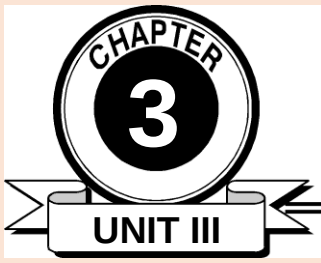

Apply Object Oriented Concepts in PHP

Syllabus

3.1 Creating Classes and Objects

3.2 Constructor and Destructor

3.3 Inheritance, Overloading and Overriding, Cloning Object

3.4 Introspection, Serialization.

3.1 Creating Classes and Objects

- The concepts of classes and objects, and the ways in which you can use them, are the fundamental ideas behind object oriented programming.
- An object oriented approach gives you some big benefits over procedural programming.

3.1.1 Advantages of OOP

- OOP makes it easy to map business requirements to code modules. Because your application is based on the idea of real world objects, you can often create a direct mapping of people, things, and concepts to classes.
- These classes have the same properties and behaviors as the real world concepts they represent, which helps you to quickly identify what code needs to be written and how different parts of the application need to interact.
- A second benefit of OOP is code reuse. You frequently need the same types of data in different places in the same application. For example, an application that manages hospital patient records might contain a class called Person. A number of different people are involved in patient care the patient, the doctors, the nurses, hospital administrators, and so on.
- By defining a class called Person that encompasses the properties and methods common to all of these people, you can reuse an enormous amount of code in a way that isn't always possible in a procedural programming approach.
- The opportunities for code reuse within a given application as well as across different projects. A well written Person class can easily be copied from one project to another with little or no change, instantly giving you all the rich functionality for dealing with information about people that you developed previously. This is one of the biggest benefits of an object oriented approach.

- Another OOP advantage comes from the modularity of classes. If you discover a bug in your Person class, or you want to add new features to the class or change the way it functions, you have only one place to go. All the functionality of that class is contained in a single PHP file. Any parts of the application that rely on the Person class are immediately affected by changes to it. Modularity is particularly important when working on large, complex applications.
- Applications written using OOP are usually relatively easy to understand the object oriented design of the application gives you a readymade framework within which you can develop new functionality.
- On larger projects, there are often many programmers with varying skill levels. Here, too, an object oriented approach has significant benefits over procedural code. Objects hide the details of their implementation from the users of those objects. The objects themselves are responsible for triggering changes to data or the state of the system.

3.1.2 Understanding basic OOP Concepts

The basic building blocks that you can use to create object oriented applications in PHP are :

Classes :

- In the real world, objects have characteristics and behaviors. A car has a color, a weight, a manufacturer, those are its characteristics. A car can accelerate, stop, signal for a turn, and sound the horn, those are its behaviors. Those characteristics and behaviors are common to all cars. Although different cars may have different colors, all cars have a color. A class called Car, would describe the characteristics and behaviors common to all cars.
- A class is a unit of code that describes the characteristics and behaviors of something, or of a group of things.
- Class is a collection of objects. Object has properties and behavior.
- Class is a user-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

Objects :

- An *object* is a specific instance of a class. For example, if you create a Car class, you might then go on to create an object called myCar that belongs to the Car class.
- Think of a class as a blueprint, or factory, for constructing an object.
- An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

Properties :

- In OOP terminology, the characteristics of a class or object are known as its *properties.*
- Properties are much like regular variables, in that they have a name and a value (which can be of any type).Some properties allow their value to be changed and others do not. For example, the Car class might have properties such as color and weight. Although the color of the car can be changed by giving it a new paint job, the weight of the car (without cargo or passengers) is a fixed value.

Methods :

- The behaviors of a class that is, the actions associated with the class are known as its *methods.* Methods are very similar to functions; you can define methods in PHP using the function statement. Like functions, some methods act on external data passed to them as arguments, but an object' s method can also access the properties of the object.

- For example, an accelerate method of the Car class might check the fuel property to make sure it has enough fuel to move the car. The method might then update the object's velocity property to reflect the fact that the car has accelerated.
- The methods of a class, along with its properties, are collectively known as *members* of the class.

3.1.3 Creating Classes and Objects in PHP

Q. How to create class in PHP?

Q. How to create an object in PHP?

- A class is a template for objects, and an object is an instance of class.
- A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods goes inside the braces :

Syntax :

```
<? Php
classname_of_class
 {
  // code goes here...
}
?>
```

- Once you defined a class, you can create objects based on the class. To create an object, you use the **new** keyword, followed by the name of the class that you want to base the object on. You can then assign this object to a variable, much like any other value.

Example :

```
<?php
class Car
 {
// Nothing to see here; move along
}
$Maruti = new Car();
$Honda = new Car();
print_r ($Maruti );       // Displays "Car Object ( )"
print_r ($Honda );            // Displays "Car Object ( )"
?>
```

Output :

```
Car Object ( )    Car Object ( )
```

- This code first defines the empty Car class as before and then creates two new instances of the Car class that is, two Car objects. It assigns one object to a variable called $Maruti and another to a variable called $Honda.

- Both objects are based on the same class, they are independent of each other, and each is stored in its own variable.
- Once the objects have been created, their contents are displayed using **print_r()**.

3.1.4 Creating and Using Properties

Q. Explain class visibility.

Class properties are very similar to variables; for example, an object' s property can store a single value, an array of values or even another object.

1. Understanding Property Visibility

The important concept of classes known as *visibility*. Each property of a class in PHP can have one of three visibility levels, known as public, private and protected :

- ***Public*** properties can be accessed by any code, whether that code is inside or outside the class. If a property is declared public, its value can be read or changed from anywhere in your script.
- ***Private*** properties of a class can be accessed only by code inside the class. So if you create a property that's declared private, only methods inside the same class can access its contents.(If you attempt to access the property outside the class, PHP generates a fatal error.)
- ***Protected*** class properties are a bit like private properties in that they can't be accessed by code outside the class, but there's one subtle difference: any class that inherits from the class can also access the properties.

2. Declaring Properties

To add a property to a class, first write the keyword public, private or protected depending on the visibility level you want to give to the property- followed by the property' s name (preceded by a $symbol) :

```
class MyClass
{
public $property1;          // This is a public property
private $property2;         // This is a private property
protected $property3;       // This is a protected property
}
```

You can also initialize properties at the time that you declare them, much like you can with variables :

```
class MyClass
{
public $rollno=123;
}
```

3. Accessing Properties

Once you have created a class property, you can access the corresponding object's property value from within your calling code by using the following syntax :

$object-> property;

– That is, you write the name of the variable storing the object, followed by an arrow symbol composed of a hyphen (-) and a greater than symbol (>), followed by the property name. (Note that the property name doesn't have a $ symbol before it.)

Example 1 :

```
<html>
<body>
<?php
class student
{
 // Properties
 public $name;
 public $city;

 // Methods
 function set_name($name)
{
   $this->name = $name;
 }
 function get_name() {
   return $this->name;
 }
}

$stu1 = new student();
$stu2 = new student();
$stu1->set_name('Vijay');
$stu2->set_name('Karan');

echo $stu1->get_name();
echo "<br>";
echo $stu2->get_name();
?>
</body>
</html>
```

Output :

```
Vijay
Karan
```

Example 2 :

```
<!DOCTYPE html>
<html>
<body>
<?php
class Fruit
{
 // Properties
 public $name;
 public $color;
 // Methods
 function set_name($name)
{
   $this->name = $name;
 }
 function get_name()
 {
   return $this->name;
 }
 function set_color($color)
{
   $this->color = $color;
 }
 function get_color() {
   return $this->color;
 }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
```

```
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " .$apple->get_color() . "<br><br>";

$pineapple = new Fruit();
$pineapple->set_name('Pine-Apple');
$pineapple->set_color('Brown');
echo "Name: " . $pineapple->get_name();
echo "<br>";
echo "Color: " .$pineapple->get_color();
?>
</body>
</html>
```

Output :

```
Name: Apple
Color: Red
Name: Pine-Apple
Color: Brown
```

Problems

1. Create a class as “Rectangle” with two properties as length and width. Calculate perimeter and area of rectangle.

```
<?php
class Rectangle
{
   // Declare  properties
   public $length = 0;
   public $width = 0;

   // Method to get the perimeter
   public function getPerimeter()
{
      return (2 * ($this->length + $this->width));
   }
```

```
    // Method to get the area
    public function getArea()
{
        return ($this->length * $this->width);
    }
}

// Create a new object from Rectangle class
$obj = new Rectangle;

// Get the object properties values
echo $obj->length . "<br>"; // 0utput: 0
echo $obj->width . "<br>"; // 0utput: 0

// Set object properties values
$obj->length = 30;
$obj->width = 20;

// Read the object properties values again to show the change
echo "Length:" . $obj->length . "<br>"; // 0utput: 30
echo "Width:" . $obj->width . "<br>"; // 0utput: 20

 // Call the object methods
echo "Perimeter:" . $obj->getPerimeter() . "<br>"; // 0utput: 100
echo "Area:" . $obj->getArea() . "<br>"; // Output: 600
?>
```

Output :

```
0
0
Length:30
Width:20
Perimeter:100
Area:600
```

2. Create a class as “Rectangle” with two properties as length and width. Calculate area of rectangle for 2 objects.

```
<?php
class Rectangle
{
   // Declare  properties
   public $length = 0;
   public $width = 0;

   // Method to get the area
   public function getArea()
  {
      return ($this->length * $this->width);
   }
}

// Create multiple objects from the Rectangle class
$obj1 = new Rectangle;
$obj2 = new Rectangle;

// Call the methods of both the objects
echo "Area of Object1:" . $obj1->getArea() . "<br>"; // Output: 0
echo "Area of Object2:" .$obj2->getArea(). "<br>"; // Output: 0

// Set $obj1 properties values
$obj1->length = 30;
$obj1->width = 20;

// Set $obj2 properties values
$obj2->length = 35;
$obj2->width = 50;

// Call the methods of both the objects again
echo "After calling”;
echo "Area of Object1:" .$obj1->getArea() . "<br>"; // Output: 600
```

```
echo "Area of Object2:" .$obj2->getArea() . "<br>"; // Output: 1750
?>
```

Output :

```
Area of Object1:0
Area of Object2:0
Area of Object1:600
Area of Object2:1750
```

3.1.5 This Keyword

- The $this keyword allows us to approach the class properties and methods from within the class.
- The $this keyword indicates that we use the class's own methods and properties and allows us to have access to them within the class's scope.
- $this is a pseudo-variable which is a reference to the current object.
- $this variable is used to call non-static method, if you are trying to call static method then it will throw the error that means $this variable is not available inside the static method.

Syntax :

- We can directly change the value of a property from outside of the class.

  $stu->roll=11;

- We can define a method in the class and call it to change the value of its own property.

```
public function changeroll()
{
    $this ->roll=12;
}
$stu->changeroll();
```

Example :

```
<?php
class demo
{
var $first=10;
var $second=20;
function  add( )
```

```
{
$add=$this->first + $this->second;
echo "Addition=".$add."<br/>";
}
function sub( )
{
$sub=$this->first - $this->second;
echo "Subtraction=".$sub."<br/>";
}
function summary()
{
$this->add( );
$this->sub( );
$res=$this->first*$this->second;
echo "Multiplication=".$res;
}
}
$obj= new demo();
$obj->summary();
?>
```

Output :

```
Addition=30
Subtraction=-10
Multiplication=200
```

Program using self keyword

```
<?php
   class Person
   {
      // first name of person
      public static $name;

      // public function to get value of name (getter method)
      public static function getName()
```

```
    {
        return self::$name;     // using self here
    }
  }
?>
```

Difference between self and this keyword

self	this
self keyword is not preceded by any symbol rather we can use as it is.	this keyword should be preceded with a $ symbol while referring class members.
To access class variables and methods using the self keyword, we use the scope resolution operator ::	-> symbol is used with $this variable as we have used with an object instance to access the property of that object.
It is used to refer the static members of the class.	It is used to access non-static members of the class with -> operator.
PHP self refers to the class members, but not for any particular object. This is because the static members(variables or functions) are class members shared by all the objects of the class.	Whereas, $this will refer the member variables and function for a particular Object.
Example : self::<class_member>	**Example :** $this-><class_member>

3.2 Constructor and Destructor

Q. Explain constructor & destructor in PHP?

- A constructor and a destructor are special functions which are automatically called when an object is created and destroyed.
- Constructors are special member functions for initialize variables on the newly created object instances from a class.
- When creating a new object, it's useful to set up certain aspects of the object at the same time. For example, you might want to set some properties to initial values, fetch some information from a database to populate the object, or register the object in some way.
- Similarly, when it's time for an object to disappear, it can be useful to tidy up aspects of the object, such as closing any related open files and database connections, or unsetting other related objects.
- An object's *constructor* method is called just after the object is created, and its *destructor* method is called just before the object is freed from memory.

3.2.1 Setting Up New Objects with Constructors

– Normally, when you create a new object based on a class, all that happens is that the object is brought into existence. (Usually you then assign the object to a variable or pass it to a function.) By creating a constructor method in your class, however, you can cause other actions to be triggered when the object is created.

To create a constructor, simply add a method with the special name **_ _construct()** to your class. (That ' s two underscores, followed by the word " construct, " followed by parentheses.) PHP looks for this special method name when the object is created; if it finds it, it calls the method.

Example :

```
class MyClass
{
function __construct()
{
echo "Welcome to PHP constructor. <br / > ";
}
}
$obj = new MyClass();
} // Displays "Welcome to PHP constructor."
```

The class, MyClass , contains a very simple constructor that just displays the message. When the code then creates an object from that class, the constructor is called and the message is displayed.

3.2.2 Constructor Types

1. Default Constructor

Q. Describe default constructor with suitable example

– It has no parameters, but the values to the default constructor can be passed dynamically.

– We can define constructor by using function _ _construct().

Example :

```
<?php
class Sample
{
   function Sample()
   {
      echo "Its a User-defined Constructor of the class Sample";
   }

   function __construct()
```

```
    {
        echo "Its a Pre-defined Constructor of the class Sample";
    }
}
$obj= new Sample();
?>
```

Output :

```
Its a Pre-defined Constructor of the class Sample
```

Note : In the case of Pre-defined Constructor(__construct) and user-defined constructor in the same class, the Pre-defined Constructor becomes Constructor while user-defined constructor becomes the normal method.

2. Parameterized Constructor

Q. Describe parameterized constructor with suitable example

- It takes the parameters, and also you can pass different values to the data members.
- Parameterized constructor is a constructor that accepts parameter and is normally used to initialize member variables at the time of object creation.
- The → operator is used to set value for the variables. In the constructor method, you can assign values to the variables during object creation.

Example 1 :

```
<?php
    class emp
        {
        // first name of person
        private $fname;
        // last name of person
        private $lname;

        // Constructor
        public function __construct($fname, $lname)
        {
            echo "Initialising the object...<br/>";
            $this->fname = $fname;
            $this->lname = $lname;
        }
```

```
    // public method to show name
    public function showName()
    {
        echo "My name is: " . "Mr.  " . $this->fname . " " . $this->lname;
    }
}

// creating class object
$sid = new emp("Amar ", "Joshi");
$sid->showName();

?>
```

Output :

```
Initializing the object...
My name is: Mr. Amar Joshi
```

Example 2 :

```
<?php
class Employee
{
    public $name;
    public $position;
    function __construct($name,$position)

    {
        // This is initializing the class properties
        $this->name=$name;
        $this->profile=$position;
      }
    function show_details()
    {
        echo $this->name." : ";
        echo "Your position is ".$this->profile."<br>";
    }
}
```

```
$employee_obj= new Employee("Prasad Koyande","developer");
$employee_obj->show_details();

$employee2= new Employee("Vijay Patil","Manager");
$employee2->show_details();
?>
```

Output :

```
Prasad Koyande : Your position is developer
Vijay Patil : Your position is Manager
```

If the PHP Class has a constructor, then at the time of object creation, the constructor of the class is called. The constructors have no Return Type, so they do not return anything not even void.

Advantages of using Constructors :

- Constructors provide the ability to pass parameters which are helpful in automatic initialization of the member variables during creation time.
- The Constructors can have as many parameters as required and they can be defined with the default arguments.
- Constructors encourage re-usability avoiding re-initializing whenever instance of the class is created.
- You can start session in constructor method so that you don't have to start in all the functions every time.
- Constructors can call class member methods and functions.
- Constructors can call other Constructors even from Parent class.

3.2.3 Cleaning Up Objects with Destructors

- A destructor is called when the object is destroyed.
- You have to manually dispose of objects you created, but in PHP, it's handled by the Garbage Collector, which keeps an eye on your objects and automatically destroys them when they are no longer needed.
- Destructors are useful for tidying up an object before it's removed from memory.
- Destructors don't have any types or return value. It is just called before de-allocating memory for an object or during the finish of execution of PHP scripts or as soon as the execution control leaves the block.
- For example, if an object has a few files open or contains data that should be written to a database, it's a good idea to close the files or write the data before the object disappears.
- You can create destructor methods in the same way as constructors, except that you use __destruct() function rather than __construct() :

Syntax :

```
function __destruct()
 {
```

```
// (Clean up here)
}
```

Note : Destructor can' t accept arguments. An object's destructor is called just before the object is deleted. This can happen because all references to it have disappeared or when the script exits, either naturally or because of an error of some sort. In each case, the object gets a chance to clean itself up via its destructor before it vanishes.

The destructor method is called when the PHP code is executed completely by its last line by using PHP **exit()** or **die() functions**.

Example 1

```
<?php
class MyDestClass
{
   function __construct()
   {
      print "In constructor<br>";
   }

   function __destruct()
   {
      print "Destroying " . __CLASS__ . "<br>";
   }
}
$obj = new MyDestClass();//object is not created
?>
```

Output :

```
In constructor
Destroying MyDestClass
```

Example 2 :

```
<?php
   class Person
      {
      // first name of person
      private $fname;
      // last name of person
      private $lname;
```

```
    // Constructor
    public function __construct($fname, $lname) {
        echo "Initialising the object...<br/>";
        $this->fname = $fname;
        $this->lname = $lname;
    }
    // Destructor
    public function __destruct()
    {
        // clean up resources or do something else
        echo "Destroying Object...";
    }

    // public method to show name
    public function showName() {
        echo "My name is: " . $this->fname . " " . $this->lname . "<br/>";
    }
}

// creating class object
$sid = new Person("Mr. Siddhesh", "Vaidya");
$sid->showName();
?>
```

Output :

```
Initialising the object...
My name is: Mr. Siddhesh Vaidya
Destroying Object...
```

Advantages of destructors :

- Destructor is used to objects to free up memory allocation, so that space is available for new objects or free up resources for other tasks.
- It effectively makes programs run more efficiently and is very useful as they carry out clean up tasks.

3.2.4 Comparison between Constructors and Destructors

Q. Differentiate between constructor and destructor

Constructors	Destructors
Accepts one or more arguments.	No arguments are passed.
Function name is _construct().	function name is _destruct()
Constructor is involved automatically when the object is created.	Destructor is involved automatically when the object is destroyed.
Used to initialize the instance of a class.	Used to de-initialize objects already existing to free up memory for new accommodation.
Constructors can be overloaded.	Destructors cannot be overloaded.
Used to initialize data members of class.	Used to make the object perform some task before it is destroyed.
Allocates memory.	It deallocates memory.
It is called each time a class is instantiated or object is created.	It is called automatically at the time of object deletion.
Multiple constructors can exist in a class.	Only one Destructor can exist in a class.

3.3 Inheritance, Overloading and Overriding, Cloning Object

Q. What is inheritance? Explain with suitable example. **(4 Marks)**

3.3.1 Inheritance

- Inheritance is a mechanism of extending an existing class by inheriting a class.
- We create a new sub class with all functionality of that existing class, and we can add new members to the new sub class.
- When we inherit one class from another we say that inherited class is a subclass and the class who has inherits is called parent class.
- In order to declare that one class inherits the code from another class, we use the **extends** keyword.

Syntax :

```
class Parent
{
  // The parent's class code
}
 class Child extends Parent
{
  // The  child can use the parent's class code
}
```

The child class can make use of all the non-private (public and protected) methods and properties that it inherits from the parent class. This allows us to write the code only once in the parent, and then use it in both the parent and the child classes.

Example :

```
<?php
class Shape
 {
  public $length;
  public $width;
  public function __construct($length, $width)
  {
    $this->length = $length;
    $this->width = $width;
  }
 }
class Rect extends Shape
{
  public $height;
  public function __construct($length, $width, $height)
   {
    $this->length = $length;
    $this->width = $width;
    $this->height = $height;
   }
  public function intro()
   {
    echo "The length is {$this->length}, the width is {$this->width}, and the height is {$this->height} ";
   }
}
$r = new Rect(10,20,30);
$r->intro();
?>
```

Output :

```
The length is 10, the width is 20, and the height is 30
```

- In the above example, all properties and functions are public hence they can access outside the class.
- What happens if function kept as protected outside the class? Just look at the next example.

```
<?php
class Shape
 {
  public $length;
  public $width;
  public function __construct($length, $width)
  {
    $this->length = $length;
    $this->width = $width;
  }
  public function intro()
   {
    echo "The length is {$this->length} and the width is {$this->width}.";
   }
 }
class Rect extends Shape
{
  public $height;
  public function __construct($length, $width, $height)
   {
    $this->length = $length;
    $this->width = $width;
    $this->height = $height;
   }

  protected function introduction ()
   {
    echo "The length is {$this->length}, the width is {$this->width}, and the height is {$this->height} ";
   }
}
$r = new Rect(10,20,30);
$r->intro();
```

```
$r->introduction ();//error
?>
```

Output :

```
The length is 10 and the width is 20.
Fatal error: Call to protected method Rect::introduction () from context "
```

- In above example, we see that if we try to call a protected method (introduction ()) from outside the class, we will receive an above error.
- Let's consider the another example, where we call the protected method (intro()) from inside the derived class that is introduction ().

```
<?php
class Shape
 {
  public $length;
  public $width;
  public function __construct($length, $width)
  {
    $this->length = $length;
    $this->width = $width;
  }
 protected function intro() //protected specifier
   {
    echo "The length is {$this->length} and the width is {$this->width}.";
   }
}
class Rect extends Shape
{
  public function introduction()
   {
    echo "The shape is rectangle. ";
$this->intro();// Call protected function from within derived class is OK
   }
}
$r = new Rect(10,20);
```

```
$r->introduction();
?>
```

Output :

```
The shape is rectangle. The length is 10 and the width is 20.
```

In above example, we called protected function intro () inside the public function introduction () in derived class.PHP only supports multilevel inheritance.

3.3.2 Types of Inheritance

Inheritance has three types : Single, multiple and multilevel Inheritance

3.3.2(A) Single Inheritance

Q. Explain single inheritance with suitable example

When a subclass is derived simply from its parent class then this mechanism is known as simple inheritance. In case of simple inheritance there is only a sub class and its parent class. It is also called **single inheritance** or **one level inheritance.**

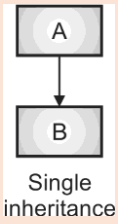

Fig. 3.3.1

Syntax :

```
class Parent
{
// The parent's class code
}
Class Child extends Parent
{
//The child can use the parent's code
}
```

Example :

```
<?php
class a
  {
    function dis1()
    {
```

```
        echo "Hello PHP <br>";
    }
}
class b extends a
{
    function dis2()
    {
        echo "PHP Programming <br>";
    }
}
$obj= new b();
$obj->dis1();
$obj->dis2();
?>
```

Output :

```
Hello PHP
PHP Programming
```

3.3.2(B) Multilevel Inheritance

Q. Explain multilevel inheritance with suitable example

When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance. The derived class is called the subclass or child class for its parent class and this parent class works as the child class for it's just above (parent) class. Multilevel inheritance can go up to any number of levels. This concept allows built a chain of classes as Grandfather -> father -> child.

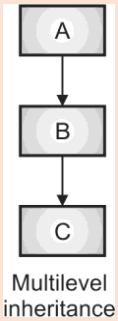

Fig. 3.3.2

Syntax :

```
Class A
{
```

```
}
Class B extends A
{
}
Class C extends B
{
}
```

Example :

```
<?php
class grandparent
   {
      function dis1()
      {
         echo "Grand-Parents <br>";
      }
   }
class parents extends grandparent
   {
      function dis2()
      {
         echo "Parents <br>";
      }
   }
class child extends parents
   {
      function dis3()
      {
         echo "Child <br>";
      }
   }

   $obj= new child();
   $obj->dis1();
   $obj->dis2();
```

```
$obj->dis3();
?>
```

Output :

```
Grand-Parents
Parents
Child
```

3.3.2(C) Hierarchical Inheritance

Q. Describe hierarchical inheritance with suitable example

Hierarchical inheritance consists of a single parent class and that parent class is inherited by multiple child class. In hierarchical type of inheritance, **one class is extended by many subclasses**. It is **one-to-many** relationship. In Simple sentence, Hierarchical inheritance is *"creating one or more child classes from the parent class".*

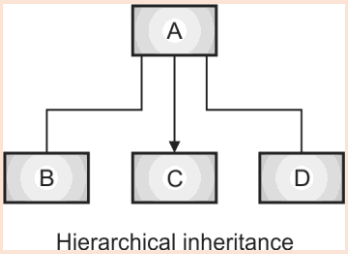

Fig. 3.3.3

Example :

```
<?php
class a
{
public function function_a()
{
echo "class A";
}
}

class b extends a
{
public function function_b()
{
echo "class B";
}
}
```

```
class c extends a
{
public function function_c()
{
echo "class C";
}
}

echo c::function_c(). "<br>";
echo c::function_a();

?>
```

Output

```
class C
class A
```

3.3.2(D) Multiple Inheritance

Q. Describe multiple inheritance with suitable example.

The mechanism of inheriting the features of more than one base class into a single class is known as multiple inheritance.PHP doesn't support multiple inheritance but by using Interfaces in PHP or using Traits in PHP instead of classes, we can implement it.

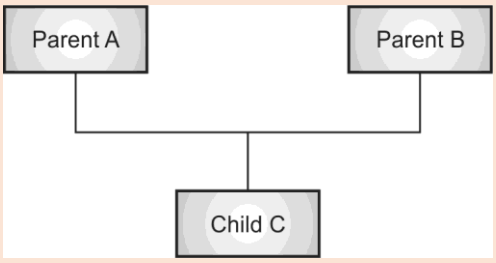

Fig. 3.3.4

Traits :

- Traits define the actual implementation of each method within each class, traits are just chunks of code injected in a class in PHP.
- Traits are not interfaces at all. Traits can define both static members and static methods.
- It helps developers to reuse methods freely in several independent classes in different class hierarchies.
- Traits reduces the complexity, and avoids problems associated with multiple inheritance.

– PHP does not allow multiple inheritance. So Traits is used to fulfil this gap by allowing us to reuse same functionality in multiple classes.

Syntax :

```
class child_class_name
{
   use trait_name;
   ...
   ...
child_class functions
}
```

Example :

```
<?php
// Class A i.e. Parent A
class A {
  public function disp1()
   {
     echo "Parent-A <br>";
  }
}

// Trait B i.e. Parent B
trait B
{
  public function disp2()
  {
     echo " Parent-B <br>";
  }
}

class C extends A
 {
   use B;
   public function disp3()
  {
```

```
    echo "\nChild-C";
  }
}
$obj = new C();
$obj->disp1();
$obj->disp2();
$obj->disp3();
?>
```

Output :

```
Parent-A
Parent-B
Child-C
```

Interface :

Q. Explain interface in php with suitable example?

- An Interface allows the users to create programs, specifying the public methods that a class must implement, without involving the complexities and details of how the particular methods are implemented.
- An Interface is defined just like a class but with the class keyword replaced by the interface keyword and just the function prototypes.
- The interface contains no data variables.
- An interface consists of methods that have no implementations, which means the interface methods are abstract methods.
- All the methods in interfaces must have public visibility scope.
- Interfaces are different from classes as the class can inherit from one class only whereas the class can implement one or more interfaces.
- Interface enables you to model multiple inheritance.

Syntax-1 :	**Syntax-2 :**
interface (using class along with interface) class child_class_name extends parent_class_name implements interface_name1, ...	Interface (using multiple interface) class child_class_name implements interface_name1, interface_name2, ...

Example :
<?php
// Class A i.e. Parent A
class A
{
	Example :
	<?php
	// interface A i.e. Parent A
	interface A
	{

```
 public function disp1()
  {
    echo "Parent-A <br>";
 }
}
// Interface B i.e Parent B
interface B
{
 public function disp2();
}
class C extends A implements B
 {
function disp2()
 {
   echo " Parent-B <br>";
 }
 public function disp3()
 {
    echo "\nChild-C";
 }
}
$obj = new C();
$obj->disp1();
$obj->disp2();
$obj->disp3();
?>
```

Output :

```
Parent-A
        Parent-B
        Child-C
```

```
 public function disp1();
}
// Interface B i.e Parent B
interface B
{
 public function disp2();
}
class C implements A,B
 {
function disp1()
  {
    echo "Parent-A <br>";
 }
function disp2()
 {
   echo " Parent-B <br>";
 }
 public function disp3()
 {
    echo "\nChild-C";
 }
}
$obj = new C();
$obj->disp1();
$obj->disp2();
$obj->disp3();
?>
```

Output :

```
Parent-A
        Parent-B
        Child-C
```

3.3.3 Concrete Class

Q. Describe concrete class?

- The class which implements an interface is called the Concrete Class.
- It must implement all the methods defined in an interface.
- Interfaces of the same name can't be implemented because of ambiguity error. Just like any class, an interface can be extended using the **extends** operator.
- It can be inherited by another class, but not an abstract class.
- It can implement any number of interfaces.

Example :

```
<?php

interface InterfaceName{

   public function method1();
   public function method2();

}

class ClassName implements InterfaceName
{
    public function method1()
{
    echo "Method1 Called" . "<br>";
  }
    public function method2()
{
     echo "Method2 Called". "<br>";
  }
}
$obj = new ClassName;
$obj->method1();
$obj->method2();
?>
```

Output :

```
Method1 Called
Method2 Called
```

3.3.4 Abstract Class and Abstract Methods

Q. Describe abstract class with suitable example?

- An **abstract class** is a class that has at least one **abstract method**.
- Abstract classes are special because they can never be instantiated.
- An abstract class is identified by having the keyword abstract in front of the class name
- It can inherit one or more interfaces using the implements keyword.
- It can inherit no more than one abstract class using the extends keyword.
- An abstract class cannot inherit from a non-abstract class.
- Any abstract methods which are inherited, either from an interface or another abstract class, do not need to be defined as signatures and their implementations are only required when inherited by a non-abstract class.
- It can contain any number of constants, variables and methods
- **Abstract methods** can only have names and arguments, and no other code. Thus, we cannot create objects out of abstract classes. Instead, we need to create child classes that add the code into the bodies of the methods, and use these child classes to create objects.
- An abstract method is defined as a method signature without any method body - i.e. it has no implementation.

Syntax :

```
abstractclassclass_name{ }
or
abstract class class_name
{
  abstract public function func_name();
}
```

Example :

```
<?php
// Abstract class
abstract class Base
{
   // This is abstract function
   abstract function printdata();
}
class Derived extends base
```

```
{
   function printdata()
  {
     echo "Derived class";
   }
}
$obj = new Derived;
$obj->printdata();
?>
```

Output :

```
Derived class
```

Difference between abstract class and interface

Q. Differentiate between abstract class and interface.

Interface	Abstract Class
Interface support multiple inheritance	Abstract class does not support multiple inheritance.
Interface doesn't contains data member.	Abstract class contains data member
Interface doesn't contains constructor.	Abstract class contains constructor.
An interface contains only incomplete member (signature of member)	An abstract class contains both incomplete (abstract) and complete member.
An interface cannot have access modifiers by default everything is assumed as public.	An abstract class can contain access modifiers for the subs, functions, properties.
Members of interface cannot be static.	Only complete member of abstract class can be static.

3.3.5 Overloading and Overriding

Q. Describe method overloading and method overriding

- Function overloading is a feature of object oriented programming that allows to create more than one methods with the same name but different number of arguments/parameter.
- Overloading in PHP creates properties and methods dynamically.
- 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.
- In PHP function overloading is done with the help of magic function __call(). This function takes function name and arguments.

3.3.5(A) 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.

3.3.5(B) Types of Overloading in PHP

There are Two types of Overloading in PHP

1. Property Overloading
2. Method Overloading

Property Overloading

- PHP property overloading is used to create dynamic properties in the object context.
- For creating these properties no separate line of code is needed. A property associated with a class instance, and if it is not declared within the scope of the class, it is considered as overloaded property.

Following operations are performed with overloaded properties in PHP.

- Setting and getting overloaded properties.
- Evaluating overloaded properties setting.
- Undo such properties setting

To perform the operations, we should define following appropriate magic methods.

Methods	Description
__set()	Triggered while initializing overloaded properties.
__get()	Triggered while using overloaded properties with PHP print statements.
__isset()	Invoked when we check overloaded properties with isset() function
__unset():	Invoked on using PHP unset() for overloaded properties.
__call()	Triggered while invoking overloaded methods in the object context.
__callStatic()	Triggered while invoking overloaded methods in static context.

Example :

```
<?php
class Toys
{
private $str;
public function __set($name,$value)
```

```
{
$this->str[$name] = $value;
}
public function __get($name)
{
echo "Overloaded Property name = " . $this->str[$name] . "<br/>";
}
public function __isset($name)
{
      if(isset($this->str[$name]))
        {
      echo "Property \$$name is set.<br/>";
        }
      else
        {
      echo "Property \$$name is not set.<br/>";
        }
}

public function __unset($name)
{
unset($this->str[$name]);
echo "\$$name is unset <br>";
}
}

$objToys = new Toys;
/* setters and getters on dynamic properties */
$objToys->overloaded_property = "CAR";
echo $objToys->overloaded_property . "\n\n";
/*Operations with dynamic properties values*/
isset($objToys->overloaded_property);
unset($objToys->overloaded_property);
isset($objToys->overloaded_property);
```

```
?>
```

Output :

```
Overloaded Property name = CAR
Property $overloaded_property is set.
$overloaded_property is unset Property $overloaded_property is not set.
```

- In above example, the dynamic property is created. While initializing this dynamic property, __set() is invoked with name and value pair to be initialized as the name and value of class property array element $str.
- Added to that, isset() and unset() with overloaded property will trigger __isset() and __unset() magic methods. After unset(), if we call isset(), then it will print "property not set" message to the browser.

Use of Magic Function :

The following example shows how method overloading performs using two function with same name.

```
<?php
class text
{
  public functiondisplay($parameter1)
{
  echo "Hello world!!";
 }
public functiondisplay($parameter1,$parameter2)
{
echo "Hello India!!";
}
}
$obj = new text;
$obj->display('Hello');      // It will show fatal error
?>
```

Output :

```
Fatal error: Cannot redeclare text::display()
```

Note : From above example we can say that in PHP overloading with same name function can't be possible. Therefore, with the help of magic function overloading is done in PHP.

Following is an example of overloading with the help of magic methods :

```
<?php
// PHP program to explain function overloading in PHP
// Creating a class of type shape
class shape
{
```

```
   // __call is magic function which accepts function name and arguments
   function __call($name_of_function, $arguments)
    {
      // It will match the function name
if($name_of_function == 'area')
          {
          switch (count($arguments))
            {
             // If there is only one argument area of circle
             case 1:
                return 3.14 * $arguments[0];
             // IF two arguments then area is rectangle;
             case 2:
                return $arguments[0]*$arguments[1]; z
             }
         }
   }
}
// Declaring a shape type object
$s = new Shape;
  // Function call
echo("Area of circle:" . $s->area(4));
echo "<br>";
// calling area method for rectangle
echo ("Area of Rectangle:" . $s->area(5, 2)) . "<br>";
?>
```

Output :

```
Area of circle:12.56
Area of Rectangle:10
```

Q. Write a php program to performs method overloading to performs addition of numbers depending on the arguments passing.

```
<?php
class addition
{
```

```
    // __call is magic function which accepts function name and arguments
    function __call($name_of_function, $arguments)
     {
        // It will match the function name
if($name_of_function == 'add')
           {
           switch (count($arguments))
             {
              // If there is only one argument area of circle
             case 1:
                return $arguments[0];
             // IF two arguments then area is rectangle;
             case 2:
                return $arguments[0] + $arguments[1];
             case 3:
                return $arguments[0] + $arguments[1] + $arguments[2];
             }
          }
    }
}
// Declaring a addition type object
$s = new addition;
echo("Return One value:" . $s->add(10))."<br>";
echo ("Addition of two numbers:" . $s->add(10,20)) . "<br>";
echo ("Addition of three numbers:" . $s->add(10,20,30)) . "<br>";
?>
```

Output :

```
Return One value:10
Addition of two numbers:30
Addition of three numbers:60
```

3.3.6 Method Overriding

Q. Describe method overriding in PHP with suitable example?

– In function overriding, both parent and child classes should have same function name and number of arguments.

- It is used to replace parent method in child class.
- The purpose of overriding is to change the behavior of parent class method.
- The two methods with the same name and same parameter is **called overriding**.
- Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

Example :

```
<?php
class Shape
 {
  public $length;
  public $width;
  public function __construct($length, $width)
{
    $this->length = $length;
    $this->width = $width;
  }
  public function intro()
   {
    echo "The length is {$this->length} and the width is {$this->width}.";
   }
}
class square extends Shape
{
  public $height;
  public function __construct($length, $width, $height)
   {
    $this->length = $length;
    $this->width = $width;
    $this->height = $height;
   }
  public function intro()
   {
    echo "The length is {$this->length}, the width is {$this->width}, the height is {$this->height} ";
  }
}
```

```
$s = new square(10,30, 50);
$s->intro();
?>
```

Output :

```
The length is 10, the width is 30, the height is 50
```

3.3.7 Final Keyword

- If we define a method with final, then it prevents us to override the method.
- The final keyword is used only for methods and classes.
- Final methods prevent method overriding and final classes prevents inheritance

Final Class :

- A class declared as final cannot be extended in future.
- When a class is declared as final that class should not be inherited due to some security or other reasons.
- A final class can contain final as well as non-final methods. But there is no use of final methods in class when class is itself declared as final because inheritance is not possible.
- The final keyword can be used to prevent class inheritance or to prevent method overriding.

Example to prevent class inheritance :

```
<?php
final class A
{
public function disp()
{
echo "Inside the Parent class";
}
}
class B extends A
{
function disp()
{
echo "Inside the Child class";
}
}
$obj=new B();
$obj->disp();
```

```
?>
```

Output :

```
Fatal error: Class B may not inherit from final class (A) in C:\xampp\htdocs\Sample\test.php on line 15
```

Final Method :

- When a method is declared as final then overriding on that method can not be performed.
- The following example shows how to prevent method overriding :

```
<?php
class A
{
final public function disp()
{
echo "Inside the Parent Class";
}
}
class B extends A
{
function disp()
{
echo "Inside the Child Class";
}
}
$obj=new B();
$obj->disp();
?>
```

Output :

```
Fatal error : Cannot override final method A::disp() in C:\xampp\htdocs\Sample\test.php on line 15.
```

3.3.8 Cloning Object

- Object cloning is the process to create a copy of an object.
- An object copy is created by using the magic method __clone().
- If you don't use the __clone() method, the internal objects of the new object will be references to the same object in memory as the internal objects of the original object that was cloned.

- An object's __clone() method cannot be called directly. When an object is cloned, PHP will perform a shallow copy of all of the object's properties. Any properties that are references to other variables will remain references.

Object copy or by reference copy

Let us understand this by other example :

```
<?php
class sample
{
public $variable1;
private $variable2;
function __construct($variable1, $variable2)
{
$this->variable1 = $variable1;
$this->variable2 = $variable2;
}
}
$variable1 = new sample("Siddheh" , "Software Developer");
$variable2 = $variable1;                    //Copy of the object
$variable1->variable1 = "Sunil";
print_r($variable1);
echo "<br>";
print_r($variable2);
?>
```

Output :

```
sample Object ( [variable1] => Sunil [variable2:sample:private] => Software Developer )
sample Object ( [variable1] => Sunil [variable2:sample:private] => Software Developer )
```

- All the above example show object copy is object by reference not by value. If you want object copy by value then this example will not work so to overcome this feature object cloning is introduced.
- We already seen that object cloning is copy of object by reference not by value but if we want to assign it by value then object cloning can be called. We implement object cloning by **clone** keyword.
- Let us see same above example using clone keyword :

```
<?php
class sample
{
public $v1;
```

```
private $v2;
function __construct($variable1, $variable2)
{
$this->variable1 = $variable1;
$this->variable2 = $variable2;
}
}
$v1 = new sample("Rohan" , "Software Developer");
$v2 = $v1;                  //Copy of the object
$v3 = clone $v1;

print_r($v1);
echo "<br>";
print_r($v2);
echo "<br>";
print_r($v3);
?>
```

Output :

```
sample Object ( [v1] => [v2:sample:private] => [variable1] => Rohan [variable2] => Software Developer )
sample Object ( [v1] => [v2:sample:private] => [variable1] => Rohan [variable2] => Software Developer )
sample Object ( [v1] => [v2:sample:private] => [variable1] => Rohan [variable2] => Software Developer )
```

In above example $variable3 is clone of $variable1. So $variable3 is completely separate from object $variable1 and $variable2.

3.3.9 Deep Copy and Shallow Copy

Shallow copy or shallow clone :

- This is a bit-wise copy of an object.
- In Shallow Copy, a new object is created. The new object is an exact copy of the values in the original object. In case if any of the fields of the object are references to other objects, only reference addresses get copied.
- In PHP the "Copy" keyword is performed as a Shallow Copy by default. In PHP5 all objects are assigned by reference. It calls the object's "__clone()" method. In a Shallow Copy any change of a reference member affects both the methods.
- Simply makes a copy of the reference to A into B. Think about it as a copy of A's Address. So, the addresses of A and B will be the same i.e. they will be pointing to the same memory location i.e. data contents.

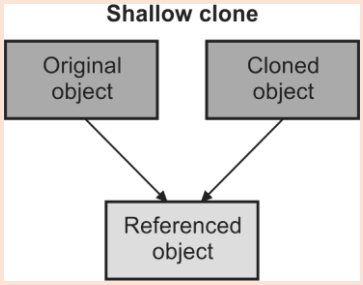

Fig. 3.3.5

Deep copy or Deep Clone :

- This is a process where the data is actually completely copied.
- In a Deep Copy everything is duplicated and all values are copied into a new instance.
- A deep copy copies all fields and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.
- A great advantage of deep copy is that the A & B do not depend on each other but this process is relatively slower and more expensive.
- We know that the B points to object A's memory location in shallow copy. However, in the deep copy, all things in object A's memory location get copied to object B's location.
- Simply makes a copy of all the members of A, allocates memory in a different location for B and then assigns the copied members to B to achieve deep copy.

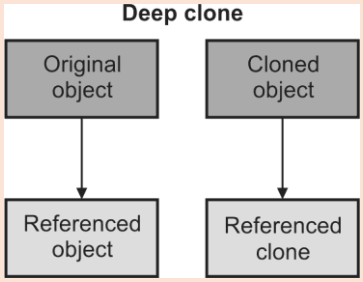

Fig. 3.3.6

Example of Shallow copy

```
<?php
class Obj
{
   public $id;
   public $size;
   public $color;
   function __construct($id, $size, $color)
     {
      $this->id = $id;
```

Example of Deep copy

```
<?php
class Obj
{
   public $id;
   public $size;
   public $color;
   function __construct($id, $size, $color)
     {
      $this->id = $id;
```

```
            $this->size = $size;
            $this->color = $color;
        }
        function __clone()
            {
            $green = $this->color->green;
            $blue = $this->color->blue;
             $red = $this->color->red;
            $this->color = new Color($green, $blue, $red);
        }
    }
    class Color
    {
        public $green;
        public $blue;
        public $red;
        function __construct($green, $blue, $red)
            {
            $this->green = $green;
            $this->blue = $blue;
            $this->red = $red;
        }
    }
    $color = new Color(23, 42, 223);
    $obj1 = new Obj(23, "small", $color);
    $obj2 = clone $obj1;
    $obj2->id++;
    $obj2->color->red = 255;
    $obj2->size = "big";
    echo "<pre>";print_r($obj1);
    echo "<pre>";print_r($obj2);
    ?>
```

```
            $this->size = $size;
            $this->color = $color;
        }
    }
    class Color
    {
        public $green;
        public $blue;
        public $red;
        function __construct($green, $blue, $red)
            {
            $this->green = $green;
            $this->blue = $blue;
            $this->red = $red;
        }
    }
    $color = new Color(23, 42, 223);
    $obj1 = new Obj(23, "small", $color);
    $obj2 = clone $obj1;

    $obj2->id++;
    $obj2->color->red = 255;
    $obj2->size = "big";

    echo "<pre>";print_r($obj1);
    echo "<pre>";print_r($obj2);
    ?>
    Output:
    Obj Object
    (
        [id] => 23
        [size] => small
```

```
Output:
Obj Object
(
    [id] => 23
    [size] => small
    [color] => Color Object
        (
            [green] => 23
            [blue] => 42
            [red] => 223
        )

)
Obj Object
(
    [id] => 24
    [size] => big
    [color] => Color Object
        (
            [green] => 23
            [blue] => 42
            [red] => 255
        )
)
```

```
    [color] => Color Object
        (
            [green] => 23
            [blue] => 42
            [red] => 255
        )

)
Obj Object
(
    [id] => 24
    [size] => big
    [color] => Color Object
        (
            [green] => 23
            [blue] => 42
            [red] => 255
        )

)
```

3.4 Introspection, Serialization

3.4.1 Introspection

Q. What is introspection? Explain with suitable example to express usage of introspection?

- Introspection in PHP offers the useful ability to examine an object's characteristics, such as its name, parent class (if any) properties, classes, interfaces and methods.
- PHP offers a large number functions that you can use to accomplish the task.

- Following are the functions to extract basic information about classes such as their name, the name of their parent class and so on.
- In-built functions in PHP Introspection :

Function	Description
class_exists()	Checks whether a class has been defined.
get_class()	Returns the class name of an object.
get_parent_class()	Returns the class name of an object's parent class.
is_subclass_of()	Checks whether an object has a given parent class.
get_declared_classes()	Returns a list of all declared classes.
get_class_methods()	Returns the names of the class' methods.
get_class_vars()	Returns the default properties of a class.
interface_exists()	Checks whether the interface is defined.
method_exists()	Checks whether an object defines a method.

Example 1 :

```
<?php
 if (class_exists('Demo'))
  {
     $demo = new Demo();
     echo "This is Demo class.";
   }
 else
  {
     echo "Class does not exist";
   }
?>
```

Output :

```
Class does not exist
```

In above example, there is no declaration for Class "Demo". If you declare the class as "Demo" you will get the output as "This is Demo class."

Example 2

```
<?php
 class Demo
```

```
{
// just declare the class as "Demo'
}
if (class_exists('Demo'))
  {
    $demo = new Demo();
    echo "This is Demo class";
  }
else
  {
    echo "Class does not exist";
  }
?>
```

Output :

```
This is Demo class
```

Example 3 :

Another example based on Introspection

```
<?php
class Derived
{
   public function details()
   {
      echo "I am a Derived(super) class for the Child(sub) class. <BR>";
   }
}
class sub extends Derived
{
   public function details()
 {
      echo "I'm " .get_class($this) , " class.<BR>";
      echo "I'm " .get_parent_class($this) , "'s child.<BR>";
   }
}
//details of parent class
```

```
if (class_exists("Derived"))
{
   $der = new Derived();
   echo "The class name is: " .get_class($der) . "<BR>";
   $der->details();
}
//details of child class
if (class_exists("sub"))
{
   $s = new sub();
   $s->details();

   if (is_subclass_of($s, "Derived"))
   {
      echo "Yes, " .get_class($s) . " is a subclass of Derived.<BR>";
   }
   else
   {
      echo "No, " .get_class($s) . " is not a subclass of Derived.<BR>";
   }
}
```

Output :

```
The class name is: Derived
I am a Derived (super) class for the Child(sub) class.
I'm sub class.
I'm Derived's child.
Yes, sub is a subclass of Derived.
```

Q. Declare a class as “Test” with three user defined functions. List name of the class and functions declared in the class “Test”.

```
<?php
  class Test
 {
    function testing_one()
```

```
    {
      return(true);
    }
    function testing_two()
    {
      return(true);
    }
    function testing_three()
    {
      return(true);
    }
  }
 //Class "Test" exist or not
if (class_exists('Test'))
  {
    $t = new Test();
    echo "The class is exist. <br>";

  }
else
  {
    echo "Class does not exist. <br>";
  }

//Access name of the class
   $p= new Test();
   echo "Its class name is " ,get_class($p) , "<br>";
//Aceess name of the methods/functions
   $method = get_class_methods(new Test());

  echo "<b>List of Methods:</b><br>";
foreach ($method as $method_name)
  {
    echo "$method_name<br>";
```

```
  }
?>
```

Output :

```
The class is exist.
Its class name is Test
List of Methods:
testing_one
testing_two
testing_three
```

Example based on interface_exists

```
<?php

interface MyInterfaceName
{
   public function method1();
   public function method2();
 }
 class MyClassName implements MyInterfaceName
{
     public function method1()
   {
       echo "Method1 Called" . "<BR>";
   }
     public function method2()
   {
       echo "Method2 Called". "<BR>";
   }
}

if (interface_exists('MyInterfaceName'))
   {
   echo "Interface exists.<br>";
   }
   else
   {
```

```
    echo "Interface does not exists.<br>";
    }
 $obj = new MyClassName;
$obj->method1();
$obj->method2();
?>
```

Output :

```
Interface exists.
Method1 Called
Method2 Called
```

3.4.2 Serialization

Q. Explain serialization in PHP

- Serialization is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form.
- Serializing an object means converting it to a byte stream representation that can be stored in a file. Serialization in PHP is mostly automatic, it requires little extra work from you, beyond calling the serialize () and unserialize( ) functions.

Serialize() :

- The serialize() converts a storable representation of a value.
- The serialize() function accepts a single parameter which is the data we want to serialize and returns a serialized string.
- A serialize data means a sequence of bits so that it can be stored in a file, a memory buffer or transmitted across a network connection link. It is useful for storing or passing PHP values around without losing their type and structure.

Syntax :

```
serialize(value1);
```

unserialize() : unserialize() can use string to recreate the original variable values i.e. converts actual data from serialized data.

Syntax :

```
unserialize(string1);
```

Example-1 of both serialize() and unserialize() functions :

```
<?php
$s_data= serialize(array('Welcome', 'to', 'PHP'));
print_r($s_data . "<br>");
$us_data=unserialize($s_data);
```

```
print_r($us_data);
?>
```

Output :

```
a:3:{i:0;s:7:"Welcome";i:1;s:2:"to";i:2;s:3:"PHP";}
Array ( [0] => Welcome [1] => to [2] => PHP )
```

- The first character of the string indicates the type of the serialized variable. Here, the letter "a" thus indicates that we have to serialize an array (array).
- The second number 3 indicates the number of elements in the array. In between parentheses, we find all the elements of the array.
- The first letter of an element group indicates the type of the index in the table followed by its value. Here we have an array indexed numerically serialized, so each index is an integer ("i") whose value is between 0 and 2 inclusive. If we had used an associative array, the indexes would have become strings ("s").
- We then read each stored value, type ("s"), its length (7) and the value stored in the table ("Welcome").

Example 2 based on array for serialize () and unserialize() :

```
<?php
$var = array( 'Welcome', 568, array(1, 'two'),  'VP');          // a complex array
$string = serialize($var);                                      // serialize the above data
print_r($string ."<br>");                                       // unserializing the data in $string
$newvar = unserialize($string);                                 // printing the unserialized data
print_r($newvar);
 ?>
```

Output :

```
a:4:{i:0;s:7:"Welcome";i:1;i:568;i:2;a:2:{i:0;i:1;i:1;s:3:"two";}i:3;s:2:"VP";}
Array ( [0] => Welcome [1] => 568 [2] => Array ( [0] => 1 [1] => two ) [3] => VP )
```

3.4.3 Magic Methods with PHP Object Serialization

- The following magic's methods will be invoked automatically while performing serialization with PHP object context. This automatic invocation will be triggered by using a pair of PHP function serialize()/unserialize() used in PHP Object Serialization.
- **sleep() :** This will be invoked on calling PHP serialize() function. It will return object's property array on cleaning PHP class objects before serialization. The __sleep function in PHP is useful when we have very large objects in our program and we do not want to save the object completely.
- **wakeup() :** It will do the reverse work to restore objects properties and resources on invoking unserialize(). __wakeup() of PHP is used to re-establish the database connections that may have been lost during any phenomenon.

Review Questions

Q.1 Define the relationship between a class and an object?

Q.2 Explain in detail the types of inheritance supports by PHP

Q.3 What is the purpose of $this & extends?

Q.4 What are serializable objects in PHP?

Q.5 How to access properties and methods explain with example and also explain $this variable?

Q.6 How to declare and access properties of class.

Q.7 List types of constructors? Explain any one with suitable example.

Q.8 Define term constructor and destructor.

Q.9 List types of inheritance? Explain any one with suitable example.

Q.10 How multiple inheritance is achieved in PHP.

Q.11 Explain method overloading concept in PHP with suitable example.

Q.12 Explain method overriding concept in PHP with suitable example.

Q.13 Explain types of overloading in PHP.

Q.14 Explain magic methods in PHP.

Q.15 Describe final class and final method.

Q.16 Explain cloning object in PHP.

❑❑❑