SELECT * FROM `USERS` WHERE 1session_unset();APUNTES BASICOS PHP:

PHP puede ser declarado en cualquier parte del documento

Iniciar un servidor PHP:

```
/Documentos/PHP/Pruebas$ php -S localhost:8000
PHP 8.1.2-1ubuntu2.22 Development Server (http://localhost:8000) started
127.0.0.1:33732 Accepted
```

Si el fichero .php no tiene HTML incluido (es puro php) nunca cerramos con “?>” (directamente no cerramos el fichero, evitando así errores al incluir ficheros)

Las variables sencillas de PHP apuntan a su valor, es decir $no puede apuntar a &m, la única manera es con una referencia (&) [RECOMENDABLE NO USAR]:

```php
$n = 75;
$s = &$n;
var_dump($s);

$n = 5;
var_dump($s);
```

int(75) int(5)

Crear una variable:

```php
$hola = "Hola";
echo $hola;
```

Evitar cambio de tipos (se va a usar casi siempre):

```php
<?php declare(strict_types=1);?>
```

CONST:
Crear una constante (IMPORTANTE: siempre en mayúscula):
1 manera (const):

```php
no usages
const IVA = 1.21;
```

2 manera (define()):

```php
define("PI", 3.14);
$radio = 10;

echo "El circulo es de " . 2 * PI * $radio;
```

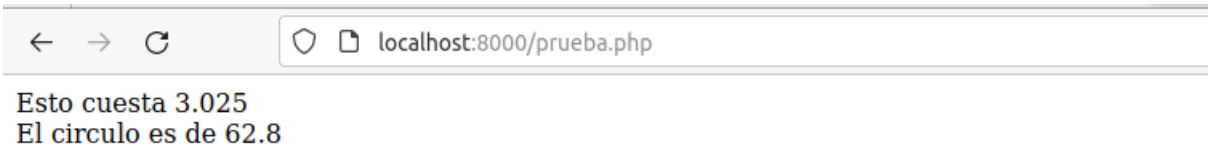
Ejemplo código sencillo:

```php
const IVA = 1.21;

$precio = 2.5;

echo "Esto cuesta " . $precio * IVA;
```

Resultado:



Esto cuesta 3.025
El circulo es de 62.8

INT:

```php
$n = 12;
$n = 0b110001110110; //0b lo convierte en binario
$n = 03457; //0 lo convierte en octal
$n = 0x73ADF1; //0x lo convierte en hexadecimal
```

con var_dump() podemos ver su valor en int:

int(3190) int(1839) int(7581169)

FLOAT:

```php
$n = 12.25;

var_dump($n);
is_float($n);
```

STRING:
Puede ser con “” o con ‘ (“” interpola variables y caracteres, ‘’ no);

```php
echo "Hola $n \n";
echo "<br />";
echo 'Hola $n \n';
?>
```

Con “” podemos llamar a una variable y ejecutar saltos de línea, con ‘’ no.

Hola 12.25
Hola $n \n

También, con [] podemos llamar a un índice del string (los espacios los cuenta tambien)

```php
$cadena = "Hola $n \n";

echo $cadena[0];
```

```
H
```

var_dump() nos permite ver el tamaño del string:

```php
var_dump($cadena);
```

```
string(12) "Hola 12.25 "
```

Con strlen podemos sacar el valor del tamaño del string (si buscamos str podemos ver otras funciones):

```php
echo strlen($cadena);
```

```
12
```

De la siguiente manera podemos declarar un string más grande (conocido como Heredoc):

```php
//Heredoc
$texto = <<<TEXT
Hola
que tal: $n
Como estas
TEXT;

echo $texto;
```

```
Hola que tal: 12.25 Como estas
```

También podemos declarar un “Nowdoc” para asi no interpolar las variables:

```php
//Nowdoc
$texto = <<<'TEXT'
Hola
que tal: $n
Como estas " y '
TEXT;

echo $texto;
```

```
Hola que tal: $n Como estas " y '
```

NULL:

```php
$n = null;

var_dump($n);
```

```
NULL
```

unset elimina una variable, convirtiéndola en null y evitando su uso, isnull devuelve un booleano confirmando si una variable es nula o no

```php
$n = 75;

unset($n);

var_dump(is_null($n));
```

bool(true)

CONVERSIÓN DE TIPOS:
En php, == nos compara el valor y === nos compara el tipo y el valor (normalmente se usa el ===):

```php
$n = 75;
$t = 75.0;

var_dump( value: $n == $t); //igualdad
var_dump( value: $n === $t); //identidad
```

bool(true) bool(false)

Al escribir el tipo con “()” delante de una variable realizamos un “casting”, en el que convertimos una variable de un tipo en otra, por ejemplo pasamos una variable de tipo float a int:

```php
$n = 75;
$t = 75.0;

var_dump( value: $n == $t); //igualdad
var_dump( value: $n === (int)$t); //identidad
```

bool(true) bool(true)

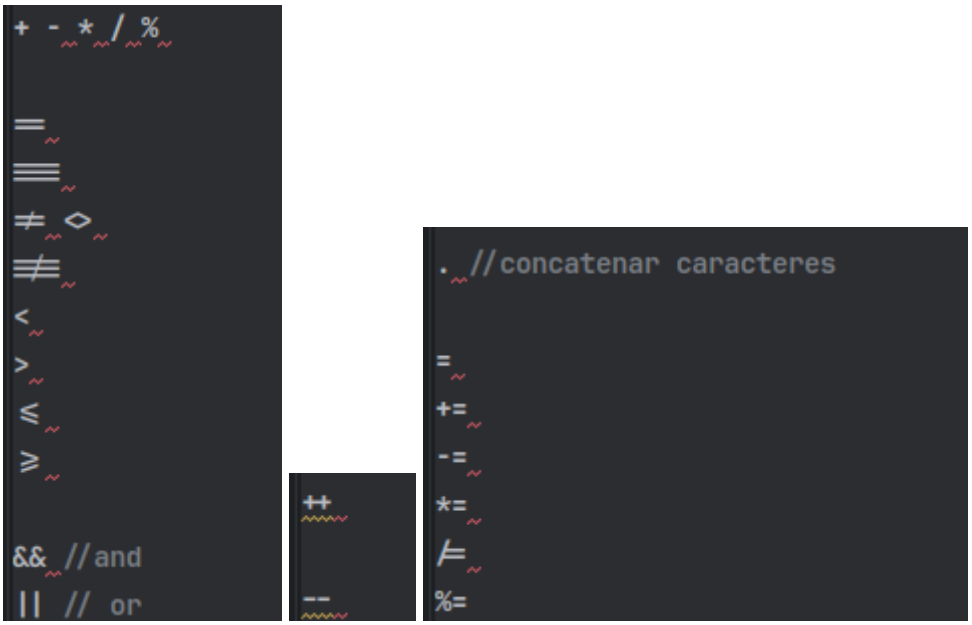
Otro ejemplo de uso en el que transformamos un String en un float (type casting):

```php
$n = "75";

var_dump((double)$n)
```

float(75)

OPERADORES



Operador de coalescencia (Null coalescing operator) asigna el valor de otra variable, en caso de no existir o no tener valor, la variable tomará otro valor.

```php
//Null Coalescing Operator
    $var = $i ?? 9;
    var_dump($var);
```

int(9)

Asignador de coalescencia (Null Coalescing Assignment), asigna a una variable un valor si no tiene ninguno (null cuenta como valor), en caso de tener un valor no lo cambia:

```php
//Null Coalescing Assigment
    $var = 23;
    $var ??= 5;
    var_dump($var);
```

int(23)

ASIGNACIÓN MÚLTIPLE
Asigna el mismo valor a diferentes variables:

```php
$n = "75";

$t = $z = (int)$n;

var_dump($n, $t, $z)
```

string(2) "75" int(75) int(75)

IF (igual a java)

```php
if ($a == $b) {

} elseif ($a == $b) {

} else {

}
```

También se puede hacer tipo Python (no se usa):

```php
if ($a == $b) :
```

SWITCH (igual a java)

```php
switch ($a) {
    case 1:
        break;
    case 2:
        break;
    default:
}
```

MATCH (se parece a switch) devuelve un resultado, switch no, ese resultado puede ser usado de diversas maneras. También se pueden usar operadores para comparar valores. Match tiene en cuenta el tipo de variable (usa ===, mientras que switch usa ==), es decir: “5” no es lo mismo que 5.

```php
$a = 1;
$valor = match($a){
    1 => "OP: 1",
    2 => "OP: 2",
    3 => "OP: 3",
    default => "OP: Default",
};
echo $valor;
```

OP: 1

OPERADOR TERNARIO, se usa para hacer un if comprimido y rapido (los parentesis no son necesarios).

```php
$edad = 15;
$valor = ($edad ⩾ 18) ? "Mayor de edad" : "Menor de edad";
echo $valor;
```

Menor de edad

BUCLE FOR (igual que en java)

```php
for ($i =1; $i ⩽ 10; $i++) {
    echo $i . "<br>";
}
```

1 2 3 4 5 6 7 8 9 10

BUCLE WHILE (igual que en java)

```php
$i = 0;
while ($i < 10) {
    echo $i ++;
}
```

BUCLE DO WHILE (igual que en java)

```php
$i = 0;
do {
    echo $i ++;
} while ($i < 10);
```

BREAK en cualquier momento rompe el bucle, CONTINUE pasa al siguiente valor, pasando de todo lo que tiene debajo (luego es difícil de mantener, por lo que no es recomendable)

FUNCIONES: se necesita la palabra function, luego los parámetros (en caso necesario) y las llaves (en el renglón de abajo)

```php
function suma($a, $b)
{
    return $a+$b;
}

echo suma( a: 1, b: 2);
```

3

también, nos permite cambiar el orden de los parámetros o usar solo algunos parámetros en específico (evitando escribir todos los parámetros si solo vamos a usar uno):

```php
function suma($a, $b)
{
    return $a+$b;
}

echo suma(b: 1,a:2);
```

Si hacemos uso de “&” llamamos a la referencia de esa variable, no a la variable, por lo que al modificar algo en la función, se modifica en la referencia de la variable también (paso por referencia). Esto solo con variables sencillas, las complejas se pasan siempre por referencia (Las buenas prácticas no recomiendan hacer pasos por referencia en funciones). Sin “&”:

```php
function suma($a, $b)
{
    return $a+$b++;
}

$a = 1;
$b = 1;
echo suma(b: $b,a:$a);

echo "<br>". $b;
```

2
1

Con “&”:

```php
function suma($a, &$b)
{
    return $a+$b++;
}

$a = 1;
$b = 1;
echo suma( & b: $b,a:$a);

echo "<br>". $b;
```

2
2

ÁMBITO DE LAS VARIABLES EN PHP (SCOPE):

1. Local:

Son las variables que se definen dentro de una función, son sólo visibles en esa función, es decir, fuera de esa función no existe, por lo que no puede ser usada fuera de la función.

2. Global:

Son variables que se definen fuera de una función, se pueden acceder desde cualquier punto del programa, excepto desde dentro de una función. Si se quiere acceder a ella desde dentro de una función, es necesario declararlo usando la palabra reservada “global”.

```php
$factor = 2;
1 usage
function suma($a, $b)
{
    global $factor;
    return $a + ($b * $factor);
}
```

3. Estático (static)

Son variables que se declaran dentro de una función, pero mantienen su valor a lo largo del tiempo en diferentes llamadas a la función (se acuerda de su ultimo valor). Se puede usar para llevar un contador de cuantas veces se usa una función.

```php
function suma($a, $b)
{
    static $factor = 2;
    return $a + $b + $factor++;
}
```

4. Parámetros de funciones

Son los parámetros que se definen dentro de una función y solo son accesibles dentro de la misma función (no existen fuera de la función).

VARIABLES SUPERGLOBALES

$GLOBALS	
$_SERVER	
$_GET	
$_POST	
$_COOKIE	
$_FILES	
$_ENV	
$_REQUEST	
$_SESSION	

Las funciones en PHP permite declarar el tipo de sus parámetros, para evitar cambios de tipo en las variables:

```php
function suma(int $a, int $b): int
{
    return $a + $b;
}

$a = "1";
$b = 1;
echo suma($a,$b);
```

(da error)

Con ini_set podemos ver los errores que tiene nuestro código (solo cuando estamos en fase de producción)

```php
ini_set( option: 'display_errors', value: 1);
```

Fatal error: Uncaught TypeError: suma(): Argument #1 ($a) must be of type int, string given, called in /home/alumnojcm/Documentos/PHP/Pruebas/prueba.php on line 21 and defined in /home/alumnojcm/Documentos/PHP/Pruebas/prueba.php:14 Stack trace: #0 /home/alumnojcm/Documentos/PHP/Pruebas/prueba.php(21): suma() #1 {main} thrown in **/home/alumnojcm/Documentos/PHP/Pruebas/prueba.php** on line **14**

Para solucionar los errores deberíamos hacer lo siguiente:

```php
function suma(int $a, int $b): int
{
    return $a + $b;
}

$a = "1";
$b = 1;
echo suma((int)$a,$b);
```

Si queremos que la funcion tambien nos permita usar float junto a int, tenemos que hacer una unión de tipos en la función:

```php
function suma(int | float $a, int $b): int | float
{
    return $a + $b;
}

$a = 1.2;
$b = 1;
echo suma($a,$b);
```

2.2

Cuando declaramos una función en PHP, vamos a declarar también sus tipos, para evitar posibles errores en un futuro.

Si queremos que nuestra función no devuelva nada, vamos a usar el tipo void:

```php
function suma(int | float $a, int $b): void
{
    echo $a + $b;
}
```

Si no sabemos qué tipo vamos a utilizar, usamos el tipo “mixed”, qué es una función de todos los tipos.

También con “?” tenemos la opción de hacer que se use un parámetro en caso que sea nulo (convertirlo en nullable)

```php
function suma(int $a, int $b, ?int $c): int
{
    if (is_null($c)){
        return $a + $b;
    }
    else {
        return $a + $b * $c;
    }
}

$a = 1;
$b = 1;
echo suma($a,$b, c: null);
```

Con un “=” podemos declarar el parámetro con un valor por defecto, haciendo que el parámetro reciba ese valor si no se nombra al llamar la función.

```php
function suma(int $a, int $b, ?int $c = 2): int
{
    if (is_null($c)){
        return $a + $b;
    }
    else {
        return $a + $b * $c;
    }
}

$a = 1;
$b = 1;
echo suma($a,$b);
```

Podemos declarar una variable con una cantidad de parámetros indefinida (manipulando esa variable como un array), esto se hace con el uso de "...".

```php
function suma(int ...$n): void
{
    var_dump($n);
}

$a = 1;
$b = 1;
echo suma( ...n: 1,2,3,4,5);
```

```
array(5) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=> int(4) [4]=> int(5) }
```

ARRAYS

Existen dos formas de declarar un array:

```php
$lista1 = [];
$lista2 = array();
```

También, podemos declarar un array con valores predefinidos:

```php
$lista1 = [1,2,3,4,5];
$lista2 = array(1,2,3,4,5);
```

Los array pueden contener valores de diferentes tipos, ya que por defecto usa el tipo "mixed".

```php
$lista1 = [1,2,"3",4,5];
$lista2 = array(1,2,3,"4",5);
```

Para acceder de manera rápida a los valores del array, se puede usar los arrays conocidos como "diccionarios" o "asociativos", en el que cada valor tiene una referencia (guardamos un clave-valor). Con print_r podemos imprimir un array:

```php
$lista3 = [
    'España' => 'Madrid',
    'Francia' => 'Paris',
    'Japon' => 'Tokio'
];
print_r($lista3);
```

```
Array ( [España] => Madrid [Francia] => Paris [Japon] => Tokio )
```

De la siguiente manera podemos agregar un valor al final de un array (dado a que los arrays son dinamicos):

```php
$lista2 = [1,2,3,4,5,6,7,8,9];

$lista2 [] = 5;
print_r($lista2);
```

```
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 [6] => 7 [7] => 8 [8] => 9 [9] => 5 )
```

Con unset podemos eliminar un valor de un array:

```php
unset($lista2[3]);
```

Con count podemos ver el tamaño del array:

```php
var_dump(count($lista2));
```

Si hacemos lo siguiente añadimos el valor al array como diccionario:

```php
$lista2 ['alumno'] = 5;
```

Podemos usar esto para crear un array bidimensional con datos:

```php
$lista[0]['nombre'] = 'Juan';
$lista[0]['apellido'] = 'Pablo';
$lista[0]['edad'] = 18;

$lista[1]['nombre'] = 'Bob';
$lista[1]['apellido'] = 'Lopez';
$lista[1]['edad'] = 28;

var_dump($lista);
```

FOR EACH parecido a java, pero va primero el array, luego la variable:

```php
$var3 = [1,2,3,4,5,6,7,8,9];

foreach ($var3 as $var) {
    dump(var: $var);
}
```

```
int(9)
int(8)
int(7)
int(6)
int(5)
int(4)
int(3)
int(2)
int(1)
```

```php
$lista =[
    ["id" => 1, "nombre" => "Alice"],
    ["id" => 2, "nombre" => "Bob"],
    ["id" => 3, "nombre" => "Charlie"],
    ["id" => 4, "nombre" => "Eve"],
    ["id" => 5, "nombre" => "Fernand"],
];

foreach ($lista as $persona) {
    foreach ($persona as $key => $value) {
        echo $key . " " . $value . "<br>";
    }
    echo "<br/>\n";
}
```

id 1
nombre Alice

id 2
nombre Bob

id 3
nombre Charlie

id 4
nombre Eve

id 5
nombre Fernand

CONSTRUCTOR SPREAD (...)

```php
$lista = [1,2,3,4,5,6,7,8,9];

$lista2= [3,4,...$lista];
dump($lista2);
```

```
array(11) {
  [0]=>
  int(3)
  [1]=>
  int(4)
  [2]=>
  int(1)
  [3]=>
  int(2)
  [4]=>
  int(3)
  [5]=>
  int(4)
  [6]=>
  int(5)
  [7]=>
  int(6)
  [8]=>
  int(7)
  [9]=>
  int(8)
  [10]=>
  int(9)
}
```

CONSTRUCTOR LIST (esta obsoleto)

```php
$valores = [100,0.21];

list($pvp, $iva) = $valores;
dump($pvp);
dump($iva);
```

```
int(100)
float(0.21)
```

DESESTRUCTURACIÓN DE ARRAY:

```php
$valores = [100,0.21,99];

[$pvp, $iva, $pvp2] = $valores;
dump($pvp);
dump($iva);
dump($pvp2);
```

```
int(100)
float(0.21)
int(99)
```

Con esto podemos intercambiar valores de variables:

```php
$x = 10;
$y = 20;

[$y,$x] = [$x,$y];
dump($y);
dump($x);
```

```
int(10)
int(20)
```

SORT ordena un array (menor a mayor):

```php
$lista = [3,5,9,1,0,22,77,88,33,212];
sort( &array: $lista);
dump($lista);
```

```
array(10) {
  [0]=>
  int(0)
  [1]=>
  int(1)
  [2]=>
  int(3)
  [3]=>
  int(5)
  [4]=>
  int(9)
  [5]=>
  int(22)
  [6]=>
  int(33)
  [7]=>
  int(77)
  [8]=>
  int(88)
  [9]=>
  int(212)
}
```

RSORT ordena de mayor a menor (el array es modificado)

```php
$lista = [3,5,9,1,0,22,77,88,33,212];
rsort( &array: $lista);
foreach ($lista as $key => $value) {
    echo "$value,";
}
```

```
212,88,77,33,22,9,5,3,1,0,
```

HTMLSPECIALCHARS sanea una cadena de caracteres

FUNCIÓN ANÓNIMA (callable):

```php
$suma = function ($a, $b) {
    return $a + $b;
};
var_dump($suma( a: 1,  b: 2));
```

```
int(3)
```

La funcion map realiza una función en un array y devuelve otro con la función ya utilizada

```php
$res = array_map(function($item){
    return $item * 2;
}, [1,2,3,4,5]);
var_dump($res);
```

```
array(5) { [0]=> int(2) [1]=> int(4) [2]=> int(6) [3]=> int(8) [4]=> int(10) }
```

FUNCIÓN FLECHA (función anónima pero escrita de otra manera, solo 1 linea):

```php
$suma2 = fn ($a, $b) => $a + $b;
```

```php
$res = array_map(fn($item) => $item * 2, [1,2,3,4,5]);
dump($res);
```

En una función privada, podemos usar “use” para usar variables globales externas a esa función (no recomendable, los cambios realizados a esa variable dentro de la función no se tienen en cuenta fuera de la función)

```php
$IVA = 1.21;

$suma = function ($a, $b) use ($IVA) {
    return $a + $b * $IVA;
};

dump($suma( a: 3,  b: 4));
```

Para poder modificar esa variable, se debe de usar el símbolo “&” (& pasa una variable por referencia

```php
$IVA = 1.21;

$suma = function ($a, $b) use (&$IVA) {
    return $a + $b * ++$IVA;
};

dump($suma( a: 3,  b: 4));
```

En las funciones “FAT ARROW” no es necesario usar “use” ya que tiene acceso a las variables globales. Si se efectúa una modificación a una variable global dentro de esta función su valor fuera de ella no se modifica (no se puede pasar la variable por referencia)

```php
$suma2 = fn ($a, $b) => $a + $b * ++$IVA;
dump($suma2( a: 3,  b: 4));

dump($IVA);
```

```
float(11.84)
float(1.21)
```

SI yo a una variable le meto una cadena de caracteres que equivalga al nombre de una función, se puede usar esa variable como función (normalmente no se usa):

```php
function suma($a, $b) {
    return $a + $b;
};

$cadena = "suma";
echo $cadena(2,3);
```

5

PROGRAMACIÓN FUNCIONAL EN PHP realiza una función en un array y devuelve otro con la función ya utilizada

- MAP

```php
$res = array_map(function($item){
    return $item * 2;
}, [1,2,3,4,5]);
var_dump($res);
```

```
array(5) { [0]=> int(2) [1]=> int(4) [2]=> int(6) [3]=> int(8) [4]=> int(10) }
```

```php
$res = array_map(fn($item) => $item**2, [1,2,3,4,5,6,7,8,9]);
dump($res);
```

```
array(9) {
  [0]=>
  int(1)
  [1]=>
  int(4)
  [2]=>
  int(9)
  [3]=>
  int(16)
  [4]=>
  int(25)
  [5]=>
  int(36)
  [6]=>
  int(49)
  [7]=>
  int(64)
  [8]=>
  int(81)
}
```

- FILTER

Filtra un array recorriendolo y enviándolo a una función (siempre devuelve true o false, dependiendo si cumple con la función o no). En caso de devolver true, lo envía al array

```php
$res = array_filter([1,2,3,4,5,6,7,8,9], fn($item) => ($item%2 == 0));
var_dump($res);
```

```
array(4) { [1]=> int(2) [3]=> int(4) [5]=> int(6) [7]=> int(8) }
```

Tambien se puede usar con una variable como función:

```php
function even($n){
    return $n % 2 == 0;
}

$res = array_filter([1,2,3,4,5,6,7,8,9], callback: "even");
var_dump($res);
```

```
}
array(4) { [1]=> int(2) [3]=> int(4) [5]=> int(6) [7]=> int(8) }
```

- REDUCE (fold en todos lenguajes)

Al igual que con filter, primero pide un array y luego una función, aunque requiere de un valor inicial (situado después de la función)
A partir de un valor inicial, va realizando una función con todos los valores del array enviado, reduciendo el array a un solo valor. El valor anterior se mantiene, para efectuarse con el siguiente del array (0+1, 1+2, 3+3, 6+4…).

```php
$res = array_reduce([1,2,3,4,5,6,7,8,9], fn($a,$b) => $a + $b, initial: 0);
var_dump($res);
```

```
int(45)
```

CARGAR UN FICHERO DESDE OTRO FICHERO (ejemplo: tools.php)
Existen 4 funciones:

1. INCLUDE incluye un fichero esté o no incluido (si ya está incluido puede dar a error). Tambien ejecuta el codigo

```php
include("tools.php");
dump([1,2,3,4,5,6]);
```

2. INCLUDE_ONCE incluye un fichero solo si no está incluido (comprueba previamente si ya está o no incluido)

```php
include_once("tools.php");
dump([1,2,3,4,5,6]);
```

3. REQUIRE realiza la misma función que include, aunque a diferencia de include nos notifica si hay algún error (es mejor que include por esto mismo)

```php
require("tools.php");
dump([1,2,3,4,5,6]);
```

4. REQUIRE_ONCE lo mismo que “require”, solo que comprueba previamente si ya está incluido o no (va a ser el que usemos)

```php
require_once("tools.php");
dump([1,2,3,4,5,6]);
```

“Recordatorio: Si el fichero .php no tiene HTML incluido (es puro php) nunca cerramos con “?>” (directamente no cerramos el fichero, evitando así errores al incluir ficheros)”

GET con QUERY STRINGS
El mecanismo más fácil para pasar datos de una página a otra seria con “Query Strings”: <a>href = “p2.php?usuario=alice”> enlace , transportando así el valor de la variable “usuario”.
Documento 1:

```php
<?php
$usuario = "alice";
$edad = 18;
echo "<a href =\"p2.php?usuario=$usuario&edad=$edad\">Enlace</a>"

?>
```

Index

Enlace hola

Documento 2:

```php
<?php

    $usuario = $_GET["usuario"];
    $edad = $_GET["edad"];

    echo "$usuario tiene $edad años";

?>
```

P2

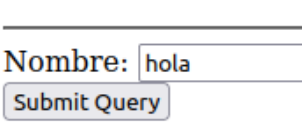
alice tiene 18 años

Esto se usa para cosas que no son cruciales para la aplicación, debido a la poca seguridad.

POST no es accesible a cualquier usuario, solo se puede usar a través de un formulario.
Fichero 1:

```
<h2>Index</h2>
<hr>

<form action="p2.php" method="POST">
    <label>Nombre: </label>
    <input id ="nombre" name ="nombre" type="text"/>
    <br/>
    <input type="submit">
</form>
```



Fichero 2:

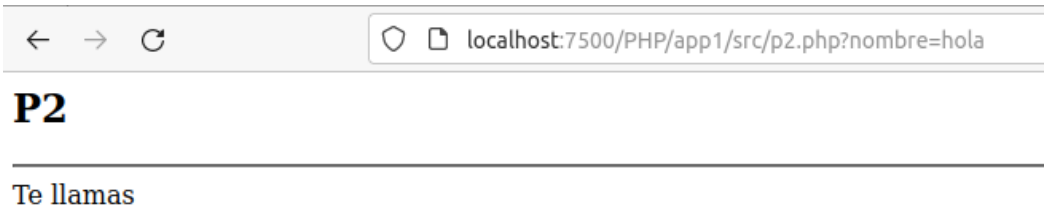
```
<?php

    $nombre = $_POST["nombre"];

    echo "Te llamas $nombre";

?>
```

Si en vez de poner POST ponemos GET lo pasamos por la QueryString (al recibir el valor por POST y enviarlo por GET, el valor es nulo.



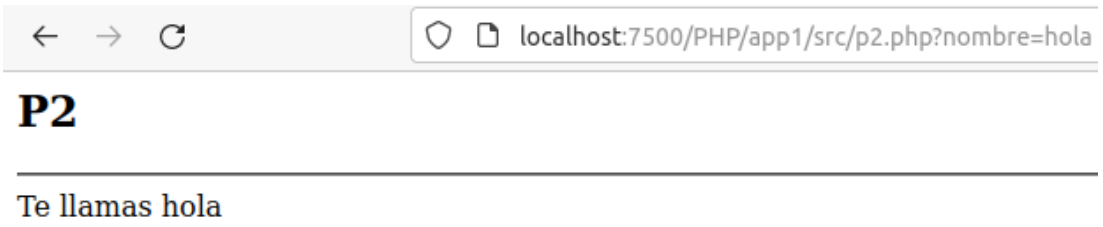
Para recibir el valor por GET y mostrarlo, deberíamos de usar la siguiente sentencia:

```php
<?php

    $nombre = $_POST["nombre"] ?? $_GET["nombre"];

    echo "Te llamas $nombre";

?>
```



Podemos meterle un “name” a un submit, al hacerlo podemos comprobar si el formulario esta correcto o no.

ISSET comprueba si esa variable existe o no (IMPORTANTE, se usa mucho).

```html
<input type="submit" name="registro">
```

```php
if (isset($_POST["registro"])) {

}
```

(El código comprueba si el formulario es correcto o no, en caso de serlo sigue el código)

VALUE (en html) mete un valor por defecto.
Otra supervariable es $_SERVER que nos puede dar información bastante interesante a la hora de hacer una aplicación web.

Otra es $_REQUEST, que muestra la información que contiene la petición (es menos segura que POST Y GET).

Con HTMLSPECIALCHARS podemos evitar que nos introduzcan código no deseado dentro del formulario, al igual pasa con las cookies, ya que estas pueden ser modificadas para tener código.

Con “FILTER_” (y algunas opciones como “VAR”, “INPUT”) podemos validar los formularios. Aunque haya filtros en el cliente, es necesario meter más filtros en el backend (no nos debemos de fiar de lo recogido del frontend):

```php
if(isset($_POST["registro"])){
    $nombre = filter_input( type: INPUT_POST, var_name: "name", filter: FILTER_VALIDATE_EMAIL);
    dump($nombre);
}
else {
    print("No vienes del formulario");
}
```

hola@gmail.com
Submit Query

```
string(14) "hola@gmail.com"
```

(codigo para validar si lo enviado es un email)

Con filter también podemos sanear (lo bueno es que no tiene que coger directamente cada post):

```php
if(isset($_POST["registro"])){
    $nombre = filter_input( type: INPUT_POST, var_name: "name", filter: FILTER_SANITIZE_SPECIAL_CHARS);
    dump($nombre);
}
else {
    print("No vienes del formulario");
}
```

t("Te he hackeado")</script>
Submit Query

```
string(65) "<script> alert("Te he hackeado")</script>"
```

Este código simula el htmlspecialchars pero con filter.

En el caso de tener más de un campo, podemos usar un array para cre

Nombre: pepe
Edad: 34
Submit Query

ar un filtro:

```php
$filtros = array(
    "name" => array(
        "filter" => FILTER_SANITIZE_SPECIAL_CHARS
    ),
    "age" => array(
        "filter" => FILTER_VALIDATE_INT
    )
);

if(isset($_POST["registro"])){
    $campos = filter_input_array( type: INPUT_POST, $filtros);
    echo "El nombre es: {$campos['name']}<br/>";
    echo "La edad es: {$campos['age']}<br/>";
```

El nombre es: pepe
La edad es: 34

En HTML podemos enviar un input de tipo “hidden” que envía un valor de manera obligatoria. Esto se usa para pasar datos de una página a otra a través del formulario sin que el usuario lo vea.

```html
<input id="id" name="id" type="hidden" value="35">
```

SESIONES o VARIABLES DE SESIÓN son variables que se crean en el servidor de modo que todas las páginas pueden leer esas variables y modificarlas. Estas variables están a salvo de ser modificadas por el frontend.Todas las páginas de una misma sesión, ven todas las variables

Requisitos para que las variables de sesiones funcionen:

1. Usar un “session_start()” para iniciar la sesión antes del html.

CREAR UNA VARIABLE SESIÓN:
Archivo1:

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

```php
$_SESSION['usuario'] = "Alice";
```

Archivo2:

```php
session_start()
```

```
echo"<br>Hola $_SESSION[usuario]";
```

Hola Alice

Las COOKIES guardan en el ordenador un fichero con información (en los nuevos navegadores es un registro, no un fichero). Para crear una cookie debemos de hacer lo mismo que con las sesiones, declararlas antes de hacer nada.

Para CREAR UNA COOKIE usamos la función setcookie (variable {obligatorio}, valor que queremos guardar [formato texto] {obligatorio}, tiempo de duración [0 = eterna], ruta [directorio en el servidor donde queremos que esté la cookie. "/" significa que está disponible en todas las carpetas, dominio, https o no [true = solo https]). No nos podemos fiar de todo lo que venga de la cookie ya que puede estar modificada.

Archivo 1:

```
setcookie("username", "alice");
```

Archivo 2:

```
$username = $_COOKIE["username"] ?? "No existe";
echo "<br>Hola {$username}";
```

Hola alice

Cómo establecer tiempo a una cookie:

```
$tiempo = time() + (3600*4);
setcookie("username", "alice",$tiempo);
```

crea una cookie que dura 4h a partir de la fecha actual.

En las sesiones podemos guardar las variables sin problemas, en las cookies solo podemos guardar strings. PHP contiene 2 funciones SERIALIZE (devuelve un string) y UNSERIALIZE (pasa un string a una variable) para poder pasar variables por cookies.

JSON_ENCODE() y JSON_DECODE() hacen lo mismo, pero json no soporta todos los posibles objetos que podemos tener en PHP (los simples los pasa bien, los complejos no). JSON es obligatorio si voy a usar una API, en caso contrario serialize es mejor.

```php
$lista = [
    "nombre" => "Alice",
    "edad" => 17
];

$cadena = serialize($lista);

$lista2 = unserialize($cadena);
echo $lista2 == $lista;
```

1

```php
$lista = [
    "nombre" => "Alice",
    "edad" => 17
];

$cadena = json_encode($lista);

$lista2 = json_decode($cadena);
echo "<pre>";
var_dump($lista2);
echo "</pre>";
echo "<pre>";
var_dump($lista);
```

```
object(stdClass)#1 (2) {
  ["nombre"]=>
  string(5) "Alice"
  ["edad"]=>
  int(17)
}

array(2) {
  ["nombre"]=>
  string(5) "Alice"
  ["edad"]=>
  int(17)
}
```

Como json_encode devuelve un objeto, no coinciden.

```php
header(header: "Location: ../index.php");
```

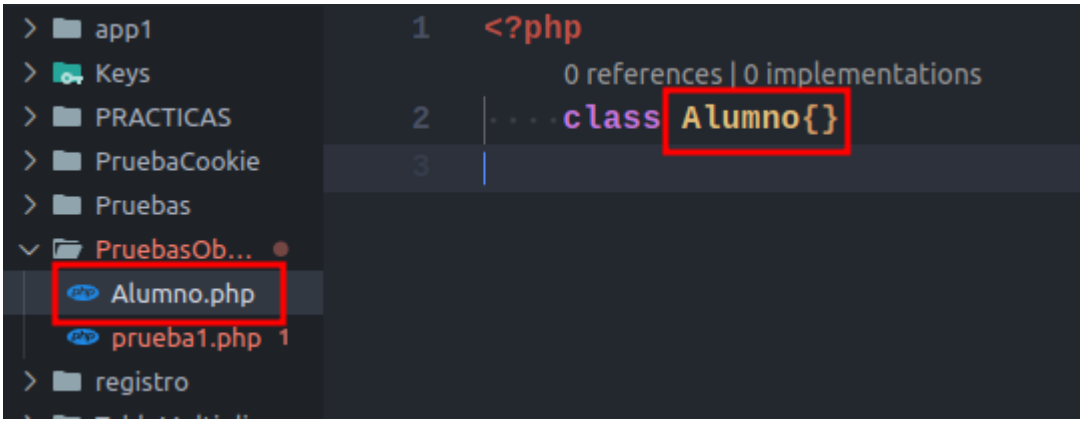
Manda directamente a otro fichero
No meter echo en modelo.

```php
var_dump(value: __DIR__);
```

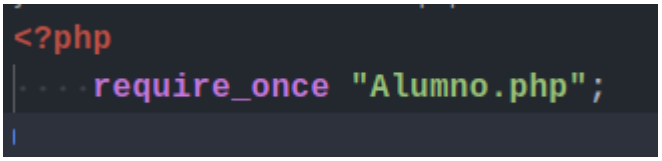
```
/var/www/html/controller/create-controller.php:10:string '/var/www/html/controller' (length=24)
```

```php
$dst = "../img/".$_FILES["img"]["name"];
move_uploaded_file(from: $_FILES["img"]["tmp_name"], to: $dst);
```

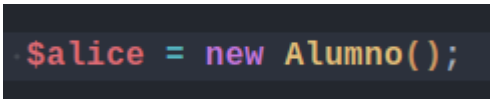
OBJETOS (NO USAR ECHOS NI COSAS QUE DEPENDAN DE LA PAGINA DENTRO DE UN OBJETO.):
Para crear una clase se usa “class”, las puedo escribir donde quiera, aunque el estándar me dice que tengo que crear un fichero por clase (cada fichero tiene que empezar en mayúscula y no se usa el plural). Se usa como sintaxis el camel case.



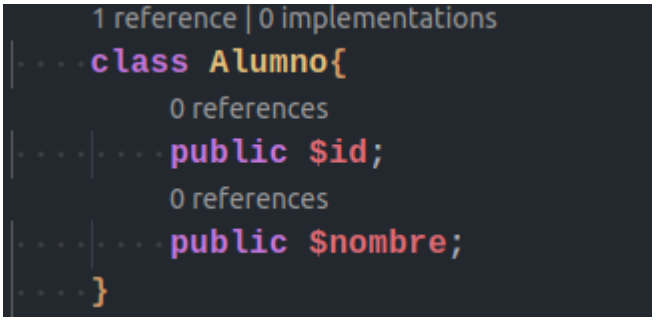
Insertamos la clase:



Si queremos crear un nuevo alumno, hacemos lo siguiente:



En PHP no se llaman atributos, se llaman propiedades, para crear una propiedad, primero tenemos que asignar una variable de acceso



Para asignar un valor a una propiedad:

```php
$alice->id = 13;
$alice->nombre = "Alice";
```

Al igual que podemos asignar propiedades, podemos asignar métodos (si queremos acceder a una propiedad dentro del metodo, usamos this):

```php
public function getNombreCompleto(): string{
    return $this->apellidos.",".$this->nombre;
}
```

Para acceder al método:

```php
echo $alice->getNombreCompleto();
```

Para concatenar métodos (es necesario devolver el objeto):

```php
public function ingresar($cantidad): static{
    $this->saldo += $cantidad;

    return $this;
}
```

```php
$cuenta -> ingresar(cantidad: 50)
        -> ingresar(cantidad: 100)
        -> ingresar(cantidad: 25);
```

Privada: sólo el objeto puede acceder
Pública: Cualquier fichero puede
Protegida: igual que privada, pero permite entrar también a la herencia.

CONSTRUCTORES:
Se puede crear un constructor con __construct:

```php
public function __construct($id,$nombre,$apellidos){
    $this->id = $id;
    $this->nombre = $nombre;
    $this->apellidos = $apellidos;
}
```

```php
$alice = new Alumno(id: 15,nombre: "Alice",apellidos: "Garcia");
```

Se puede simplificar el código con promoción automática de constructores (solo puedo tener un constructor, a partir de la versión 8 de PHP):

```php
class Alumno{
    1 reference | 0 overrides
    public function __construct(
        private $id,
        private $nombre,
        private $apellidos){
    }
```

DESTRUCTORES:
Se puede usar para cerrar una conexión a la base de datos, permite destruir un objeto.
Sirve para liberar recursos (no admite parámetros y solo puedo tener uno):

```php
public function __destruct(){

}
```

Es recomendable ponerle un tipo a las propiedades (en las variables no se puede):

```php
private int $id,
private string $nombre,
private string $apellidos){}
```

Puedo decir que un método me devuelva un tipo de objeto en especifico:

```php
public function ingresar(float $cantidad): Cuenta{
    $this->saldo += $cantidad;

    return $this;
}
```

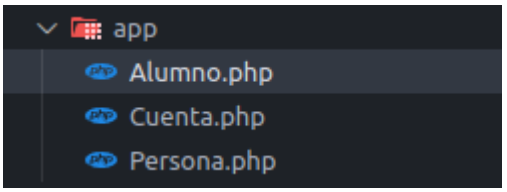
Puedo asignar valores por defecto, con o sin constructor:

```php
public float $saldo = 0.0;
```

Si queremos un atributo que no se pueda modificar, podemos usar un “readonly” (hay que meterle un tipo a la fuerza):

```php
public function __construct(
    private readonly int $id,
    private string $nombre,
    private string $apellidos){

    }
```



HERENCIA:
Cada clase, solo puede heredar de una clase, para marcar que una clase hereda de otra se usa “extends”:

```php
class Alumno extends Persona
{
```

El constructor se crea de la siguiente manera:

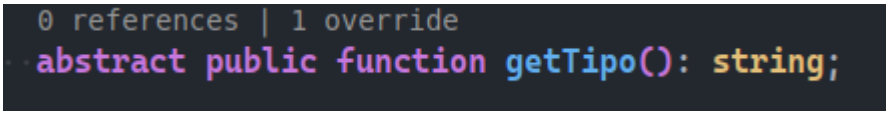
```php
class Alumno extends Persona
{

    1 reference | 0 overrides | prototype
    public function __construct(
        int $id,
        string $dni,
        string $nombre,
        string $apellidos,
        protected int $matricula
    ) {
        parent::__construct(id: $id, dni: $dni, nombre: $nombre,
        apellidos: $apellidos);
    }
}
```

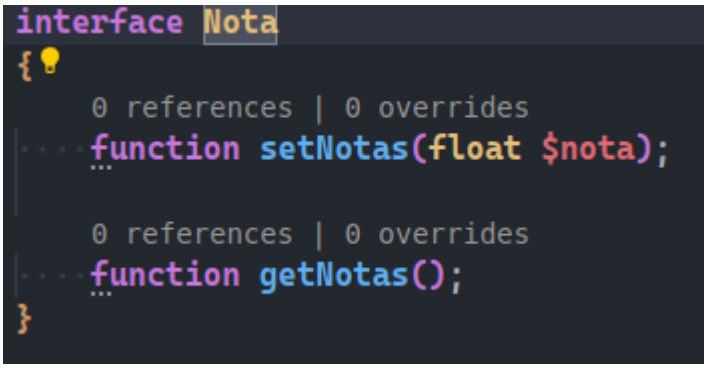
Una CLASE ABSTRACTA es una clase base, en la que no debemos de implementar todos los métodos, al hacerlo no podemos crear objetos de esta clase. Para crear una clase abstracta, se usa la palabra reservada “abstract”.

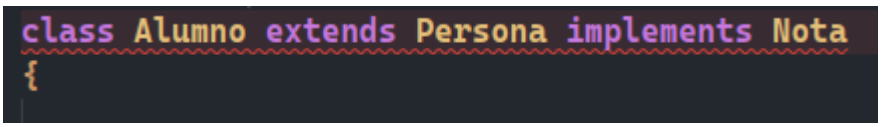
```php
abstract class Persona
{
```

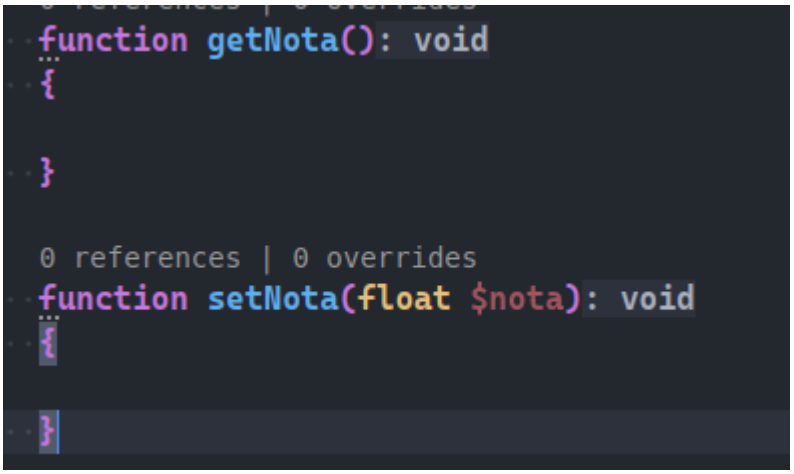
También podemos crear un método abstracto, al hacerlo, todas las clases hijas tienen que implementar este método, en caso de no hacerlo, el hijo también tiene que ser abstracto y alguna de sus herencias tiene que implementar ese método.:



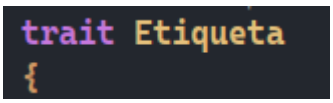
INTERFAZ







Trait es una clase que implementa la funcionalidad que otras clases necesitan, si tenemos varias clases con una misma funcionalidad, podemos crear una clase especial que se usa dentro de otras clases (rasgo).



Para implementar un trait en una clase, usamos “use”.

```php
class Alumno extends Persona implements Nota
{
    use Etiqueta;
```

METODOS Y PROPIEDADES ESTÁTICAS:
Es una propiedad que puedo declarar

```php
class Producto
{

    1 reference
    private static int $n = 0;
    1 reference
    protected int $id;
    2 references | 0 overrides
    public function __construct(
        protected string $name,
    ) {
        $this->id = static::$n++;
    }
```

```php
$p = new Producto(name: "sopa");
$q = new Producto(name: "leche");

var_dump(value: $p);
var_dump(value: $q);
```

```
/var/www/html/public/index.php:8:
object(Producto)[1]
  protected int 'id' => int 0
  protected string 'name' => string 'sopa' (length=4)

/var/www/html/public/index.php:9:
object(Producto)[2]
  protected int 'id' => int 1
  protected string 'name' => string 'leche' (length=5)
```

```php
public static function n(): int
{
    return self::$n;
}
```

```php
var_dump(value: Producto::n());
```

Si yo declaro (principalmente con los métodos) self, nos devuelve el valor de el mismo,

En php.net, magic methods:

> The following method names are considered magical: __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __serialize(), __unserialize(), __toString(), __invoke(), __set_state(), __clone(), and __debugInfo().

CLASES ANONIMAS son clases que definimos pero que no tienen nombre:

```php
$p->setLogger(new class {
    public function log($msg): string
    {
        return $msg . PHP_EOL;
    }
});
```

"PHP_EOL" realiza un salto de línea.

ESPACIOS DE NOMBRE
Al definir una clase es posible que exista una clase con ese nombre, para evitarlo se usa los "namespace". Con use, usamos una clase externa (tenemos que escribirlo en todos los archivos que usen esa clase, además el require es obligatorio).

```php
namespace App\Model;

use App\Model\Producto;
```

Clase Bebida

```php
require_once("../app/model/Producto.php");
require_once("../app/model/Bebida.php");
use App\Model\Producto;
use App\Model\Bebida;
```

Index

Con file_exists comprobamos si un fichero existe

```php
if (file_exists($file)) {
    require $file;
}
```

Con str_replace podemos reemplazar unos caracteres por otros en un string.

```
$file = str_replace(search: "\\", replace: "/",
subject: $class);
```

lcfirst pasa el primer carácter a minúscula

dirname devuelve la ubicación padre de un directorio.

```
$file = dirname(path: __DIR__) .
```

Podemos evitar el uso de require con spl_autoload_register():

```
spl_autoload_register(callback: function ($class): void {
    $file = dirname(path: __DIR__) . "/" . lcfirst(string: str_replace(search: "\\", replace: "/", subject: $class)) . ".php";

    if (file_exists(filename: $file)) {
        require $file;
    }
});
```

Otra manera de hacer esto es con composer, con un archivo llamado “composer.json”:

```
{
    "autoload": {
        "psr-4": {
            "App\\": "app/"
        }
    }
}
```

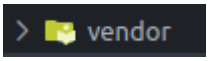
Ahora, accedemos a composer dentro del contenedor:

```
alumnojcm@alumno-E70-SFF:~$ sudo docker exec -it php-fpm-cont /bin/bash
```

Dentro, usamos el siguiente comando:

```
root@fda307ab447a:/var/www/html# composer dump-autoload
```

```
root@fda307ab447a:/var/www/html# ls
app  composer.json  public  vendor
root@fda307ab447a:/var/www/html#
```



Ahora, en index.php:

```
require_once "../vendor/autoload.php";
```

COMPOSER (lo hemos instalado en el contenedor, no en el dispositivo):

```
sudo docker compose ps
```

```
sudo docker exec -it php-fpm-cont /bin/bash
```

```
root@fda307ab447a:/var/www/html# composer --version
Cannot load Xdebug - it was already loaded
Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9003 (fallback through xdebu
g.client_host/xdebug.client_port).
Composer version 2.8.12 2025-09-19 13:41:59
PHP version 8.4.14 (/usr/local/bin/php)
Run the "diagnose" command to get more detailed diagnostics output.
root@fda307ab447a:/var/www/html#
```

EXCEPCIONES:
Iguales que java:

```php
try {
    $agua = new Bebida(name: "Agua");
} catch (Exception $e) {
    echo $e->getMessage();
}
```

Podemos crear una excepcion:

```php
class MiError extends Exception
{

}
try {
    $agua = new Bebida(name: "Agua");
} catch (MiError $e) {
    echo $e->getMessage();
}
```

Con thow lanzamos un error:

```php
}
try {
    $agua = new Bebida(name: "Agua");
    throw new Exception(message: "Error");
} catch (MiError | Exception $e) {
    echo $e->getMessage();
}
```

Error

También podemos añadir un finally, que se ejecuta siempre al acabar el try

```php
} finally {
    echo "";
}
```

Si tenemos un error sin controlar, podemos usar la siguiente funcion:

```php
set_exception_handler(callback: function ($e): void {
    echo "ERROR DESCONTROLADO";
});

$agua = new Bebida(name: "Agua");
throw new Exception(message: "Error");
```

ERROR DESCONTROLADO

Si no se registra el mismo tipo de error, se accede a la función.

```php
class MiError extends Exception
{

}

set_exception_handler(callback: function ($e): void {
    echo "ERROR DESCONTROLADO";
});

try {
    $agua = new Bebida(name: "Agua");
    throw new MiError(message: "Error");
} catch (IntlException $e) {
    echo "ERROR!!";
}
```

ERROR DESCONTROLADO

<<Interfaz>> Traversable

<<Interfaz>> Iterator
+ current(): mixed + key(): mixed + next(): void

+ rewind(): void + valid():bool

<<Interfaz>> IteratorAggregate
+ getIterator():Traversable

En .php (esto sirve para hacer que funcione el for each en una clase)

```php
<?php
3 references | 2 implementations
interface Traversable
{
}

0 references | 0 implementations
interface Iterator extends Traversable
{
    0 references | 0 overrides
    public function current(): mixed;
    0 references | 0 overrides
    public function key(): mixed;
    0 references | 0 overrides
    public function next(): void;
    0 references | 0 overrides
    public function rewind(): void;
    0 references | 0 overrides
    public function valid(): bool;
}

0 references | 0 implementations
interface IteratorAggregate extends Traversable
{
    0 references | 0 overrides
    public function getIterator(): Traversable;
}
```

Si tenemos un método count(), podemos implementar la interfaz Countable para poder usar la función count():

```php
<?php
0 references | 0 implementations
class Datos implements Countable
{
    1 reference
    private int $size = 29;

    0 references | 0 overrides
    public function count(): int
    {
        return $this->size;
    }
}
```

count($misDatos) en vez de $misDatos->count()

ENUM: tipo especial de clase que sirve para definir un conjunto fijo de valores posibles.

```php
<?php
$estado = "pendiente"; //Sin enum

$estado = EstadoPedido::Pendiente; //Con enum
```

CONSTANTES DE CLASE: Se usan constantes dentro de las clases (las constantes deben de ir en Mayúscula la primera letra):

```php
class EstadoPedido
{
    1 reference
    const Pendiente = "pendiente";
    0 references
    const Enviado = "enviado";
    0 references
    const Entregago = "entregado";
}
```

DECLARAR UN ENUM:

```php
2 references
enum EstadoPedido
{
    1 reference
    case Pendiente;
    1 reference
    case Enviado;
    0 references
    case Entregago;
}
```

CASO DE USO:

```php
$estado = EstadoPedido::Pendiente;

if ($estado === EstadoPedido::Pendiente) {
    echo "Tu pedido esta pendiente";
}
```

```
Tu pedido esta pendiente
```

```php
enum Usuarios: string
{
    0 references
    case Administrador = "admin";
    0 references
    case Editor = "editor";
    1 reference
    case Invitado = "guest";
}

$rol = Usuarios::Invitado;
```

```
guest
```

Otra manera de asignar valores del enum:

```php
$rol = Usuarios::from(value: "admin");
$rol = Usuarios::tryFrom(value: "admin");
```

En caso de no existir, from devuelve una excepción y tryFrom null.

Caso de uso con for each:

```php
<?php
3 references
enum EstadoPedido: string
{
    0 references
    case Pendiente = "pendiente";
    0 references
    case Enviado = "enviado";
    2 references
    case Entregado = "entregado";

    1 reference | 0 overrides
    public function esFinalizado(): bool
    {
        return $this === self::Entregado;
    }
}

$estado = EstadoPedido::Entregado;

var_dump(value: $estado->esFinalizado());

foreach (EstadoPedido::cases() as $estado) {
    echo $estado->name . "=>" . $estado->value;
}
```

PATRONES DE DISEÑO (existen muchos, unos 23):
Singleton (solitario/solteron) es un patrón de diseño creacional que sirve para crear una clase que solo puede tener un único objeto (instancia), se suele usar en las bases de datos, si la quiero tener orientada a objetos, para configuraciones globales o para logs o controladores de recursos compartidos.

Funcionamiento:

- Constructor privado -> evita crear objetos con new.
- Propiedad estática -> Almacena la única instancia.
- Método estático getInstance() -> Devuelve la misma instancia siempre.

Ejemplo para crear una clase “Bd”:

```php
class Bd
{
    3 references
    private static ?Bd $instancia = null;

    1 reference
    private function __construct()
    {
        self::conectar();    Call to unknown method: Bd::conectar()
    }

    0 references
    private static function getInstancia(): Bd
    {
        if (self::$instancia === null) {
            self::$instancia = new Bd();
        }

        return self::$instancia;
    }
}
```

BASES DE DATOS:
Existe un driver para cada base de datos, aunque existe uno genérico llamado PDO (PHP DATA OBJECTS).

PDO es una interfaz de acceso a bases de datos de forma segura, consistente y orientada a objetos. Usa sentencias preparadas, evitando inyecciones SQL.

Forma de conectarse (los valores tienen que se los mismos que el contenedor):

```php
$dsn = "mysql:host=localhost;dbname=mi_bd;charset=utf8mb4";
$usuario = "root";
$clave = "";

try{
    $pdo = new PDO($dsn, $usuario, $clave);

    $pdo->setAttribute( attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);
        echo "conectado";
} catch (PDOException $e){
    echo "Error: " . $e->getMessage();
}
```

Hacer consultas (es mejor guardar las consultas en variables), esta es la manera no recomendable, ya que es recomendable preparar primero la consulta:

```php
$sql = "SELECT * FROM usuarios";
$stmt = $pdo->query($sql);

foreach ($stmt as $fila) {
    echo $fila["nombre"] . "<br>";
}
```

Preparar consultas:

Se puede con marcadores posicionales o con marcadores con nombre (el recomendado por el profesor) ya que es menos lioso.

```php
//PREPARAR CONSULTAS
//POSICIONALES
$sql = "SELECT * FROM usuarios WHERE id = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute([1]);

$usuario = $stmt->fetch( mode: PDO::FETCH_ASSOC);

//MARCADORES
$sql = "SELECT * FROM usuarios WHERE email = :email";
$stmt = $pdo->prepare($sql);
$stmt->execute(["email" => "ejemplo@correo.com"]);

$usuario = $stmt->fetch( mode: PDO::FETCH_ASSOC);
```

Realizar modificaciones:

```php
//INSERTAR
$smt = $pdo->prepare( query: "INSER INTO usuarios(nombre, email) VALUES (:nombre, :email)");
$smt->execute(["nombre" =>"Ana", "email" => "ana@mail.com"]);

//UPDATE
$smt = $pdo->prepare( query: "UPDATE usuarios SET email = :email WHERE id = :id");
$smt->execute(["email" => "nuevo@mail.com", "id" => 2]);

//DELETE
$smt = $pdo->prepare( query: "DELETE FROM usuarios WHERE id = :id");
$smt->execute(["id" => 3]);
```

Con $smt->rowCount(); vemos la cantidad de filas modificadas.
Podemos usar el fetch para una o todas las filas.

```php
// una fila
$fila = $stmt->fetch( mode: PDO::FETCH_ASSOC);

//todas
$filas = $stmt->fetchAll( mode: PDO::FETCH_NUM);
```

Una transacción es ejecutar diferentes sentencias que modifican la base de datos y al final se decide si se realizan todas o ninguna (en muchas aplicaciones son importantes).

```php
//Transacciones
try{
    $pdo->beginTransactoion();

    $pdo->prepare( query: "INSER INTO cuentas(nombre, saldo) VALUES ('Juan', '1100')");
    $pdo->prepare( query: "UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1");

    $pdo->commit();
} catch (PDOException $e){
    $pdo->rollBack();
    echo "Error: " . $e->getMessage();
}
```

Cerramos la conexión con $pdo = null (aunque no es obligatorio);

```php
//Acabar
$pdo = null;
```

Podemos usar fectchColumn si queremos recoger un solo resultado (es más rápido).

CREAR BD SINGLETON:

```php
namespace app;
use PDO;
```

Primero, tenemos que crear 4 variables constantes que se usarán para crear el pdo de la base de datos. Para eso necesitamos recoger los datos del docker-compose:

```
const HOSTNAME = "";
no usages
const DBNAME = "";
no usages
const USERNAME = "";
no usages
const PASSWORD = "";
```

En docker-compose:

```
db:
  image: mysql:8.0
  container_name: mysql-db-cont
  environment:
    MYSQL_ROOT_PASSWORD: root_password_segura # ¡Cámbiala!
    MYSQL_DATABASE: my_application_db
    MYSQL_USER: app_user
    MYSQL_PASSWORD: app_password
  volumes:
    - db_data:/var/lib/mysql
  ports:
    - "3307:3306"
```

Entonces, quedaría de la siguiente manera:

```
no usages
const HOSTNAME = "db";
no usages
const DBNAME = "my_application_db";
no usages
const USERNAME = "app_user";
no usages
const PASSWORD = "app_password";
```

Ahora, con todo esto, Bd quedaría de la siguiente manera:

```php
class Bd{
    3 usages
    private static ?Bd $instance = null;
    2 usages
    private static ?PDO $pdo = null;

    1 usage
    private function __construct(){
        self::conectar();
    }
    no usages
    public static function getInstance(): Bd{
        if (self::$instance === null) {
            self::$instance = new Bd();
        }
        return self::$instance;
    }
    1 usage
    private static function conectar(): void{
        $dsn="mysql:host=".HOSTNAME.";dbname=".DBNAME.";charset=utf8";
        if (self::$pdo === null) {
            self::$pdo = new PDO(
                $dsn,
                username: USERNAME,
                password: PASSWORD,
            [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, //Si hay error genera una excepcion
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
            ]);
        }
    }
}
```

Ahora, dentro de esta clase se implementarán las funciones que sean necesarias.

EXIT [exit(-1)]:
Se le pasa un número que devuelve al terminar y para el código.

DIE [die("----")]:
Lo mismo que existe, pero se pasa una línea de caracteres.

En laravel se usa dd que es como el die, pero mejor.

Fetch puede devolver una lista con los datos o un false.

session_unset destruye todas las variables de la sesion, si quieres solo eliminar algunas variables es mejor usar unset:

```php
unset($_SESSION["user"]);
//session_unset();
session_destroy();
```