



Análisis y desarrollo de software.

ID: 2669959



www.sena.edu.co

@SENAComunica

Instructor

Diego Fernando Calderón Silva
Correo: dfcalderon@sena.edu.co

4. Funciones

- **¿Cómo incluir funciones de un archivo PHP a otro?**



4. Funciones

• Include & Require

- Ambas funciones sirven para añadir otros ficheros a nuestros script en PHP

INCLUDE: Inserta en nuestro script un código procedente de otro archivo, si no existe dicho archivo o trae un error, se mostrara un “Warning” y el script seguirá ejecutándose.

REQUIRE: Hace la misma operación que el “include”, pero en caso de no existir el archivo o error mostrara un “Fatal error” y detendrá la ejecución del script.

4. Funciones

funciones.php X incluir_funciones.php

```
funciones.php
<?php

3 function promedio_alumno($nota_1,$nota_2,$nota_3){
    $promedio=($nota_1+$nota_2+$nota_3)/3;
    return $promedio;
}
```

funciones.php X incluir_funciones.php X

incluir_funciones.php

```
<?php

include "funciones.php";

echo "El promedio es: ".promedio_alumno(7,3,9);
```

4. Funciones



- **Include & Require**
- Ambas funciones sirven para añadir otros ficheros a nuestros script en PHP
 - Include_once:** Se incluye una única vez el archivo.
 - Require_once:** Se incluye una única vez el archivo.

Programación orientada a objetos (POO)



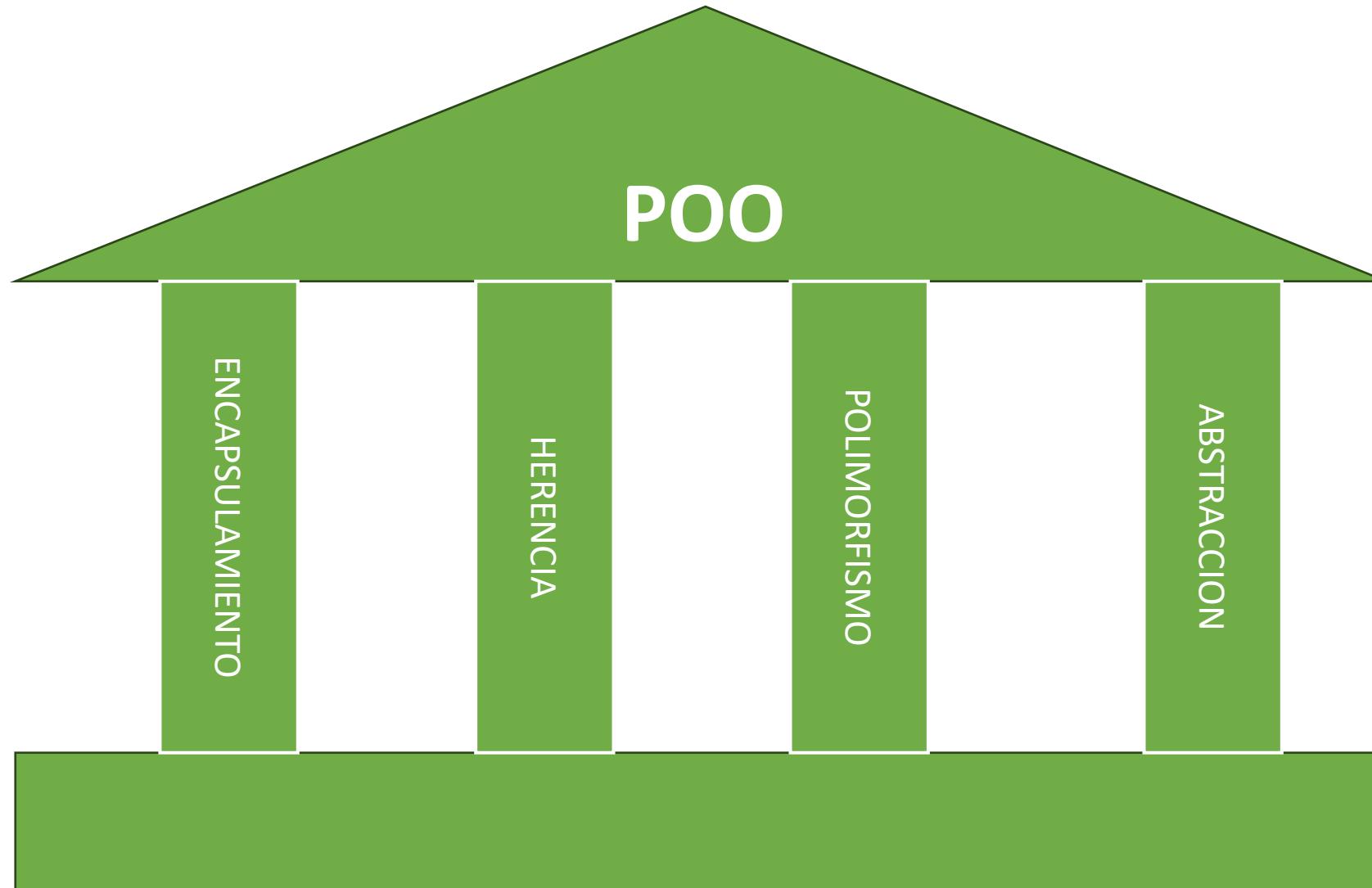
La programación orientada a objetos (POO) es el paradigma de programación más utilizado por la comunidad. Este tipo de programación se basa en la reutilización de código haciendo uso de clases, objetos y métodos. Se utiliza básicamente para darle una coherencia al código y hacerlo reutilizable.

Programación orientada a objetos (POO)



- La POO es un paradigma de programación (o técnica de programación) que utiliza objetos e interacciones en el diseño de un sistema.
- Paradigma: teoría cuyo núcleo central [...] suministra la base y modelo para resolver problemas [...] (Definición de la Real Academia Española, vigésimo tercera edición)
- Como tal, nos enseña un método -probado y estudiado- el cual se basa en las interacciones de objetos para resolver las necesidades de un sistema informático.
- Básicamente, este paradigma se compone de 4 pilares y diferentes características que veremos a continuación.

Programación orientada a objetos (POO)



Programación orientada a objetos (POO)



¿Qué es POO?

para definir POO podemos definir cada sigla con lo que sabes hasta ahora:

Programación: *Escribir código con el objetivo de darle ordenes al ordenador.*

Orientado a: *dirigido a.*

Objetos: modelo informático de un objeto real/ficticio.

Programación orientada a objetos (POO)



Objetos: Es una colección de datos con comportamientos asociados con el afán de representar en código a un objeto real o ficticio.

Propiedades: A la colección de datos que conforman un objeto se les conoce como: propiedades y sirven para describir cómo lucirán los objetos; por ejemplo: color, tamaño, peso, etc...

Métodos: A los comportamientos asociados a un objeto se les conoce como métodos, cada método es algo que nuestro objeto puede realizar: una función.



Programación orientada a objetos (POO)



Clase: En una clase se definen propiedades y métodos. Es el taller que contiene todas las herramientas necesarias para **crear objetos de un tipo específico** como un cuadro por lo que no sirve para crear un Automóvil. Si quisieras crear un Automóvil tendrías que crear una fábrica especialmente para eso.

Instancia: Crear un objeto es hacer una instancia. Si por ejemplo tienes definida una **clase Libro** (que contiene lo necesario para crear Libros) y creas un objeto partiendo de esa clase, estás haciendo una **instancia** de la **clase Libro**



Programación orientada a objetos (POO)

Ejemplo:

Objeto

- Goku
- Vegueta
- Gohan

Clase: Sayayin

Propiedades

- Terco
- Se convierten en SS
- Pelo puntudo
- SS de ojos verdes



Métodos

- Entrenar
- Dormir
- Jugar
- Comer

Programación orientada a objetos (POO)



CLASES: Una clase es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad por eje:



Class:OVNI

PROPIEDADES

- Gris
- Azul
- Redondo

METODOS

- Volar
- Abducciones
-

OBJETO
-Platillo volador

Programación orientada a objetos (POO)



CLASES: Una clase es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad por eje:



PROPIEDADES

- cantidadUno
- cantidadDos
- Resultado

Operaciones

METODOS

- Suma
- Resta
- Multiplicación
- División

OBJETO
-Calculadora

Programación orientada a objetos (POO)



```
1 <?php
2
3 class Operacion{
4     //crear propiedades
5     public $cantidadUno = 0;
6     public $cantidadDos = 0;
7     public $resultado = 0;
8
9     //definir el metodo constructor
10    function __contruct($intCant1, $intCant2){
11        //para asignar valores hacemos lo siguiente
12        //propiedades
13        $this->cantidadUno = $intCant1;
14        $this->cantidadDos = $intCant2;
15    }
16
17    public function getSuma(){
18        $this->resultado = $this->cantidadUno + $this->cantidadDos;
19        return $this->resultado;
20    }
21    public function getResta(){
22        $this->resultado = $this->cantidadUno - $this->cantidadDos;
23        return $this->resultado;
24    }
25    public function getMultiplicacion(){
26        $this->resultado = $this->cantidadUno * $this->cantidadDos;
27        return $this->resultado;
28    }
29    public function getDivision(){
30        $this->resultado = $this->cantidadUno / $this->cantidadDos;
31        return $this->resultado;
32    }
33
34 } //end class Operacion
35
36 ?>
```

Programación orientada a objetos (POO)



```
index | < > index
index > No Selection
1 <?php
2 require_once ("classOperacion.php");
3
4 $hola = new Operacion(10,2);
5
6 $totalSuma = $hola->getSuma();
7 echo "El total de la suma es: ".$totalSuma;
8
```

Programación orientada a objetos (POO)



- **Abstracción:** Aislamiento de un elemento de su contexto. Define las características esenciales de un elemento(propiedades).
- **Encapsulamiento:** Reúne al mismo nivel de abstracción, a todos los elementos que puedan considerarse pertenecientes a una misma entidad.
- **Modularidad:** Característica que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos), independientes unas de otras.

Programación orientada a objetos (POO)



- **Polimorfismo:** Es la capacidad que da a diferentes objetos, la posibilidad de contar con métodos, propiedades y atributos de igual nombre, sin que los de un objeto interfieran con el de otro.
- **Herencia:** Es la relación existente entre dos o más clases, donde una es la principal (padre) y otras son secundarias y dependen (heredan) de ellas (clases “hijas”), donde a la vez, los objetos heredan las características de los objetos de los cuales heredan.

Abstracción y Encapsulamiento



USUARIO



PROPIEDADES

- Nombre
- Email
- Tipo
- Clave

FUNCIONALIDADES

- REGISTRARSE
- Ver Perfil
- Cambiar clave

DIAGRAMA DE CLASES

Usuario
Nombre Email Tipo Clave
Registrarse() Ver_Perfil() Cambiar_Clave()

Abstracción y Encapsulamiento



```
ClassUsuario | < > ClassUsuario
ClassUsuario > No Selection

1 <?php
2
3     class Usuario{
4         //PROPIEDADES
5         private $strNombre;
6         private $strEmail;
7         private $strTipo;
8         private $strClave;
9         protected $strFechaRegistro;
10        static $strEstado = "Activo";
11
12    //METODOS
13    function __construct(string $nombre, string $email, string $tipo)
14    {
15        $this->strNombre = $nombre;
16        $this->strEmail = $email;
17        $this->strTipo = $tipo;
18        $this->strClave = rand();
19    }
20
21    }//End class usuario
22
23 ?>
24
```

Programación orientada a objetos (POO)



- **Modificadores de acceso:** Modificadores de acceso son palabras clave utilizadas en la programación orientada a objetos para especificar el nivel de visibilidad y accesibilidad de los miembros de una clase (atributos y métodos) desde otras partes del programa.

1. Public: Es un modificador de acceso que indica que el miembro de una clase es accesible desde cualquier parte del programa, tanto desde dentro de la clase como desde fuera de ella. Los miembros públicos se pueden acceder y utilizar libremente por otros objetos y clases.

2. Private: Es un modificador de acceso que indica que el miembro de una clase solo es accesible desde dentro de la propia clase. Los miembros privados no se pueden acceder o utilizar directamente desde otras clases u objetos. Su uso principal es encapsular y ocultar la implementación interna de una clase, lo que mejora la seguridad y el control sobre los datos y comportamientos internos.

3. Protected: Es un modificador de acceso similar al privado, pero permite que los miembros de una clase sean accesibles también desde las clases derivadas (subclases) de esa clase. Esto significa que los miembros protegidos pueden ser utilizados en la clase base y en sus subclases, pero no desde otras clases que no sean derivadas de ella.

Programación orientada a objetos (POO)



- **Modificadores de acceso:**

4. Static: No es estrictamente un modificador de acceso, sino un modificador que afecta el ámbito y comportamiento de un miembro de clase. Cuando se aplica el modificador static a un miembro (variable o método), se asocia con la clase en lugar de con las instancias individuales de la clase. Esto significa que se puede acceder al miembro estático sin crear una instancia de la clase. Los miembros estáticos se comparten entre todas las instancias de la clase y se pueden acceder mediante el nombre de la clase en lugar de a través de un objeto específico

Modificadores y métodos de acceso(Getter y Setter)

```
1  <?php
2
3 usages
4 class Usuario{
5     //la abstraccion por que definimos las propiedades
6     //propiedad
7     3 usages
8     private $strName;
9     3 usages
10    private $strEmail;
11    2 usages
12    private $strTipo;
13    3 usages
14    private $strClave;
15    2 usages
16    protected $strFechaRegistro;
17    2 usages
18    static $strEstado = "Activo";//se puede acceder sin instanciar
19    //metodo constructor
20    2 usages
21    function __construct(string $nombre, string $email, string $tipo){
22        $this -> strName = $nombre;
23        $this -> strEmail = $email;
24        $this -> strTipo = $tipo;
25        $this -> strClave = rand();
26        $this -> strFechaRegistro = date( format: 'Y-m-d H:m:s' );
27    }
28
29 } 1 usage
```

```
1  <?php
2
3 require_once "classUsuario.php";
4
5 $objUsuario1 = new Usuario( nombre: "diego", email: "hola@ingo.com", tipo: "admin");
6
7 //echo $objUsuario1->strName."\n";//si fuera publico asi lo podria llamar
8 echo $objUsuario1->getNombre()."\n"; //asi puedo acceder al metodo privado llamando a
9 la funcion getNombre
10
11 //encapsulamiento no permite acceder desde un lugar que no sea la clase donde se creo,
12 esto mismo pasa con protected y static
13 //echo $objUsuario1->srtClave;
14 //echo $objUsuario1->strTipo."\n";
15 echo $objUsuario1->getTipo()."\n";
16
17 //nos estamos dirigiendo directamente a la propiedad
18 //entonces no se necesita instanciar una clase para hacer uso de las declaraciones ya
19 que le corresponde a la clase no a los objetos que se estan creando
20 echo Usuario::$strEstado."\n";
21
22 //mostrar la informacion del usuario
23
24 echo $objUsuario1->getPerfil();
25
26 echo "\n\n";
27 //Voy a crear un nuevo metodo llamado andrea
28
29 $objAndrea = new Usuario( nombre: "Andrea", email: "andrea@info.com", tipo: "vendedor");
```

Modificadores y métodos de acceso(Getter y Setter)

```
//constructor  
  
3 usages  
function __construct(string $nombre, string $documento){  
    $this -> strNombre = $nombre;  
    $this -> strDocumento = $documento;  
  
}  
  
1 usage  
public function getNombre():string{  
    return $this->strNombre;  
}  
  
no usages  
public function getDocumento():string{  
    return $this -> strDocumento;  
}  
  
1 usage  
public function setPlaca(string $placa){  
    $this -> strPlaca = $placa;  
}  
  
1 usage  
public function perfilUsuario(){  
    echo "LOS DATOS DEL USUARIO ".strtoupper($this->strNombre)."\n";  
    echo "Documento ".$this->strDocumento."\n";  
    echo "Placa ".$this->strPlaca."\n";  
}  
}//FIN CLASS USUARIO
```

```
▲ 9 ▼ 7 ^ v 1 <?php  
2  
3 require_once ("classParqueadero.php");  
4  
5 $placa = readline( prompt: "Ingrese el numero de placa del usuario: \n");  
6 $objUsuario = new Usuario("Diego", "1094914260");  
7 //asi se muestran los publicos  
8 //echo "El documento del usuario es: ".$objUsuario->strDocumento." y su nombr  
9  
10 $objUsuario->setPlaca($placa);  
11  
12 echo $objUsuario->perfilUsuario();  
13 //echo "El usuario ".$objUsuario->getNombre()." con documento ".$objUsuario->
```



G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



www.sena.edu.co