Desarrollo de Aplicaciones con Tecnologías Web

Código: IFCD0210

Familia profesional: Informática y Comunicaciones

Área profesional: Desarrollo

Nivel de cualificación profesional: 3

1

Programación Orientada a Objetos (POO) con PHP:

POO Principios Básicos

Crear una clase con PHP

Programación Orientada a Objetos (POO) con PHP:

1.1.- POO – Principios Básicos

Objeto: Conjunto de métodos y propiedades que responden a ciertos eventos.

Ej: un gato tiene características y realiza acciones que lo hacen único.

Características: color del pelo, forma orejas, peso, edad, raza, color ojos, género, forma de la cola, etc…

Acciones: ronronear, maullar, correr, dormir, trepar, comer, saltar, cazar ratones, etc…

Los gatos responden a diferentes **eventos**: llamarlos, asustarlos, rascarlos, etc.

Los gatos pertenecen a la misma **familia**: felis vulgaris

Todos son diferentes, pero todos son gatos. También tienen diferente nombres…

En POO cada vez que pensamos en un OBJETO, podemos pensar en un GATO.

1

Programación Orientada a Objetos (POO) con PHP:

1.1.- POO – Principios Básicos

En programación orientada a objeto, un objeto es una colección de propiedades (datos) y métodos (funciones) que pertenecen a una clase.

- ***Datos*** *(representados en variables)* .- color pelo, ojos, cola, bigotes, etc…
- ***Métodos (o funciones)***.- ronronear, maullar, comer, dormir, etc…
- ***Eventos***.- a lo que van a responder (psssss psssss psssss pssssss….) o no.

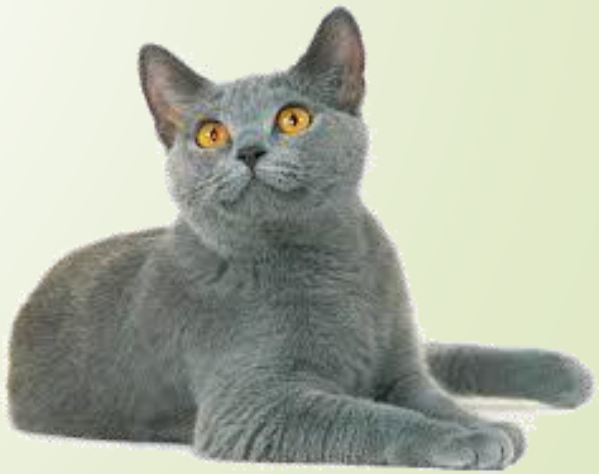
Los objetos en POO se comunican por medio de Eventos.

Vamos a ver como crear el código en PHP para realizar nuestras **clases**, crear nuestras **propiedades** y manejar nuestros **eventos** para utilizarlos en nuestros programas trabajando con **objetos**.

Programación Orientada a Objetos (POO) con PHP:

1.2.- Crear una clase con PHP (versión 5.0 en adelante)

```
<?php
    class Gatos {


    }

//  mostrar todas las clases tanto globales como propias
    $clases = get_declared_classes();
    foreach ($clases as $clase) {
        echo $clase."<br>";
    }
?>
```



Programación Orientada a Objetos (POO) con PHP:

1.2.- Crear una clase con PHP

```
<?php
    class Gatos {

    }
    // Saber si una clase existe o no existe
    if(class_exists("Gatos")){
        echo "La clase existe";
    }else{
        echo "La clase NO existe";
    }
    if(class_exists("FelinusVulgaris")){
        echo "La clase existe";
    }else{
        echo "La clase NO existe";
    }
?>
```

Programación Orientada a Objetos (POO) con PHP:

1.3.- Crear un método de una clase con PHP

```
<?php
class Gatos {
    // creamos un método o función
    function maullar(){
        echo "El gato dice miauuuuu <br>";
    }
}
//para usar un método, podemos crear un objeto, pero de momento
//vamos a ver los métodos que tenemos en una clase determinada
$metodos = get_class_methods("Gatos");
foreach ($metodos as $metodo) {
    echo $metodo."<br>";
}
?>
```

Programación Orientada a Objetos (POO) con PHP:

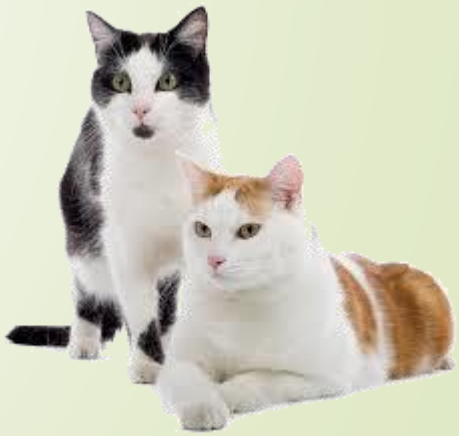
1.3.- Crear un método de una clase con PHP

```
<?php
class Gatos {
        // creamos un método o función
        function maullar(){
                echo "El gato dice miauuuuu <br>";
        }
}
// saber si existe un metodo en una clase. aun no lo estamos llamando
if(method_exists("Gatos", "maullar"))
        echo "El metodo existe";
else
    echo "El metodo NO existe";
// Aún NO hemos creado el objeto, NO lo hemos instanciado, solo hemos preguntado por su método
?>
```

Programación Orientada a Objetos (POO) con PHP:

1.4.- Instanciar un objeto de una clase con PHP

```
<?php
class Gatos {
        // creamos un método o función
        function maullar(){
                echo "miau miauuuuu <br>";
        }
}
// vamos a crear instancias de la clase Gatos, instanciamos objetos
$minino = new Gatos();
$teodoro = new Gatos();
$garfield = new Gatos();

// detecta la clase del objeto
echo "Teodoro pertenece a la clase ".get_class($teodoro)."<br>";
……..
```



Programación Orientada a Objetos (POO) con PHP:

1.4.- Instanciar un objeto de una clase con PHP

```
.............
// verifica que un objeto pertenece a una clase
if(is_a($minino, "Gatos"))
        echo "Si pertenece a la clase Gatos <br>";
else
        echo "NO pertenece a la clase Gatos <br>";

// hacemos una llamada al método "maullar"
echo "Teodoro dice ";
$teodoro->maullar();

echo "Garfield tambien dice ";
$garfield->maullar();
?>
```

Programación Orientada a Objetos (POO) con PHP:

1.5.- Crear propiedades de clase con PHP (variables)

```
<?php
class Gatos {
        var $nombre; //declaramos la variable o propiedad
        var $colorPelo;
        var $corbata="Si"; // declarada con un valor
        // creamos los métodos o funciones
        function maullar(){
                echo "miau miauuuuu <br>";
        }
        function tieneCorbata(){
                return $this->nombre." ".$this->corbata." tiene corbata.<br>";
        }
        function tipoGato(){
                return $this->nombre." ".$this->corbata." tiene corbata y el pelo es de color ".
                $this->colorPelo."<br>";
        }
} ……
```

Programación Orientada a Objetos (POO) con PHP:

1.5.- Crear propiedades de clase con PHP (variables)

```
……..
// vamos a crear instancias de la clase Gatos, instanciamos objetos
$minino = new Gatos();
$teodoro = new Gatos();
$garfield = new Gatos();

echo "Garfield ".$garfield->corbata." tiene corbata<br><hr>";

$minino->nombre = "Minino";
$teodoro->nombre = "Teodoro";
$garfield->nombre = "Garfield";

$minino->corbata="No";
echo $minino->nombre." ".$minino->corbata." tiene corbata<br><hr>";
……..
```

Programación Orientada a Objetos (POO) con PHP:

1.5.- Crear propiedades de clase con PHP (variables)

```
.............
echo $minino->tieneCorbata();
echo $garfield->tieneCorbata();

echo "<hr>";


$minino->colorPelo = "Gris";
$teodoro->colorPelo = "Blanco y Negro";
$garfield->colorPelo = "Naranja";


echo $minino->tipoGato();
echo $teodoro->tipoGato();
echo $garfield->tipoGato();
?>
```

Programación Orientada a Objetos (POO) con PHP:

1.6.- Crear el constructor de una clase

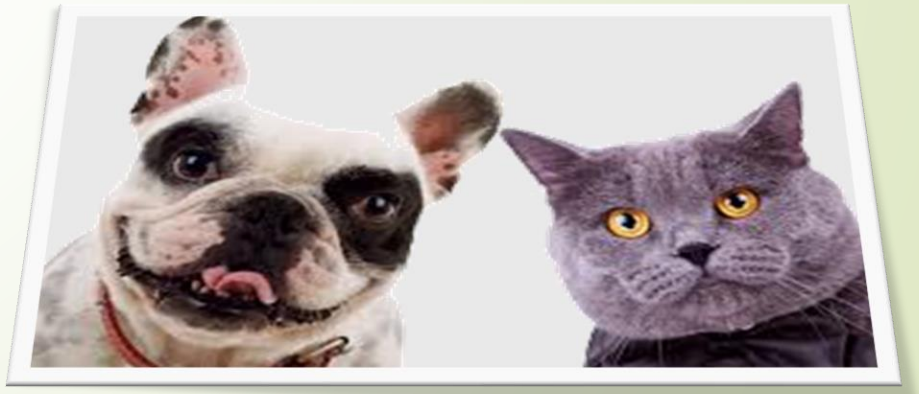
Un constructor es la función que se ejecuta cuando el objeto es creado o instanciado, y nos servirá para recibir parámetros iniciales o para ejecutar acciones o métodos cuando es objeto es "construido".

```
<?php
class Gatos {
        var $nombre;
        var $colorPelo;
        var $corbata="Si";
        // vamos a crear la funcion constructora
        function __construct($n="",$p="negro"){
                $this->nombre = $n;
                $this->pelo = $p;
        }
        // métodos o funciones
        function saludo(){
                print "Hola, soy ".$this->nombre." y tengo el pelo ".$this->pelo.".<br>";
        }
} ……
```

Programación Orientada a Objetos (POO) con PHP:

1.6.- Crear el constructor de una clase

```
.....
// vamos a crear instancias de la clase Gatos, instanciamos objetos
$minino = new Gatos("Minino","rosa");
$teodoro = new Gatos("Teodoro II","azul");
$garfield = new Gatos("Garfield","naranja");
$benito = new Gatos("Benito");

//hacemos la llamada
$minino->saludo();
$teodoro->saludo();
$garfield->saludo();
$benito->saludo();

?>
```



Programación Orientada a Objetos (POO) con PHP:

1.7.- Crear una función destruct para los objetos de nuestra clase

```
<?php
class Gatos {
    var $nombre; //declaramos la variable o propiedad
    var $colorPelo;
    var $corbata="Si"; // declarada con un valor
    // creamos la funcion constructora
    function __construct($n="",$p="negro"){
        $this->nombre = $n;
        $this->pelo = $p;
    }
    //creamos la función destructora
    function __destruct(){
        print $this->nombre." dice: Adios, mundo cruel<br>";
    }
    // métodos o funciones
    function saludo(){
        print "Hola, soy ".$this->nombre." y tengo el pelo ".$this->pelo.".<br>";
    } .....
```

Programación Orientada a Objetos (POO) con PHP:

1.7.- Crear una función destruct para los objetos de nuestra clase

.....

```
// vamos a crear instancias de la clase Gatos con variables para la funcion constructora
$minino = new Gatos("Minino","rosa");
$teodoro = new Gatos("Teodoro II","azul");
$garfield = new Gatos("Garfield","naranja");

// llamamos a la función saludo
$minino->saludo();
$teodoro->saludo();
$garfield->saludo();

// Para eliminar los objetos utilizamos unset
unset($minino);
unset($teodoro);
unset($garfield);
?>
```

Programación Orientada a Objetos (POO) con PHP:

MANOS A LA OBRA

1.- Crear una clase que sirva para….. Representar botones en HTML, que contenga sus medidas y su contenido, así como cualquier otra propiedad o método que se te ocurra (color, fuente, etc…).

Programación Orientada a Objetos (POO) con PHP:

1.8.- La herencia en PHP, programación orientada a objetos

Vamos a jugar a ser dioses y vamos a crear gatos voladores… partimos de la clase Gatos que ya tenemos, lógicamente, así no necesitamos partir de nada.

```
<?php
class Gatos {
        function __construct($n="",$p="negro"){
                $this->nombre = $n;
                $this->pelo = $p;
        }
        function __destruct(){
                print $this->nombre." dice: Adios, mundo cruel<br>";
        }
        function saludo(){
                print "Hola, soy ".$this->nombre." y tengo el pelo ".$this->pelo.".<br>";
        }
}
// creamos una clase que va a heredar las características de Gatos
class GatoVolador extends Gatos{

}    ……
```

Programación Orientada a Objetos (POO) con PHP:

1.8.- La herencia en PHP, programación orientada a objetos

.....

// crear instancia de la clase Gatos

$teodoro = new Gatos("Teodoro","rosa");

// al crear una instancia de la nueva clase, esta hereda las propiedades de la anterior (Gatos)

$garfield = new GatoVolador("Garfield","naranja");

// llamamos a la funcion saludo para comprobar la herencia

$teodoro->saludo();

$garfield->saludo();

// también hereda la función destructora

unset($teodoro);

unset($garfield);

.....

Programación Orientada a Objetos (POO) con PHP:

1.8.- La herencia en PHP, programación orientada a objetos

Vamos a jugar a ser dioses y vamos a crear gatos voladores… partimos de la clase Gatos que ya tenemos, lógicamente, así no necesitamos partir de nada.

……

```
// como conocer las herencias de una clase... quien tiene padre?... la segunda SI es
echo "El padre de la clase Gatos es ".get_parent_class("Gatos").", <br>";
echo "el padre de la clase GatoVolador es ".get_parent_class("GatoVolador")." <br>";

// saber si una clase es subclase de otra..... Primera NO, Segunda SI
echo is_subclass_of("Gatos","GatoVolador")?"SI":"NO";
echo "<br>";
echo is_subclass_of("GatoVolador","Gatos")?"SI":"NO";
echo "<br>";

?>
```

Programación Orientada a Objetos (POO) con PHP:

1.9.- Introducción al encapsulamiento en PHP

La base del encapsulamiento es la capacidad de modificar los accesos. Nuestras clases son capsulas de programación que podemos proteger o permitir su acceso, dependiendo de su encapsulación.

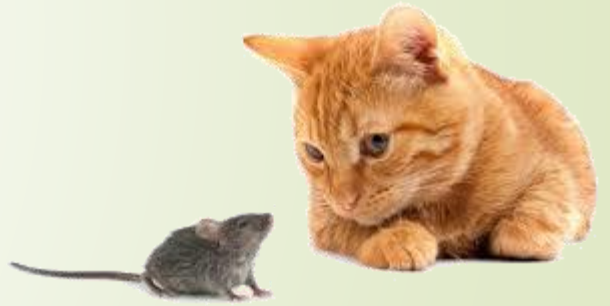
Modificadores de acceso

- **Public .- se puede ver desde cualquier sitio y todos tienen acceso**
- **Private.- solo se puede ver dentro de la clase, nadie puede acceder desde fuera de la clase**
- **Protected.- se ve dentro de la clase, pero las subclases también tienen acceso al método o a la propiedad**

Programación Orientada a Objetos (POO) con PHP:

1.10.- Los modificadores de acceso en PHP: public, private y protected.

```
class Gatos {
        private $nombre; //solo accesible dentro de la clase
        public $pelo; // será accesible desde cualquier parte del codigo
        protected $cola="Sl"; // accesible en la clase y subclases, pero no fuera
        function __construct($n="",$p="negro"){
                $this->nombre = $n;
                $this->pelo = $p;
        }
        function __destruct(){
                print $this->nombre." dice: Adios, mundo cruel<br>";
        }
        public function saludo(){
                print "Hola, soy ".$this->nombre." y tengo el pelo ".$this->pelo.".<br>";
        }
}
```



Programación Orientada a Objetos (POO) con PHP:

1.10.- Los modificadores de acceso en PHP: public, private y protected.

```
.....
class GatoVolador extends Gatos{
        function volador(){
                // echo $this->nombre; no se ve porque es privado en la clase Gatos
                echo "Tengo el pelo ".$this->pelo."<br>";  // se ve porque es publico
                echo "Tengo cola ".$this->cola."<br>";  // se ve porque es protected y es clase heredada
        }
}
$teodoro = new Gatos("Teodoro","rosa");
$garfield = new GatoVolador("Garfield","naranja");

echo $teodoro->pelo;  // se ve porque la variable es publica
echo $teodoro->nombre; dará un error porque es una variable privada
echo "<br>";
echo $garfield->volador();  //muestra las partes publicas y protected
 ?>
```

Programación Orientada a Objetos (POO) con PHP:

1.11.- Crear setters y getters en PHP-POO.

Las variables nunca deberían ser públicas. Para eso utilizaremos los setters (modificar o asignar variables) y los getter (recuperar valores de variables)

```
<?php
class Gatos {
// las variables nunca deben ser publicas por seguridad
    private $nombre;  private $pelo;  private $cola="Si";
    // aquí el constructor y el destructor
    public function setNombre($n){
        if($n!=""){
            $this->nombre=$n;
        }
    }
    //getter para recuperar los valores
    public function getNombre(){
        return $this->nombre;
    }
} .....
```

Programación Orientada a Objetos (POO) con PHP:

1.11.- Crear setters y getters en PHP-POO.

Las variables nunca deberían ser públicas. Para eso utilizaremos los setters (modificar o asignar variables) y los getter (recuperar valores de variables)

.....

```
$teodoro = new Gatos("Teodoro","rosa");
$garfield = new GatoVolador("Garfield","naranja");

//como hacer las llamadas y asignaciones?

//echo $teodoro->nombre;  ahora esto da un error
echo $teodoro->getNombre();
$teodoro->setNombre("Teodoro Peligro");
echo $teodoro->getNombre();
?>
```

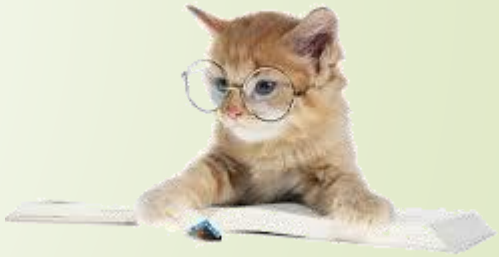
Programación Orientada a Objetos (POO) con PHP:

1.12.- Metodos mágicos __get y __set.

NO confundir con los getters y los setters.

Sustituimos las funciones con __set y __get pero cometemos un error ya que con ello conseguimos abrir el encapsulamiento y perdemos la privacidad.

```
<?php
……..
function __set($name, $valor){
        $this->$name=$valor;
    }
    function __get($name){
        return $this->$name;
    }
……..
// ahora las llamadas a las variables funcionan porque hemos abierto el encapsulamiento
echo $teodoro->pelo;
echo $teodoro->nombre;
?>
```



Programación Orientada a Objetos (POO) con PHP:

1.13.- Métodos y propiedades estáticas con static.

Se utiliza para definir datos que sean comunes a toda la clase. Se pueden definir constantes y métodos static. Las llamadas se hacen como Clase::$constante ó Clase::metodo();

```
<?php
class Gatos {
        static $claveSecreta="12345";
        ……..
        // creamos un metodo estatico, no se puede utilizar $this
        static function caminoMagico(){
                echo "El camino magico es que puedas trabajar en lo que mas te guste ";
                // echo $this->claveSecreta;       esto NO se puede utilizar, ERROR, se hace como sigue…
                echo "y para ello utiliza la clave secreta ".Gatos::$claveSecreta."<br>";
        }
……….
// Para llamarlos
echo "La clave secreta para ser un super gato es ".Gatos::$claveSecreta."<br>";
echo Gatos::caminoMagico();
?>
```

Programación Orientada a Objetos (POO) con PHP:

1.14.- Sobrescribir un método en una clase: overriding.

Consiste en crear un método en una clase heredada con el mismo nombre que el de la clase padre. Al hacer la llamada prevalece la heredada sobre la original.

```
<?php
class Gatos {
        static $claveSecreta="12345";
        protected $nombre;
        ……..
        function maullar(){
                echo $this->nombre." dice miau miau.<br>";
        }
}
class GatoVolador extends Gatos{
        function maullar(){
                echo $this->nombre." dice Miauuuuuuuuu Miauuuuuuuuu.<br>";
        }
} ……..
```

Programación Orientada a Objetos (POO) con PHP:

1.14.- Sobrescribir un método en una clase: overriding.

Consiste en crear un método en una clase heredada con el mismo nombre que el de la clase padre. Al hacer la llamada prevalece la heredada sobre la original.

……

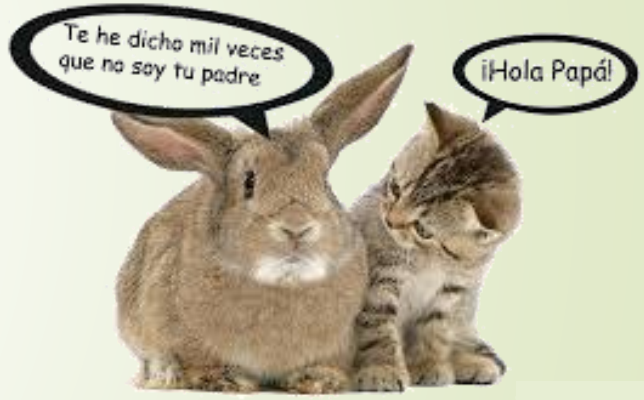
```
$teodoro = new Gatos("Teodoro","rosa");
$garfield = new GatoVolador("Garfield","naranja");

echo $teodoro->maullar(); // llamada clase padre
echo $garfield->maullar();  // llamada clase heredada

?>
```

Programación Orientada a Objetos (POO) con PHP:

1.15.- Scope Resolution Operator (operador de resolucion de alcance): self y parent.

Se utiliza con los parámetros self y parent para hacer llamadas a métodos dentro de la misma clase (self) y de la clase padre (parent)

```
<?php
class Gatos {
    static $claveSecreta="12345";
    ……..
    static function adios(){
        echo "Adios amigos de los gatos<br>";
    }
    static function caminoMagico(){
        echo "El camino magico es que puedas trabajar en lo que mas te guste ";
        // vamos a sustituir el nombre de la clase por self
        echo "y para ello utiliza la clave secreta ".self::$claveSecreta."<br>";
        self::adios();
    }
……..
```

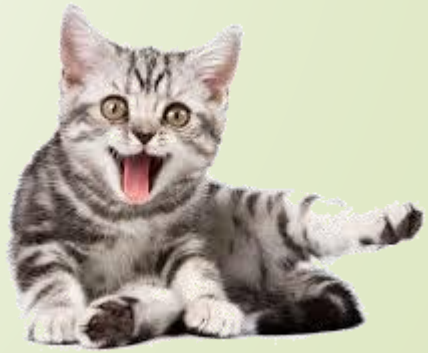


Programación Orientada a Objetos (POO) con PHP:

1.15.- Scope Resolution Operator (operador de resolucion de alcance): self y parent.

…….. En la clase que hereda se haría así

```
class GatoVolador extends Gatos{
        function volador(){
                parent::maullar(); // si quiero llamar al maullar de la clase padre
        }
        function maullar(){
                echo $this->nombre." dice Miauuuuuuuuu Miauuuuuuuuu.<br>";
                parent::adios(); // podemos usar tambien $this->adios();
                // parent::maullar(); si quiero llamar al maullar de la clase padre
        }
}
….. // y las llamadas
echo $teodoro->maullar();
echo $garfield->maullar();
Gatos::caminoMagico();
$garfield->volador();
?>
```



Programación Orientada a Objetos (POO) con PHP:

1.16.- Clonar objetos en PHP con la sentencia clone.

```
<? ..............
// utilizando referencias como a continuación, debemos tener cuidado porque lo que modifiquemos en uno se modificará en el otro
 $supergato = $teodoro;
echo $supergato->getNombre()."<br>";
$supergato->setNombre("Supergato");
echo $teodoro->getNombre()."<br>";
echo $supergato->getNombre()."<br>";

// Si queremos hacer una copia de un objero, utilizaremos clone, asi no modificamos el original
$supergato = clone $teodoro;
echo $supergato->getNombre()."<br>";
$supergato->setNombre("Supergato");
echo $teodoro->getNombre()."<br>";
echo $supergato->getNombre()."<br>";
…….. ?>
```