

UNIT - 3

PHP WITH OOPS

Prepared By: Prof. Disha H. Parekh

PHP – CREATING A CLASS

- OOP is all about creating modular code, so our object oriented PHP code will be contained in dedicated files that we will then insert into our normal PHP page using php 'includes'.
- In this case, all our OO PHP code will be in the PHP file: class1.php
- OOP revolves around a construct called a 'class'. Classes are the cookie-cutters / templates that are used to define objects.
- You define your own class by starting with the keyword 'class' followed by the name you want to give your new class.
 - <?php
 - **class** classname
 - {
 - Variable declaration and code;
 - }
 - ?>

PHP – ADDING DATA TO A CLASS

- Classes are the blueprints for php objects – more on that later. One of the big differences between functions and classes is that a class contains both data (variables) and functions that form a package called an: ‘object’.
- When you create a variable inside a class, it is called a ‘property’.
 - `<?php`
 - **class** classname
 - {
 - **var** variable_name;
 - }
 - `?>`

PHP – ADDING METHODS TO A CLASS

- “Getters” and “Setters” are object methods that allow you to control access to a certain class variables / properties.
- Sometimes, these functions are referred to as “mutator methods”.
- A “getter” allows to you to retrieve or “get” a given property.
- A “setter” allows you to “set” the value of a given property.
- Let us look at an example:

PHP – OOPS : EXAMPLE

- I have made a file named class1.php and wrote the below mentioned code:

```
<?php
```

```
    class person
```

```
    {
```

```
        var $name;
```

```
        function set_name($new_name)
```

```
        {
```

```
            $this->name = $new_name;
```

```
        }
```

```
        function get_name()
```

```
        {
```

```
            return $this->name;
```

```
        }
```

```
    }
```

```
?>
```

PHP – \$THIS

- The \$this is a built-in variable (built into all objects) which points to the current object or in other words, \$this is a special self-referencing variable.
- We use \$this to access properties and to call other methods of the current class.

PHP – OOPS CODE INSIDE A MAIN PHP PAGE

- You would never create your PHP classes directly inside your main php pages – that would help defeat the purposes of object oriented PHP in the first place!
- Instead, it is always best practice to create separate php pages that only contain your classes. Then you would access your php objects/classes by including them in your main php pages with either a php 'include' or 'require'.
- For our example:
- I have created another file called index_cl.php and wrote

```
<?php  
    include("class1.php");  
?>
```

PHP – OOPS CREATING AN OBJECT

- Classes are the blueprints/templates of php objects. Classes don't actually become objects until you do something called: instantiation.
- When you instantiate a class, you create an instance of it, thus creating the object.
- In other words, instantiation is the process of creating an **instance** of an object in memory.
- For our example inside the same index_cl.php file write:

```
<?php
```

```
    $disha = new person();
```

```
    $dhp = new person;
```

```
?>
```


PHP – OOPS SETTING AN OBJECT PROPERTIES

- After we have created two different objects of our class, now we have to set their properties using the setters method that we used in the class1.php
- To set the properties, with the same example in the same file index_cl.php:

```
$disha->set_name("Disha Parekh");  
$dhp->set_name("Disha H. Parekh");
```

PHP – OOPS ACCESSING AN OBJECT'S DATA

- To access the data of an object, we use getter method of the class.
- While accessing data of the object we use -> arrow sign, which is named as???

```
echo "Disha's full name: " . $disha->get_name()."<br>";
```

```
echo "DHP's full name: " . $dhp->get_name();
```

PHP – OOPS : EXAMPLE

- Finally the second file index_cl.php looks like:

```
<?php include("class1.php"); ?>
```

```
<?php
```

```
    $disha = new person();
```

```
    $dhp = new person;
```

```
    $disha->set_name("Disha Parekh");
```

```
    $dhp->set_name("Disha H. Parekh");
```

```
    echo "Disha's full name: " . $disha->get_name()."<br>";
```

```
    echo "DHP's full name: " . $dhp->get_name();
```

```
?>
```

PHP – OOPS : EXAMPLE

- Finally execute the index_cl.php to check the output.
- Disha's full name: Disha Parekh
DHP's full name: Disha H. Parekh

PHP – OOPS CONSTRUCTORS

- All objects can have a special built-in method called a 'constructor'.
- Constructors allow you to initialise your object's properties when you instantiate (create) an object.
- If you create a `__construct()` function PHP will automatically call the `__construct()` method/function when you create an object from your class.
- The 'construct' method starts with two underscores (`__`) and the word 'construct'.

PHP – OOPS CONSTRUCTORS : EXAMPLE

- For our example:

<?php

```
class person {  
    var $name;  
    function __construct($persons_name)      {  
        $this->name = $persons_name;  
    }  
    function set_name($new_name)              {  
        $this->name = $new_name;  
    }  
    function get_name()                       {  
        return $this->name;  
    }  
}
```

?>

CREATE AN OBJECT WITH CONSTRUCTORS

- Now that we've created a constructor method, we can provide a value for the \$name property when we create our person objects.
- You 'feed' the constructor method by providing a list of arguments (like you do with a function) after the class name.
- For our example in index_con.php file:

```
<?php include("class_constructor.php"); ?>
```

```
<?php
```

```
    $abc = new person("A Book Check");
```

```
    echo "ABC's full name: ".$abc->get_name();
```

```
?>
```

PHP – OOPS – ACCESS MODIFIERS

- One of the fundamental principles in OOP is 'encapsulation'. The idea is that you create cleaner better code, if you restrict access to the data structures (properties) in your objects.
- You restrict access to class properties using something called 'access modifiers'. There are 3 access modifiers:
 - public
 - private
 - protected
- 'Public' is the default modifier.

PHP – OOPS – ACCESS MODIFIERS : EXAMPLE

- Example to be shown in lab.

PHP – OOPS – INHERITANCE

- **Inheritance** is a mechanism of extending an existing class by inheriting a class we create a new class with all functionality of that existing class, and we can add new members to the new class.
- When we inherit one class from another we say that inherited class is a subclass and the class who has inherit is called parent class.
- We declare a new class with additional keyword **extends**.
- Note : PHP only supports multilevel inheritance.

PHP – OOPS – INHERITANCE

Types of PHP Inheritance

- Generally, inheritance has three types, *single*, *multiple* and *multi-level* inheritance. But, PHP supports *single* inheritance and multi-level inheritance. That means the subclass will be derived from a single parent class. Even though PHP is not supporting any multiple inheritance, we can simulate it by using PHP Interfaces.
- In PHP, inheritance can be done by using *extends* keyword, meaning that, we are extending the derived class with some additional properties and methods of its parent class. The syntax for inheriting a class is as follows.

PHP – OOPS – INHERITANCE

- Example to be shown in lab

PHP – OOPS – ABSTRACTION

- Abstraction is a way of hiding information .
- In abstraction there should be at least one method which must be declared but not defined.
- The class that inherit this abstract class need to define that method.
- There must be a abstract keyword that must be return before this class for it to be abstract class.
- This class cannot be instantiated .
- Only the class that implement the methods of abstract class can be instantiated.
- There can be more than one methods that can left undefined.

PHP – OOPS – ABSTRACTION – EXAMPLE

```
<?php
```

```
abstract class a {  
    abstract function b();  
    public function c() {  
        echo "Can be used as it is"; } }  
}
```

```
class m extends a {  
    public function b() {  
        echo "Defined function b<br/>";  
    } }  
}
```

```
$tClass = new m();
```

```
$tClass->b();
```

```
$tClass->c();
```

PHP – OOPS – ABSTRACTION – EXAMPLE

```
<?php
```

```
abstract class A {  
    abstract function f1();  
    function f2() {  
        echo "hello"; }  
}
```

```
class B extends A {  
    function f1() {  
        echo "hi"; }  
}
```

```
$ob=new B();
```

```
$ob->f1();
```

```
PREPARED BY: DISHA H. PAREKH, DCS, INDUS  
UNIVERSITY
```

```
?>
```

PHP – OOPS – INTERFACE

- The class that is fully abstract is called interface.
- Any class that implement this interface must use implements keyword and all the methods that are declared in the class must be defined here. otherwise this class also need to be defined as abstract.
- Multiple inheritance is possible only in the case of interface.

PHP – OOPS – INTERFACE – EXAMPLE

```
<?php  
  
interface A {  
    function f1(); }  
  
interface B {  
    function f2(); }  
  
class C implements A,B {  
    function f1() {  
        echo "hi"; }  
  
    function f2() {  
        echo "hello"; } }  
  
$ob=new C();  
  
$ob->f1();  
  
$ob->f2();
```

PHP – OOPS – FINAL

- The **final** keyword prevents child classes from overriding a method by prefixing the definition with final.
- It means if we define a method with final then it prevent us to override the method.

PHP – OOPS – FINAL – EXAMPLE

```
<?php  
class A {  
    final function show($x,$y) {  
        $sum=$x+$y;  
        echo "Sum of given no=".$sum; } }  
class B extends A {  
    function show($x,$y) {  
        $mult=$x*$y;  
        echo "Multiplication of given no=".$mult; } }  
$obj= new B();  
$obj->show(100,100);  
?>
```