jugador: Sin equipo  
Jugador: Pepe  
Jugador: Sin equipo  
Jugador: Pepe  
Jugador: Racing de Ferrol
```

- Dentro del código del método genérico se usarán las siguientes funciones:
  - **func\_get\_args()**: obtenemos los parámetros pasados a la clase.
  - **func\_num\_args()**: obtenemos el número de parámetros pasados a la clase.
  - **method\_exists()**: una vez creado el formato de nombre de los constructores, formado por la cadena “\_\_construct” concatenada por el número de parámetros recibidos por la clase, se comprueba si existe dicho constructor (construct0, construct1, etc.).
  - **call\_user\_func\_array()**: en caso de que exista un constructor para el número de parámetros recibido por la clase, este será invocado junto con los parámetros actuales recibidos por la clase.
- Las llamadas entre constructores atiende a un modelo en cascada.

## Modificadores de acceso

- Establecen desde donde pueden ser accedidos métodos y atributos:
  - **public**: por defecto. acceso desde cualquier ámbito.
  - **protected**: solo accesible desde la propia clase y desde clases descendientes.
  - **private**: solo accesible desde la clase.

## Herencia y operador de ámbito

- Es el mecanismo que permite que unas clases se construyan a partir de otras.
- En PHP la herencia es simple. Más adelante veremos cómo salvar esta limitación.

- **extends**: implementa la herencia.
- **::** : el operador de resolución de ámbito “::”, permite acceder a métodos y atributos que están fuera de la propia clase. Podemos usarlo para acceder a elementos de la clase padre (**parent**) de la propia clase (**self**) o de otras clases especificando su nombre.

```
<?php
class Constantes{
    const SALUDO = "Hola ";
    const DESPEDIDA = "Adiós! ";
}

class Persona {

    private $nombre;
    private $edad;

    function __construct($nombre, $edad){
        echo "<br />" . Constantes::SALUDO;
        $this->nombre = $nombre;
        $this->edad = $edad;
    }

    function __destruct(){
        echo "<br />" . Constantes::DESPEDIDA;
    }

    function get_nombre(){
        return $this->nombre;
    }

    function get_edad(){
        return $this->edad;
    }
}

class Estudiante extends Persona {
    const ALUMNO = " Alumno ";

    private $curso;

    function get_curso()
    {
        return $this->curso;
    }

    function __construct($nombre, $edad, $curso) {
        parent::__construct($nombre, $edad);
        $this->curso = $curso;
        echo self::ALUMNO;
    }
}
```

```
Hola Pepe, 35 años
Adiós!
Hola Alumno Manuel, 20 años,
Primero CS
Adiós!
```

```

    }
}

$persona = new Persona("Pepe", 35);
echo $persona->get_nombre() . ", " .
      $persona->get_edad() . " años ";
unset($persona);

$estudiante = new Estudiante("Manuel", 20, "Primero CS");
echo $estudiante->get_nombre() . ", " .
      $estudiante->get_edad() . " años, " . "<br />" .
      $estudiante->get_curso();

```

## Clases abstractas

- Son clases que implementadas parcialmente, es decir, al menos un método no estará implementado, solo declarado.
- Serán las clases derivadas por herencia las que implementen los métodos abstractos.
- **abstract**: modificador de clase que indica que una clase es abstracta.

```

<?php
abstract class Instrumentos {
    public $nombre_inst;
    function __construct($nombre_inst) {
        $this->nombre_inst = $nombre_inst;
    }

    abstract function caracteristica() : string;
}

class Viento extends Instrumentos {
    function caracteristica() : string {
        return "$this->nombre_inst: " .
            "El sonido se produce por aire soplado<br />";
    }
}

class Cuerda extends Instrumentos {
    function caracteristica() : string {
        return "$this->nombre_inst: " .
            "El sonido se produce por vibración de cuerdas<br />";
    }
}

class Percusion extends Instrumentos {
    function caracteristica() : string {
        return "$this->nombre_inst: " .
            "El sonido se produce por golpeo<br />";
    }
}

```



```

    }
}

$inst = new Viento("Trompeta");
echo $inst->caracteristica();

$inst = new Cuerda("Guitarra");
echo $inst->caracteristica();

$inst = new Percusion("Timbal");
echo $inst->caracteristica();

```

Trompeta: El sonido se produce por aire soplado  
 Guitarra: El sonido se produce por vibración de cuerdas  
 Timbal: El sonido se produce por golpeo

- Las subclases de la clase **Instrumentos**, **Viento**, **Cuerda** y **Percusión**, implementan el método abstracto **caracteristica**, adaptándolo convenientemente.

## Traits

- trait**: palabra clave que permite declarar métodos que pueden ser usados en múltiples clases, salvando así la limitación de la herencia simple.
- use**: permite llamar a un método **trait** desde una clase.

```

<?php
trait mensaje1 {
    public function msg1() {
        echo "Hola";
    }
}

trait mensaje2 {
    public function msg2() {
        echo ", qué tal??<br />";
    }
}

class Bienvenida1 {
    use mensaje1;
}

class Bievenida2 {

```

Hola  
 Hola, qué tal??

```

    use mensaje1, mensaje2;
}

$saludo1 = new Bienvenida1();
$saludo1->msg1();
echo "<br />";
$saludo2 = new Bievenida2();
$saludo2->msg1();
$saludo2->msg2();

```

## Métodos y atributos estáticos

- Los métodos y atributos estáticos pueden ser usados directamente, sin necesidad de instanciar una clase.
- **static**: declara métodos y atributos como estáticos.

```

<?php
class saludo {

    static $saludo = "Hola ";

    static function saludar($nombre="") {
        echo self::$saludo . $nombre;
    }

    function __construct($nombre){
        self::saludar($nombre);
    }
}

echo saludo::$saludo;

echo "<br />";

saludo::saludar();

echo "<br />";

new saludo( "Pepe");

```

Hola  
Hola  
Hola Pepe

- Se crea un atributo estático, **saludo**, el cual es accedido desde la propia clase mediante **self::** y desde fuera de la clase mediante el nombre de la clase y el operador de ámbito **saludo::**.

- Se crea también un método estático, saludar, el cual es llamado desde la propia clase mediante `self::` y desde fuera de la clase mediante el nombre de la clase y el operador de ámbito `saludo::`.
- Finalmente, también se crea, a través de su constructor, el cual no es un método estático, un objeto de tipo saludo pasándole un parámetro tipo cadena.

## Bases de datos

- PHP proporciona una mecanismos para conectarse a bases de datos y manipularlas de forma que los datos almacenados en las mismas se integren con los sitios web y, de esta forma, poder realizar cualquier operación sobre las bases de datos de una forma controlada desde el propio sitio web sin necesidad de acceder al sistema de gestión de bases de datos para realizarlas.

## Conexión a BBDD

- Usaremos MariaDB (MySQL), como SGBD.
- Formas de conexión al SGBD y principales características:
  - **MySQLi:** (MySQL improved):
    - Extensión solo valida para MySQL.
    - Si se necesitase cambiar una BD de un SGBD a otro, los cambios a realizar en el código PHP con MySQLi son totales.
    - Permite trabajar de forma procedimental y orientada a objetos.
  - **PDO:** (PHP Data Objects):
    - Permite conexión a múltiples SGBD.
    - Si se necesitase cambiar una BD de un SGBD a otro, los cambios a realizar en el código PHP con PDO son mínimos.
    - Solo permite trabajar de forma orientada a objetos.

### Conexión mediante MySQLi

```
<?php
$servidor = "localhost";
$usuario = "root";
$password = "";
// Creacion de la conexion orientada a objetos
$conexion = new mysqli($servidor, $usuario, $password);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
} else {
    echo "MySQLi: Conexion establecia!<br />";
}

/*
 * Punto de creacion de una base de datos (vacio, de momento)
 */

$conexion->close();
```

### Conexión mediante MySQLi

```
<?php
$servidor = "mysql:host=localhost";
$usuario = "root";
$password = "";

try {
    // PDO requiere una BD valida para conectarse
    $conexion = new PDO($servidor, $usuario, $password);
    // Configuramos el modo de error de PDO para excepciones
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "PDO: Conexion establecia!<br >";

    /*
     * Punto de creacion de una base de datos (vacio, de momento)
     */

} catch (PDOException $e){
    echo "PDO: No se pudo realizar la conexion al SGBD: " . $e->getMessage();
}

$conexion=null;
```

MySQLi: Conexion establecia!

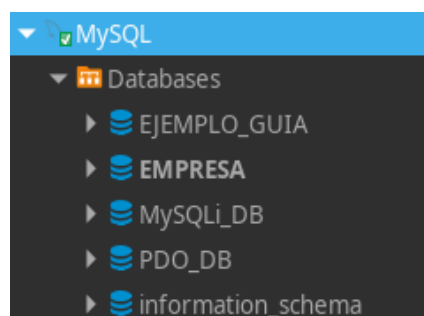
PDO: Conexion establecia!

## Creación de una BD

- Insertaremos ahora el código necesario para la creación de una base de datos en la zona del código comentado para tal tarea.

<pre> ... // Creacion de una base de datos \$consulta = "CREATE DATABASE MySQLi_DB"; if (\$conexion-&gt;query(\$consulta) === true) {     echo "MySQLi: Base de datos creada!&lt;br /&gt;"; } else {     echo "MySQLi: Error creando la base de datos&lt;br /&gt;"; } ... </pre>	MySQLi: Conexion establecida! MySQLi: Base de datos creada!
<pre> ... // Creacion de una base de datos \$consulta = "CREATE DATABASE PDO_DB"; \$conexion-&gt;exec(\$consulta); echo "PDO: Base de datos creada!&lt;br /&gt;"; ... </pre>	PDO: Conexion establecida! PDO: Base de datos creada!

- Podemos comprobar que las BBDD se han creado correctamente en el SGBD:



## Creación de tablas

- Ahora que tenemos las BBDD creadas, podemos eliminar el código que las crea e insertar código para crear tablas.
- En esta versión se realizan cambios en el código:
  - Se comprueba si la base de datos ya existe y, en caso contrario, se crea.
  - Se comprueba si la tabla ya existe y, en caso contrario, se crea.

### Versión MySQLi

```

<?php
$servidor = "localhost";
$db = "MySQLi_DB";
$usuario = "root";
$password = "";
// Creacion de la conexion orientada a objetos
$conexion = new mysqli($servidor, $usuario, $password);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
} else {
    echo "MySQLi: Conexion establecida!<br />";
}

// Creacion de una base de datos
if (!$conexion->select_db($db)) {
    $consulta = "CREATE DATABASE $db";
    if ($conexion->query($consulta) === true) {
        echo "MySQLi: Base de datos creada!<br />";
        // seleccionamos la base de datos y creamos la tabla
        $conexion->select_db($db);
        crear_tabla_MySQLi($conexion);
    } else {
        echo "MySQLi: Error creando la base de datos<br />";
    }
} else {
    echo "MySQLi: La base de datos ya existe<br />";
    $tabla_existe = "SELECT * FROM Contactos";
    if ($conexion->query($tabla_existe)) {
        echo "MySQLi: la tabla existe<br />";
    } else {
        crear_tabla_MySQLi($conexion);
    }
}

// Creación de una tabla
function crear_tabla_MySQLi($conexion) {
    $consulta = "CREATE TABLE Contactos (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR(30) NOT NULL,
        email VARCHAR(50))";
    if ($conexion->query($consulta) === true) {
        echo "MySQLi: Tabla creada!<br />";
    } else {
        echo "MySQLi: La tabla ya existe<br />";
    }
}

$conexion->close();

```

## Versión PDO

```

<?php
$servidor = "mysql:host=localhost";

```

```

$db = "PDO_DB";
$usuario = "root";
$password = "";

try {
    // PDO requiere una BD valida para conectarse
    $conexion = new PDO($servidor, $usuario, $password);
    // Configuramos el modo de error de PDO para excepciones
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "PDO: Conexion establecida!<br >";

    // Se comprueba si existe la BD
    try {
        $conexion->exec("USE $db");
        echo "PDO: La base de datos ya existe<br />";
        crear_tabla_PDO($conexion);
    } catch (PDOException $e){
        // La BD no existe y se crea
        $consulta = "CREATE DATABASE $db";
        $conexion->exec($consulta);
        echo "PDO: Base de datos creada!<br />";
        // seleccionamos la base de datos y creamos la tabla
        $consulta = "USE $db";
        $conexion->exec($consulta);
        crear_tabla_PDO($conexion);
    }
} catch (PDOException $e){
    echo "PDO: No se pudo realizar la conexion al SGBD: " . $e->getMessage();
}

function crear_tabla_PDO($conexion){
    $consulta = "CREATE TABLE Contactos (
        id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR(30) NOT NULL,
        email VARCHAR(50))";

    try {
        $conexion->exec($consulta);
        echo "PDO: Tabla creada!<br />";
    } catch (PDOException $e){
        echo "PDO: La tabla ya existe<br />";
    }
}

$conexion=null;

```

## Inserción de datos

- Crearemos una función que permita insertar datos en la base de datos, una vez que esté creada la tabla y seleccionada.

```

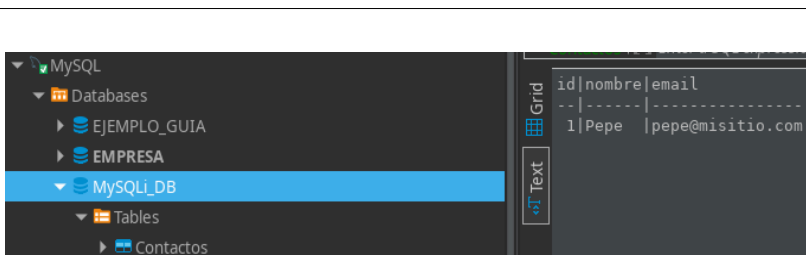
    crear_tabla_MySQLi($conexion);
    insertar_datos_MySQLi($conexion);
} else {

    echo "MySQLi: la tabla existe<br />";
    insertar_datos_MySQLi($conexion);
} else {

}

function insertar_datos_MySQLi($conexion){
    $consulta = "INSERT INTO Contactos (nombre, email)
                VALUES ('Pepe', 'pepe@misitio.com')";
    if ($conexion->query($consulta) === true) {
        echo "MySQLi: Datos insertados!<br />";
    } else {
        echo "MySQLi: Error insertando datos<br />";
    }
}

```



MySQLi: Conexion establecida!  
 MySQLi: Base de datos creada!  
 MySQLi: Tabla creada!  
 MySQLi: Datos insertados!

## Versión PDO

```

    try {
        $conexion->exec("USE $db");
        echo "PDO: La base de datos ya existe<br />";
        crear_tabla_PDO($conexion);
        insertar_datos_PDO($conexion);
    } catch (PDOException $e){
        // La BD no existe y se crea
        $consulta = "CREATE DATABASE $db";
        $conexion->exec($consulta);
        echo "PDO: Base de datos creada!<br />";
        // seleccionamos la base de datos y creamos la tabla
        $consulta = "USE $db";
        $conexion->exec($consulta);
        crear_tabla_PDO($conexion);
        insertar_datos_PDO($conexion);
    }

function insertar_datos_PDO($conexion){

```

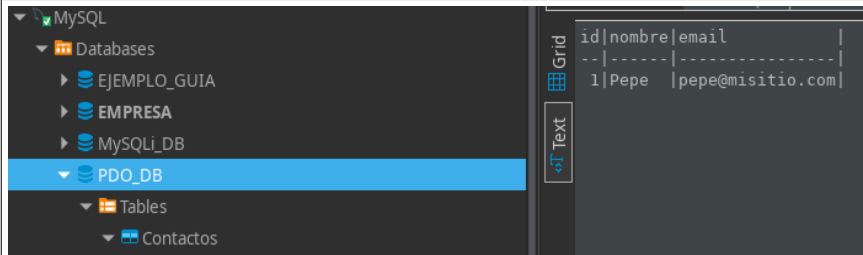


```

$consulta = "INSERT INTO Contactos (nombre, email)
            VALUES ('Pepe', 'pepe@misitio.com')";

try {
    $conexion->exec($consulta);
    echo "PDO: Datos insertados!<br />";
} catch (PDOException $e) {
    echo "PDO: Error insertando datos<br />";
}
}

```



PDO: Conexion establecida!

PDO: Base de datos creada!

PDO: Tabla creada!

PDO: Datos insertados!

- **NOTA:** A partir de este punto trabajaremos ya sobre las BBDD creadas asumiendo que existen y minimizando las comprobaciones con el objetivo de aumentar la claridad del código.

## Inserción de múltiples valores

- **multi\_query()**: permite realizar múltiples inserciones de datos en la BD con una sola instrucción. Es una función de MySQLi. Cada una de las consultas terminará en '; '.
- No existe una instrucción similar en PDO, pero se puede crear una **transacción** que englobe todas las sentencias **INSERT**, de esta forma, en caso de que haya un error y no se completen todas las inserciones, se realizará un **rollback** y la BD no reflejaría ninguna modificación.

### Versión MySQLi

```

<?php
$servidor = "localhost";
$db = "MySQLi_DB";

```

```

$usuario = "root";
$password = "";

$conexion = new mysqli($servidor, $usuario, $password, $db);

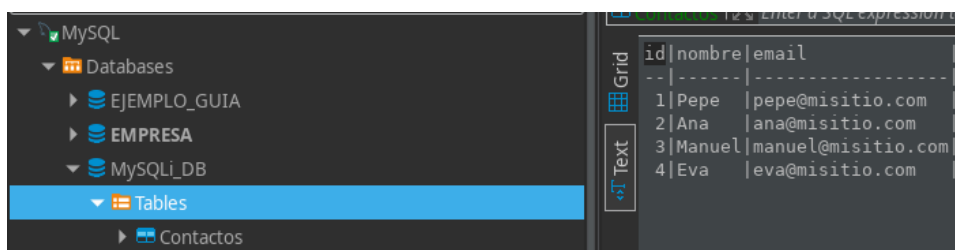
if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
}

$consulta = "INSERT INTO Contactos (nombre, email)
VALUES ('Ana', 'ana@misitio.com');";
$consulta .= "INSERT INTO Contactos (nombre, email)
VALUES ('Manuel', 'manuel@misitio.com');";
$consulta .= "INSERT INTO Contactos (nombre, email)
VALUES ('Eva', 'eva@misitio.com');";

if ($conexion->multi_query($consulta) === true) {
    echo "MySQLi: Datos insertados!<br />";
} else {
    echo "MySQLi: Error insertando datos<br />";
}

$conexion->close();

```



id	nombre	email
1	Pepe	pepe@misitio.com
2	Ana	ana@misitio.com
3	Manuel	manuel@misitio.com
4	Eva	eva@misitio.com

MySQLi: Datos insertados!

## Versión PDO

```

<?php
$srv_db = "mysql:host=localhost;dbname=PDO_DB";
$usuario = "root";
$password = "";

try {
    $conexion = new PDO($srv_db, $usuario, $password);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $conexion->beginTransaction();

    $conexion->exec("INSERT INTO Contactos (nombre, email)
VALUES ('Ana', 'ana@misitio.com')");
    $conexion->exec("INSERT INTO Contactos (nombre, email)
VALUES ('Manuel', 'manuel@misitio.com')");
    $conexion->exec("INSERT INTO Contactos (nombre, email)
VALUES ('Eva', 'eva@misitio.com')");

    $conexion->commit();
}

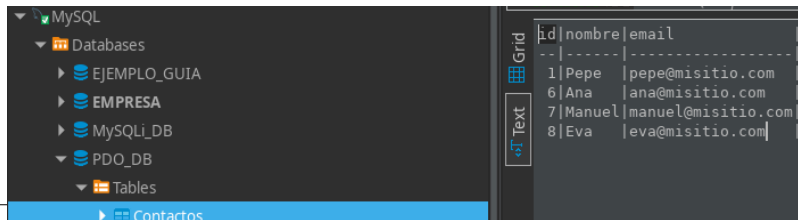
```

```

    echo "PDO: Datos insertados!<br />";
} catch (PDOException $e) {
    echo "PDO: Error insertando datos<br />";
}

$conexion = null;

```



id	nombre	email
1	Pepe	pepe@misitio.com
6	Ana	ana@misitio.com
7	Manuel	manuel@misitio.com
8	Eva	eva@misitio.com

PDO: Datos insertados!

## Sentencias preparadas

- Las sentencias preparadas (prepared statements), permiten parametrizar consultas de forma que se pueda ejecutar una misma consulta todas las veces que sea necesario, pasándole en cada caso, los valores de los campos adecuados.
- prepare()**: configura la consulta parametrizada.
- bind\_param()** y **bindParam()**: el primero de MySQLi y el segundo de PDO, enlazan los parámetros de la consulta parametrizada con variables propias del código. El primer parámetro de este método permite especificar entre comillas los tipos de datos de las variables que se enlazan. Tipos más comunes:
  - i**: entero.
  - s**: string.
- execute()**: una vez asignados valores a las variables enlazadas, este método ejecutará la consulta parametrizada con dichos valores.

### Versión MySQLi

```

<?php
$servidor = "localhost";
$db = "MySQLi_DB";
$usuario = "root";

```

```

$password = "";

$conexion = new mysqli($servidor, $usuario, $password, $db);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
}

$consulta = "INSERT INTO Contactos (nombre, email)
            VALUES (?, ?)";
$prep_stmt = $conexion->prepare($consulta);
$prep_stmt->bind_param("ss", $nombre, $email);

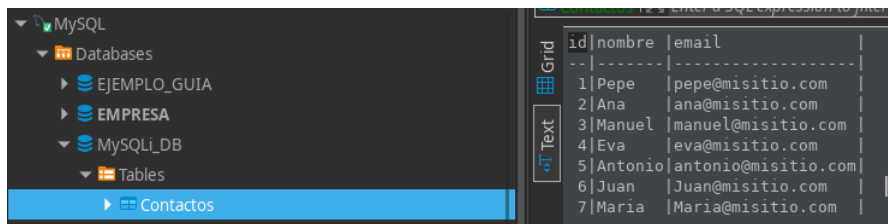
$nombre = "Antonio"; $email = "antonio@misitio.com";
$prep_stmt->execute();

$nombre = "Juan"; $email = "Juan@misitio.com";
$prep_stmt->execute();

$nombre = "Maria"; $email = "Maria@misitio.com";
$prep_stmt->execute();

echo "MySQLi: Datos insertados!<br />";
$conexion->close();

```



id	nombre	email
1	Pepe	pepe@misitio.com
2	Ana	ana@misitio.com
3	Manuel	manuel@misitio.com
4	Eva	eva@misitio.com
5	Antonio	antonio@misitio.com
6	Juan	Juan@misitio.com
7	Maria	Maria@misitio.com

MySQLi: Datos insertados!

## Versión PDO

```

<?php
$srv_db = "mysql:host=localhost;dbname=PDO_DB";
$usuario = "root";
$password = "";

try {
    $conexion = new PDO($srv_db, $usuario, $password);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $consulta_prep = $conexion->prepare("INSERT INTO Contactos (nombre, email)
                                        VALUES (:nombre, :email)");
    $consulta_prep->bindParam(':nombre', $nombre);
    $consulta_prep->bindParam(':email', $email);

    $nombre = "Antonio"; $email = "antonio@misitio.com";
    $consulta_prep->execute();
}

```

```

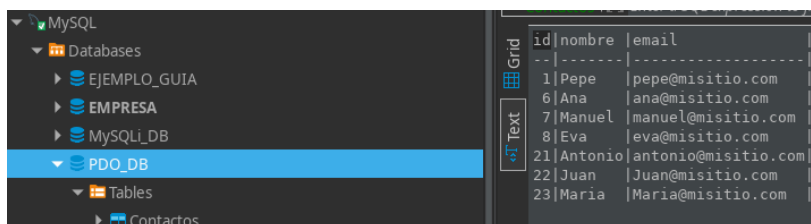
$nombre = "Juan"; $email = "Juan@misitio.com";
$consulta_prep->execute();

$nombre = "Maria"; $email = "Maria@misitio.com";
$consulta_prep->execute();

echo "PDO: Datos insertados!<br />";
} catch (PDOException $e) {
    echo "PDO: Error insertando datos<br />";
}

$conexion = null;

```



id	nombre	email
1	Pepe	pepe@misitio.com
6	Ana	ana@misitio.com
7	Manuel	manuel@misitio.com
8	Eva	eva@misitio.com
21	Antonio	antonio@misitio.com
22	Juan	Juan@misitio.com
23	Maria	Maria@misitio.com

PDO: Datos insertados!

## Select

- Para realizar consultas de recuperación de datos, lo que se de forma general es recuperar los datos para luego ir convirtiéndolos en filas, a partir de las cuales podremos llegar a los campos. Se plantean en los ejemplos dos formas de hacerlo, una, recuperando los datos fila por fila y otra recuperando los datos como un todo.
- **num\_rows()** : comprueba si una consulta ha devuelto al menos una fila.
- **fetch\_assoc()** : devuelve una única fila del resultado y la convierte en un array asociativo
- **fetchAll()** : devuelve un array con todos los datos de la consulta.
- Más sobre ejecución de consultas:
  - **exec()** : ejecuta un programa externo. Evita el uso de **try - catch**.
  - **execute()** : ejecuta una sentencia preparada.

- **query()**: ejecuta una consulta a una base de datos. Indicada para consultas sin parámetro.

#### Version MySQLi

```
<html>
<head>
<style>
table, th, td {
    border: 1px solid black;
}
</style>
</head>
<body>

<?php
$servidor = "localhost";
$db = "MySQLi_DB";
$usuario = "root";
$password = "";

$conexion = new mysqli($servidor, $usuario, $password, $db);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
}

$consulta = "SELECT id, nombre, email FROM Contactos";
$resultado = $conexion->query($consulta);

if ($resultado->num_rows > 0) {
    echo "<table><tr><th><b>ID</b></th>" .
        "<th><b>Nombre</b></th>" .
        "<th><b>eMail</b></th></tr>";
    while ($fila = $resultado->fetch_assoc()) {
        echo "<tr><td>" . $fila["id"] . "</td>" .
            "<td>" . $fila['nombre'] . "</td>" .
            "<td>" . $fila["email"] . "</td></tr>";
    }
    echo "</table>";
} else {
    echo "MySQLi: No hay resultados que mostrar<br />";
}

$conexion->close();
?>
</body>
</html>
```

ID	Nombre	eMail
1	Pepe	pepe@misitio.com
2	Ana	ana@misitio.com
3	Manuel	manuel@misitio.com
4	Eva	eva@misitio.com
5	Antonio	antonio@misitio.com
6	Juan	Juan@misitio.com
7	Maria	Maria@misitio.com

### Version PDO

```
<?php
$srv_db = "mysql:host=localhost;dbname=PDO_DB";
$usuario = "root";
$password = "";

try {
    $conexion = new PDO($srv_db, $usuario, $password);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $consulta = "SELECT id, nombre, email FROM Contactos";
    $resultado = $conexion->query($consulta)->fetchAll();

    if ($resultado) {
        foreach ($resultado as $fila) {
            echo $fila["id"] . " <b>-</b> " .
                $fila["nombre"] . " <b>-</b> " .
                $fila["email"] . "<br />";
        }
    } else {
        echo "PDO: No hay resultados que mostrar<br />";
    }
} catch (PDOException $e) {
    echo "PDO: Error recuperando datos<br />";
}

$conexion = null;
```

1 - Pepe - pepe@misitio.com  
6 - Ana - ana@misitio.com  
7 - Manuel - manuel@misitio.com  
8 - Eva - eva@misitio.com  
21 - Antonio - antonio@misitio.com  
22 - Juan - Juan@misitio.com  
23 - Maria - Maria@misitio.com

## Borrado

### Version MySQLi

```
<?php
$servidor = "localhost";
$db = "MySQLi_DB";
$usuario = "root";
$password = "";
```

MySQLi: Registro borrado!

```

$conexion = new mysqli($servidor, $usuario, $password, $db);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
}

$consulta = "DELETE FROM Contactos WHERE id=7";
if ($conexion->query($consulta) === true) {
    echo "MySQLi: Registro borrado!<br />";
} else {
    echo "MySQLi: No se ha podido borrar el registro<br />";
}

$conexion->close();

```

id	nombre	email
1	Pepe	pepe@
2	Ana	ana@
3	Manuel	manu@
4	Eva	eva@
5	Antonio	anto@
6	Juan	Juan@

```

<?php
$srv_db = "mysql:host=localhost;dbname=PDO_DB";
$usuario = "root";
$password = "";

try {
    $conexion = new PDO($srv_db, $usuario, $password);
    $conexion->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    $consulta = "DELETE FROM Contactos WHERE id=7";
    $conexion->exec($consulta);
    echo "PDO: Registro borrado!<br />";
} catch (PDOException $e) {
    echo "PDO: No se ha podido borrar el registro<br />";
}

$conexion = null;

```

PDO: Registro borrado!

id	nombre	email
1	Pepe	pepe@
6	Ana	ana@
8	Eva	eva@
21	Antonio	anto@
22	Juan	Juan@
23	Maria	Maria@

## Actualización

### Versión MySQLi

```

<?php
$servidor = "localhost";
$db = "MySQLi_DB";
$usuario = "root";
$password = "";

$conexion = new mysqli($servidor, $usuario, $password, $db);

if ($conexion->connect_error){
    die("MySQLi: No se pudo realizar la conexion al SGBD");
}

```

MySQLi: Registro actualizado!

id	nombre	email
1	Pepe	otroemail@misitio.com
2	Ana	ana@misitio.com



```

$consulta = "UPDATE Contactos SET
email='otroemail@misitio.com' WHERE id=1";
if ($conexion->query($consulta) === true) {
    echo "MySQLi: Registro actualizado!<br />";
} else {
    echo "MySQLi: No se ha podido actualizar el
registro<br />";
}

$conexion->close();

```

## Version PDO

```

<?php
$srv_db = "mysql:host=localhost;dbname=PDO_DB";
$usuario = "root";
$password = "";

try {
    $conexion = new PDO($srv_db, $usuario, $password);
    $conexion->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

    $consulta = "UPDATE Contactos SET
email='otroemail@misitio.com' WHERE id=1";
    $conexion->exec($consulta);

    echo "PDO: Registro actualizado!<br />";
} catch (PDOException $e) {
    echo "PDO: No se ha podido actualizar el registro<br
/>";
}

$conexion = null;

```

PDO: Registro actualizado!

id	nombre	email
1	Pepe	otroemail@misitio.com
6	Ana	ana@misitio.com