



PHP

Clase 13

OOP + PHP

PHP

+ OOP



Classes

1. Atributos
2. Constructor
3. \$this
4. new
5. Arrow operator
6. Métodos
7. Constantes

Type Hinting

Return Type



PHP

Clases



Una clase es un molde para la creación de objetos.

Definen un conjunto de propiedades, estados y, mediante sus métodos, el comportamiento de dicha entidad.



Class

Se declara como:

```
class Persona {  
    //...  
}
```



Class - Atributos

```
class Persona {  
    public $nombre;  
    public $edad;  
    public $dni;  
}
```

Class - Constructor

Los constructores son funciones en una clase que son invocadas automáticamente cuando se crea una nueva instancia de una clase con **new**.

En php se define como:

```
public function __construct()  
{  
    //...  
}
```

(También puede ser privado...)

Class - \$this

\$this es una pseudo-variable. El uso de **\$this** dentro de un método, referencia a la instancia puntual en donde será ejecutado el método.

```
public function __construct($nombre, $edad, $dni)
{
    $this->nombre = $nombre;
    $this->edad = $edad;
    $this->dni = $dni;
}
```

¿Pero cómo ejecutamos el método constructor?

Class - *new Class()*

La palabra reservada **new** va a ejecutar el método constructor de la clase para **crear una instancia de objeto del tipo de esta**.

```
$persona1 = new Persona("Diego", 22, 34333232);
```

¿Y si quisiéramos visualizar un atributo o utilizar/acceder a un método de un objeto?

Arrow Operator (->)

El operador -> nos permite acceder a un atributo o un método de la instancia de un objeto.

```
echo $diego->edad;
```

```
//Esto imprime la edad de $diego en pantalla!
```

Nuestra primera Class Persona

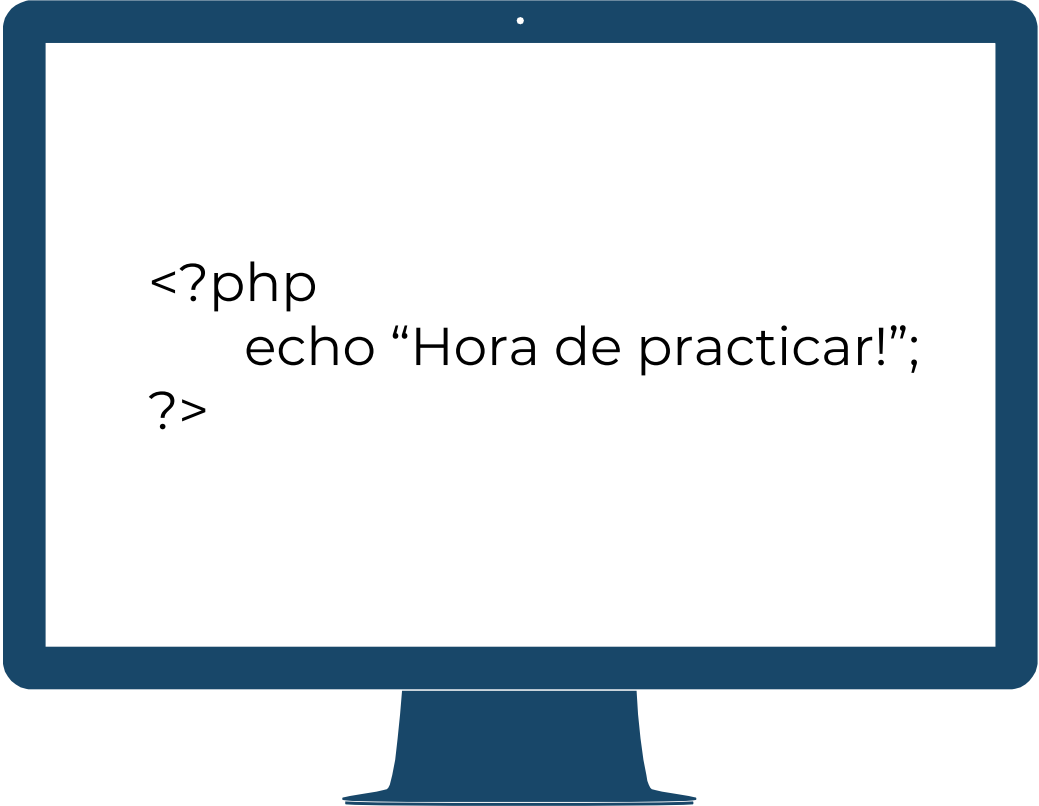
```
class Persona {  
    public $nombre;  
    public $edad;  
    public $dni;  
  
    public function __construct($nombre, $edad, $dni)  
    {  
        $this->nombre = $nombre;  
        $this->edad = $edad;  
        $this->dni = $dni;  
    }  
}
```

```
$persona1 = new Persona("Diego", 22, 34333232);
```

(Con nuestro primer objeto del tipo Persona!)

¡A practicar!

Ejercicios 1 a 9.



```
<?php  
    echo "Hora de practicar!";  
?>
```

PHP

Scope



Dentro de una clase, los distintos atributos y métodos tienen diferentes alcances:


- public
- private
- protected





Scope public

El modificador public hace esa propiedad visible desde cualquier entorno en PHP.



Scope public

```
class Usuario  
{  
    public $nombre;  
}
```


```
$usu1 = new Usuario();  
$usu1->nombre = "Juan";
```

Esto vale! Seteamos nombre porque el atributo es público!





Scope private

El modificador private hace que esa propiedad sea únicamente visible desde la clase a la que pertenece.



Solo podemos acceder a la misma declarando un método público para setearla y otro para mostrarla (getters y setters!).



Scope private

```
class Usuario  
{  
    private $nombre;  
}
```

```
$usu1 = new Usuario();  
$usu1->nombre = "Juan";
```



Esto no vale! Daria un error!



Scope protected

El modificador protected hace que esa propiedad sea únicamente visible desde la clase a la que pertenece **y de sus clases hijas.**

(Veremos más adelante cómo implementar protected de manera funcional cuando sumemos el concepto de herencia en PHP).

Scope protected

```
class Usuario  
{  
    protected $nombre;  
}
```

```
$usu1 = new Usuario();  
$usu1->nombre = "Juan";
```



Esto no vale! Tambien nos daria error!



Class - Métodos

¡Los métodos pueden tener los mismos scopes que los atributos!

Class - Constantes

¡Las constantes también pueden ser públicas (default), privadas o protegidas!



Class - Métodos

Getters/Setters

Los usamos para respetar el concepto de **encapsulamiento**.

Escalabilidad: Es mas facil y practico refactorizar un getter que buscar todas las variables en el código de un proyecto.

Debugging: Podemos insertar un breakpoint (`var_dump()`) en getters y setters.

Class - Métodos

```
class User {  
    private $email;  
    private $password;  
  
    public function __construct($email, $password) {  
        $this->email = $email;  
        $this->password = $password;  
    }  
    public function setEmail($email) {  
        $this->email = $email;  
    }  
    public function getEmail() {  
        return $this->email;  
    }  
}
```

Class - Métodos

Getters/Setters

```
$usuario1 = new User("rodo@digitalhouse.com", "123456");  
//Instancio un nuevo usuario  
$usuario1->setEmail("rodolfo@digitalhouse.com");  
//Cambio de email  
echo $usuario1->getEmail();  
//Imprimo el email en pantalla!
```

Constantes de clase

```
class Prefijo {  
    const BUENOS_AIRES = '011';  
    const MAR_DEL_PLATA = '0223';  
}
```

```
echo Prefijo::BUENOS_AIRES;
```

El operador :: nos permite acceder.

¡A practicar!

Ejercicios 10 a 19.



```
<?php  
    echo "Hora de practicar!";  
?>
```


PHP

Type Hinting



En la firma de una función, PHP nos permite aclarar qué tipo de datos se esperan.

Hasta PHP 7, sóloamente podíamos “hintear” arrays y clases. Desde PHP 7, también podemos “hintear” los tipos básicos.



Type Hinting

```
function holaSoy(string $nombre, int $edad) {  
    return "Hola soy ".$nombre." y tengo ".$edad." años";  
}
```

De esta forma, nos aseguramos que ambos parámetros van a ser del tipo **que la función necesita**. Los tipos de datos que se pueden “Hintear” son: string, int, float, boolean, array, objeto*.

PHP

Return Types



En PHP 7, una nueva característica ha sido implementada: **Declarar el Tipo de Retorno**. La declaración *Return type* especifica el tipo de dato que una función debe retornar.





Return Types

- int
- float
- bool
- string
- Interfaces y clases
- array

Agregando un “?” podemos indicar que se podría devolver null.

Return Types

```
<?php
```

```
declare(strict_types = 1);
```

```
function returnIntValue(int $value): int {  
    return $value;  
}
```

Devuelve 5.

Return Types

```
<?php

declare(strict_types = 1);

function returnIntValue(int $value): int {
    return $value + 1.0;
    //Le sumo 1 a $value pero como float
}
```

Fatal error: Uncaught TypeError: Return value of returnIntValue() must be of the type integer, float returned...

Return Types

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

function returnValue(int $value): ?int {
    return $value > 0 ? $value : null;
    // si $value es > a 0 devuelve $value
    // else, devuelve null
}

echo returnValue(5); // imprime 5
echo returnValue(-5); // imprime vacío
```

¡Gracias!



¿Preguntas?