**MODULE – 4(Advance PHP) OOPS**

1. **What Is Object Oriented Programming?**

* In this tutorial you will learn how to write code in object-oriented style in PHP. Object-Oriented Programming (OOP) is a programming model that is based on the concept of classes and objects.

1. **What Are Properties Of Object Oriented Systems?**

* An object oriented system revolves around a Class and objects. A class is used to describe characteristics of any entity of the real world. An object is a pattern of the class. An actual object created at runtime is called as an instance. A class, apart from characteristics has some functions to perform called as methods. For.e.g A class named “Food” has attributes like ‘price’, ‘quantity’. “Food” class has methods like Serve\_food(), bill\_food().  
    
  **Object:** Objects in Object Oriented Systems interact through messages.  
    
  **Inheritance:** The main class or the root class is called as a Base Class. Any class which is expected to have ALL properties of the base class along with its own is called as a Derived class. The process of deriving such a class is Derived class. For the “Food” class, a Derived class can be “Class Chinesefood”.  
    
  **Abstraction:** Abstraction is creating models or classes of some broad concept. Abstraction can be achieved through Inheritance or even Composition.  
    
  **Encapsulation:** Encapsulation is a collection of functions of a class and object. The “Food” class is an encapsulated form. It is achieved by specifying which class can use which members (private, public, protected) of an object.  
    
  **Polymorphism:** Polymorphism means existing in different forms. Inheritance is an example of Polymorphism. A base class exists in different forms as derived classes. Operator overloading is an example of Polymorphism in which an operator can be applied in different situations.

1. **What Is Difference Between Class And Interface?**

* Interface are similar to abstract classes. The difference between interfaces and abstract classes are: Interfaces cannot have properties, while abstract classes can. All interface methods must be public, while abstract class methods is public or protected.

1. **What Is Overloading?**

Function overloading is the ability to create multiple functions of the same name with different implementations. **Function overloading in PHP?** Function overloading in PHP is used to dynamically create properties and methods. These dynamic entities are processed by magic methods which can be used in a class for various action types. Function overloading contains same function name and that function performs different task according to number of arguments. For example, find the area of certain shapes where radius are given then it should return area of circle if height and width are given then it should give area of rectangle and others. Like other OOP languages function overloading can not be done by native approach. In PHP function overloading is done with the help of magic function \_\_call(). This function takes function name and arguments.

**Property and Rules of overloading in PHP:**

* All overloading methods must be defined as Public.
* After creating the object for a class, we can access a set of entities that are properties or methods not defined within the scope of the class.
* Such entities are said to be overloaded properties or methods, and the process is called as overloading.
* For working with these overloaded properties or functions, PHP magic methods are used.
* Most of the magic methods will be triggered in object context except \_\_callStatic() method which is used in a static context.

**Types of Overloading in PHP:** There are two types of overloading in PHP.

* Property Overloading
* Method Overloading

1. **What Is T\_PAAMAYIM\_NEKUDOTAYIM (Scope Resolution Operator ::) and give an example?**

The Scope Resolution Operator (also called Paamayim Nekudotayim) or in simpler terms, the double colon, is a token that allows access to [static](https://www.php.net/manual/en/language.oop5.static.php), [constant](https://www.php.net/manual/en/language.oop5.constants.php), and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

It's possible to reference the class using a variable. The variable's value can not be a keyword (e.g. self, parent and static).

Paamayim Nekudotayim would, at first, seem like a strange choice for naming a double-colon. However, while writing the Zend Engine 0.5 (which powers PHP 3), that's what the Zend team decided to call it. It actually does mean double-colon - in Hebrew!

Example #1 :: from outside the class definition

<?php  
class MyClass {  
    const CONST\_VALUE = 'A constant value';  
}  
  
$classname = 'MyClass';  
echo $classname::CONST\_VALUE;  
  
echo MyClass::CONST\_VALUE;  
?>

Three special keywords *self*, *parent* and *static* are used to access properties or methods from inside the class definition.

Example #2 :: from inside the class definition

<?php  
class OtherClass extends MyClass  
{  
    public static $my\_static = 'static var';  
  
    public static function doubleColon() {  
        echo parent::CONST\_VALUE . "\n";  
        echo self::$my\_static . "\n";  
    }  
}  
  
$classname = 'OtherClass';  
$classname::doubleColon();  
  
OtherClass::doubleColon();  
?>

When an extending class overrides the parent's definition of a method, PHP will not call the parent's method. It's up to the extended class on whether or not the parent's method is called. This also applies to [Constructors and Destructors](https://www.php.net/manual/en/language.oop5.decon.php), [Overloading](https://www.php.net/manual/en/language.oop5.overloading.php), and [Magic](https://www.php.net/manual/en/language.oop5.magic.php) method definitions.

Example #3 Calling a parent's method

<?php  
class MyClass  
{  
    protected function myFunc() {  
        echo "MyClass::myFunc()\n";  
    }  
}  
  
class OtherClass extends MyClass  
{  
    // Override parent's definition  
    public function myFunc()  
    {  
        // But still call the parent function  
        parent::myFunc();  
        echo "OtherClass::myFunc()\n";  
    }  
}  
  
$class = new OtherClass();  
$class->myFunc();  
?>

See also [some examples of static call trickery](https://www.php.net/manual/en/language.oop5.basic.php#language.oop5.basic.class.this).

1. **What are the differences between abstract classes and interfaces?**

|  |  |
| --- | --- |
| Interface Class | Abstract Class |
| Interface class supports multiple inheritance feature | Abstract class does not support multiple inheritances. |
| This does not contain a data member. | Abstract class does contain a data member. |
| The interface does not allow containers. | The abstract class supports containers. |
| An interface class only contains incomplete members which refer to the signature of the member. | Abstract class contains both incomplete(i.e. abstract) and complete members. |
| Since everything is assumed to be public, an interface class does not have access modifiers by default. | An abstract class can contain access modifiers within subs, functions, and properties. |
| Any member of an interface cannot be static. | Only a complete member of the abstract class can be static. |

1. **Define Constructor and Destructor?**

Constructors are special member functions for initial settings of newly created object instances from a class, which is the key part of the object-oriented concept in **PHP5**.  
Constructors are the very basic building blocks that define the future object and its nature. You can say that the Constructors are the blueprints for object creation providing values for member functions and member variables.  
Once the object is initialized, the constructor is automatically called. Destructors are for destroying objects and automatically called at the end of execution.  
In this article, we are going to learn about object-oriented concepts of constructors and destructors.   
Both are special member functions of any class with different concepts but the same name except destructors are preceded by a **~ Tilda operator**.  
**Syntax:** 

* **\_\_construct():**

function \_\_construct()

{

// initialize the object and its properties by assigning

//values

}

* **\_\_destruct():**

function \_\_destruct()

{

// destroying the object or clean up resources here

}

**Note:** The constructor is defined in the public section of the Class. Even the values to properties of the class are set by Constructors.  
**Constructor types:** 

* **Default Constructor:**It has no parameters, but the values to the default constructor can be passed dynamically.
* **Parameterized Constructor:** It takes the parameters, and also you can pass different values to the data members.
* **Copy Constructor:** It accepts the address of the other objects as a parameter.

1. **How to Load Classes in PHP?**

PHP load classes are used for declaring its object etc. in object oriented applications. PHP parser loads it automatically, if it is registered with **spl\_autoload\_register()**function. PHP parser gets the least chance to load class/interface before emitting an error.

**Syntax:**

spl\_autoload\_register(function ($class\_name) {

include $class\_name . '.php';

});

1. **How to Call Parent Constructor?**

* PHP 5 introduced the concept of constructor. Constructors are [magic method](https://webrewrite.com/magic-methods-php/) which call automatically when object is created. But when you define the constructor method both in Child and Parent class, it will not call until you explicitly reference it. So How to call parent class constructor in child class.

1. **Are Parent Constructor Called Implicitly When Create An ObjectOf Class?**

* Parent constructors are not called implicitly if the child class defines a constructor. In order to run a parent constructor, a call to parent::\_\_construct() within the child constructor is required.

1. **What Happen, If Constructor Is Defined As Private Or Protected?**

Public, private and protected are called access modifiers. Just like C++, PHP also have three access modifiers such as public, private and protected. The visibility of a property, a method or a constant can be defined by prefixing the declaration with these keywords. 

* If the class member declared as public then it can be accessed everywhere.
* If the class members declared as protected then it can be accessed only within the class itself and by inheriting child classes.
* If the class members declared as private then it may only be accessed by the class that defines the member.

1. **Define New-style Constructor & Old-style Constructor. Check whichOne Will Be Called?**



1. **Create Abstract class and method?**

When can define a class abstract using the abstract keyword. An class which is defined as abstract cannot be instantiated.

Following are some important points about abstract class and method:

1. An abstract class can have methods and properties just like any other normal class.
2. An abstract class cannot be instantiated, hence we need to create a child class which extends it, then we can create object of the child class.
3. If a class has even a single abstract method then the class should also be abstract.
4. An abstract method is just the declaration, where we provide name of the method and argument, while the body part is empty.
5. **Define 3 types of visibility of data member & member function.**

**Visibility** is a big part of **Object Oriented Programming**. It allows you to control where your class members can be accessed from, for instance to prevent a certain variable from being modified from outside the class. The default visibility is public, which means that the class members can be accessed from anywhere. This means that declaring the visibility is optional, since it will just fall back to public if there is no access modifier. For backwards compatibility, the old way of declaring a class variable, where you would prefix the variable name with the “var” keyword (this is from PHP 4 and should not be used anymore) will also default to public visibility.

PHP is pretty simple in this area, because it comes with only 3 different access modifiers: **private**, **protected** and **public**.

1. **Create a method which will never inherited?**

$class = new \ReflectionClass($model);

$modelMethods = $class->getMethods(\ReflectionMethod::IS\_PUBLIC);

$exclude = ['getSource', 'upsert', 'getUniqueKeyAttributes', 'beforeSave', 'columnMap', 'initialize', 'setId', 'getId'];

$CLASS = \_\_CLASS\_\_;

$props = array\_filter(array\_map(function(\ReflectionMethod $method) use ($exclude, $CLASS) {

if($method->class === $CLASS && !in\_array($method->name, $exclude)) {

if(strpos($method->name, 'get') === 0) {

return str\_replace('get', '', $method->name);

}

}

}, $props));

1. **What is the difference between Abstract class and Interface?**

## Interface Class

As we already know, an interface is actually defined by an interface keyword where all the methods are abstract. In addition to this, all the methods declared in this type of class must be declared in public which reflects the true nature of an interface.

Let's help to demonstrate that with an example:

1. **interface** Logger {
2. **public**
3. **function** execute();
4. }

As you can see above, in the interface, the method body is not defined. Only the name and the parameters are being defined. Now, let's move on to the abstract class.

Abstract Class

In PHP, an abstract class is one being partially implemented by any developer. It might contain at least one abstract method which is basically a method without any written code. It just contains the name and the parameters and has been marked as “abstract”.

By definition, an abstract class is simply a function definition whose purpose is to serve the programmer by telling them the method in question must be implemented in a child class.

Here is an example to demonstrate the abstract class:

1. **abstract** **class** AbstractClass {
2. **abstract** **protected**
3. **function** getValue();
4. **public**
5. **function** printOut() {
6. print $**this** - > getValue().
7. "\n";
8. }
9. }
10. **Create parent class for car and child class for car\_model and usecar functionality in car\_modelclass**
11. **Override the parent’s properties and methods in the child class?**

<?php

class A {

public $option;

public function \_\_construct() {

$this->option = array(1,2,3);

}

public function bb() {

$obj = new B;

$obj->aa();

print\_r($this->option);die;

}

}

class B extends A {

public function \_\_construct() {

parent::\_\_construct();

}

public function aa() {

$this->option[] = 4;

//print\_r($this->option);die;

}

}

$obj3 = new A;

$obj3->bb();

?>

returns Array ( [0] => 1 [1] => 2 [2] => 3 ) without elemnt from method aa()

1. **Prevent the child class from overriding the parent’s methods?**

I'm a beginner in PHP OOP. I want to prevent overriding of parent class properties when child class initiated. For example, I've got Parent and Childclasses as follows:

class Parent {

protected $array = [];

public function \_\_construct() {

}

public function add($value) {

$this->array[] = $value;

}

public function get() {

return $this->array;

}

}

class Child extends Parent {

public function \_\_construct() {

}

}

Firstly, I initiated Parentclass added 3 items to the array property:

$parent = new Parent;

$parent->add('a');

$parent->add('b');

$parent->add('c');

Then, I initiated Child class and added 1 item to the array property:

$child = new Child;

$child->add('d');

Actual result:

var\_dump($parent->show()); // outputs array('a', 'b', 'c')

var\_dump($child->show()); // outputs array('d')

Expected result:

var\_dump($parent->show()); // outputs array('a', 'b', 'c', 'd')

var\_dump($child->show()); // outputs array('a', 'b', 'c', 'd')

How can I do this? I tried this, but it didn't work:

class Child extends Parent {

public function \_\_construct() {

$this->array = parent::get();

}

}

1. **Declare classes and methods as abstract**

In order to declare a class as abstract, we need to prefix the name of the class with the abstract keyword.

See the following example:

1abstract class Car { }

We put the abstract methods that are also declared with the abstract keyword within the abstract class. Abstract methods inside an abstract class don't have a body, only a name and parameters inside parentheses.

In the example given below, we create a public abstract method, calcNumMilesOnFullTank(), that is the skeleton for methods that we will create in the child classes. Once created, these methods will return the number of miles a car can be driven on a tank of gas.

1 2 3 4// Abstract classes are declared with the abstract keyword, and contain abstract methods.

abstract class Car {

  abstract public function calcNumMilesOnFullTank();

}

It is important to know that once we have an abstract method in a class, the class must also be abstract.

1. **Can we have non abstract methods inside an abstract class? ExplainWith Example**

An abstract class can have non abstract methods. In fact, it can even have properties, and properties couldn't be abstract.

Let's add to our example the protected property, $tankVolume, and public method with the name of setTankVolume().

1 2 3 4 5 6 7 8 910111213abstract class Car {

  // Abstract classes can have properties

  protected $tankVolume;

  // Abstract classes can have non abstract methods

  public function setTankVolume($volume)

  {

    $this -> tankVolume = $volume;

  }

  // Abstract method

  abstract public function calcNumMilesOnFullTank();

}

1. **How to create child classes from an abstract class?**

* Since we cannot create objects from abstract classes, we need to create child classes that inherit the abstract class code. Child classes of abstract classes are formed with the help of the extends keyword, like any other child class. They are different in that they have to add the bodies to the abstract methods.

Let's create a child class with the name of Honda, and define in it the abstract method that it inherited from the parent, calcNumMilesOnFullTank().

1 2 3 4 5 6 7 8 9class Honda extends Car {

  // Since we inherited abstract method, we need to define it in the child class,

  // by adding code to the method's body.

  public function calcNumMilesOnFullTank()

  {

    $miles = $this -> tankVolume\*30;

    return $miles;

  }

}

We can create another child class from the Car abstract class and call it Toyota, and here again define the abstract method calcNumMilesOnFullTank() with a slight change in the calculation. We will also add to the child class its own method with the name of getColor() that returns the string "beige".

1 2 3 4 5 6 7 8 910111213class Toyota extends Car {

  // Since we inherited abstract method, we need to define it in the child class,

  // by adding code to the method's body.

  public function calcNumMilesOnFullTank()

  {

    return $miles = $this -> tankVolume\*33;

  }

  public function getColor()

  {

    return "beige";

  }

}

Let's create a new object, $toyota1, with volume of 10 Gallons, and let it return the number of miles on full tank and the car's color.

1 2 3 4$toyota1 = new Toyota();

$toyota1 -> setTankVolume(10);

echo $toyota1 -> calcNumMilesOnFullTank();//330

* echo $toyota1 -> getColor();//beige

1. **What are PHP Magic Methods/Functions? List them**

PHP magic methods are special methods that are called automatically when certain conditions are met. There are several magic methods in  PHP. Every magic method follows certain rules –

* Every magic method starts with a double underscore (  \_\_ ).
* They are predefined and neither can be created nor removed.
* Magic methods have reserved names and their name should not be used for other purposes.
* Magic methods are automatically called when certain criteria are met.

|  |  |  |
| --- | --- | --- |
| **Method Names** | **Return types** | **Condition of calling** |
| \_\_construct() | NaN | This method gets called automatically every time the object of a particular class is created.  The function of this magic method is the same as the constructor in any OOP language. |
| \_\_destruct() | NaN | As the name suggests this method is called when the object is destroyed and no longer in use.  Generally at the end of the program and end of the function. |
| \_\_call($name,$parameter) | Not mandatory | This method executes when a method is called which is not defined yet. |
| \_\_toString() | String | This method is called when we need to convert the object into a string.  For example:   echo $obj;  The $obj->\_\_toString(); will be called magically. |
| \_\_get($name) | NaN | This method is called when an inaccessible variable or non-existing variable is used. |
| \_\_set($name , $value) | NaN | This method is called when an inaccessible variable or non-existing variable is written. |
| \_\_debugInfo() | array | This magic method is executed when an object is used inside var\_dump() for debugging purposes. |

1. **Write program for Static Keyword in PHP?**

* <?php  
  class MyClass {  
    public static $str = "Hello World!";  
    
    public static function hello() {  
      echo MyClass::$str;  
    }  
  }  
    
  echo MyClass::$str;  
  echo "<br>";  
  echo MyClass::hello();  
  ?>

1. **Create multiple Traits and use it in to a single class?**

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the trait keyword:

### **Syntax**

<?php  
trait TraitName {  
  // some code...  
}  
?>

To use a trait in a class, use the use keyword:

### **Syntax**

<?php  
class MyClass {  
  use TraitName;  
}  
?>

1. **Write PHP Script of Object Iteration?**

PHP provides a way for objects to be defined so it is possible to iterate through a list of items, with, for example a [foreach](https://www.php.net/manual/en/control-structures.foreach.php) statement. By default, all [visible](https://www.php.net/manual/en/language.oop5.visibility.php) properties will be used for the iteration.

**Example #1 Simple Object Iteration**

<?php  
class MyClass  
{  
    public $var1 = 'value 1';  
    public $var2 = 'value 2';  
    public $var3 = 'value 3';  
  
    protected $protected = 'protected var';  
    private   $private   = 'private var';  
  
    function iterateVisible() {  
       echo "MyClass::iterateVisible:\n";  
       foreach ($this as $key => $value) {  
           print "$key => $value\n";  
       }  
    }  
}  
  
$class = new MyClass();  
  
foreach($class as $key => $value) {  
    print "$key => $value\n";  
}  
echo "\n";  
  
  
$class->iterateVisible();  
  
?>

1. **Create PHP script for objects passed in PHP – by reference or by value?**

## Introduction

In PHP**,**arguments to a function can be passed by value or passed by reference. By default, values of actual arguments are passed by value to formal arguments which become local variables inside the function. Hence, modification to these variables doesn't change value of actual argument variable.

When arguments are passed by reference, change in value of formal argument is reflected in actual argument variable because the former is a reference to latter. Thus pass by reference mechanism helps in indirectly manipulating data in global space. It also helps in overcoming the fact that a function can return only one variable.

## Pass by Value

In following example, two variables are passed to swap() function. Even though swapping mechanism takes place inside the function it doesn't change values of variables that were passed

<?php

function swap($arg1, $arg2){

   echo "inside function before swapping: arg1=$arg1 arg2=$arg2\n";

   $temp=$arg1;

   $arg1=$arg2;

   $arg2=$temp;

   echo "inside function after swapping: arg1=$arg1 arg2=$arg2\n";

}

$arg1=10;

$arg2=20;

echo "before calling function : arg1=$arg1 arg2=$arg2\n";

swap($arg1, $arg2);

echo "after calling function : arg1=$arg1 arg2=$arg2\n";

?>

## Pass by reference

In order to receive arguments by reference, variable used formal argument must be prefixed by & symbol. It makes reference to variables used for calling the function. Hence, result of swapping inside function will also be reflected in original variables that were passed

<?php

function swap(&$arg1, &$arg2){

   echo "inside function before swapping: arg1=$arg1 arg2=$arg2\n";

   $temp=$arg1;

   $arg1=$arg2;

   $arg2=$temp;

   echo "inside function after swapping: arg1=$arg1 arg2=$arg2\n";

}

$arg1=10;

$arg2=20;

echo "before calling function : arg1=$arg1 arg2=$arg2\n";

swap($arg1, $arg2);

echo "after calling function : arg1=$arg1 arg2=$arg2\n";

?>

1. **What is data modeling?**

### **Data modelling is the process of diagramming data flows. When creating a new or alternate database structure, the designer starts with a diagram of how data will flow into and out of the database. This flow diagram is used to define the characteristics of the data formats, structures, and database handling functions to efficiently support the data flow requirements. After the database has been built and deployed, the data model lives on to become the documentation and justification for why the database exists and how the data flows were designed.**

1. **Use of The $this keyword**

* **$this** is a reserved keyword in PHP that refers to the calling object. It is usually the object to which the method belongs, but possibly another object if the method is called statically from the context of a secondary object. This keyword is only applicable to internal methods.

1. **Declare and implement an interface and implement more than one interface in the same class.**
2. **How to implement the polymorphism principle**

Polymorphism is a Greek word that literally means many forms. In object-oriented programming, polymorphism is closely related to [inheritance](https://www.phptutorial.net/php-oop/php-inheritance/).

Polymorphism allows [objects](https://www.phptutorial.net/php-oop/php-objects/) of different classes to respond differently based on the same message.

To implement polymorphism in PHP, you can use either [abstract classes](https://www.phptutorial.net/php-oop/php-abstract-class/) or [interfaces](https://www.phptutorial.net/php-tutorial/php-interface/).

Polymorphism helps you create a generic framework that takes the different object types that share the same interface.

Later, when you add a new object type to the system, you don’t need to change the framework to accommodate the new object type as long as it implements the same interface.

By using polymorphism, you can reduce coupling and increase code reusability.

1. **Explain Scope resolution operator with example**

The Scope Resolution Operator (also called Paamayim Nekudotayim) or in simpler terms, the double colon, is a token that allows access to [static](https://www.php.net/manual/en/language.oop5.static.php), [constant](https://www.php.net/manual/en/language.oop5.constants.php), and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

It's possible to reference the class using a variable. The variable's value can not be a keyword (e.g. self, parent and static).

Paamayim Nekudotayim would, at first, seem like a strange choice for naming a double-colon. However, while writing the Zend Engine 0.5 (which powers PHP 3), that's what the Zend team decided to call it. It actually does mean double-colon - in Hebrew!

**Example #1 :: from outside the class definition**

<?php  
class MyClass {  
    const CONST\_VALUE = 'A constant value';  
}  
  
$classname = 'MyClass';  
echo $classname::CONST\_VALUE;  
  
echo MyClass::CONST\_VALUE;  
?>

1. **Access child class property to parent.**

abstract class A {

private $n = 1;

public function getN() {

$ref = new ReflectionProperty($this, 'n');

$ref->setAccessible(true);

echo $ref->getValue($this);

}

}

class B extends A {

protected $n = 2;

}

$b = new B;

echo $b->getN(); // 2