**Week - 2**

**Task -1**

**'Variables Scope' in PHP, explain it using an example.**

Scope can be defined as the range of availability a variable has to the program in which it is declared.

Scopes of variable in PHP:

1. **Local variable:**

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function.

| <?php  $x = 4;    function assignx()  {  $x = 0;  echo "\$x inside function is $x \n";  }   assignx();   echo "\$x outside of function is $x \n"; ?> |
| --- |

Output:

| $x inside function is 0  $x outside of function is 4 |
| --- |

In the above example, ‘$x’ defined inside the ‘assignx()’ function is a local variable.

1. **Function Parameters**

Function parameters are declared after the function name and inside parentheses.

| <?php  // multiply numbers and return it to the caller  function multiply ($a , $b)  {  $product = $a \* $b;  return $product;  }   $product = multiply(5,6);   echo "Product of 5 and 6 is $product\n" ?> |
| --- |

Output

| Product of 5 and 6 is 30 |
| --- |

In the above example ‘$x’ and ‘$y’ are function parameters.

1. **Global Variables**

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword GLOBAL in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.

| <?php  $globalvar = 15;   function addit()  {  GLOBAL $globalvar;  echo "Global variable value inside local function $globalvar\n";  $globalvar++;   }   addit();   echo "Global variable value inside global function $globalvar\n" ?> |
| --- |

Output:

| Global variable value 15 Global variable value 16 |
| --- |

In the above example, ‘$globalvar’ is a global variable.

1. **Static Variables**

In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

| <?php   function keep\_track()  {  STATIC $count = 0;  echo ++$count."\n";  }  keep\_track();  keep\_track();  keep\_track(); ?> |
| --- |

Output:

| 1 2 3 |
| --- |

In the above example, ‘$count’ is a static variable.

**Task - 2**

**List some key differences between 'echo' and 'print'.**

The differences between ‘echo’ and ‘print’ are listed below:

1. ‘echo’ has no return value while ‘print’ has a return value of 1 so it can be used in expressions.
2. ‘echo’ can take multiple parameters (although such usage is rare) while ‘print’ can take one argument.
3. ‘echo' is marginally faster than ‘print’.

**Task - 3**

**Explain is\_int(), is\_numeric() and is\_integer() using examples.**

1. **is\_int():** It is used to check the type of a variable is integer.
2. **is\_integer():** alias of is\_int(). It is also used to test whether the type of the specified variable is an integer or not.
3. **is\_numeric():** It is used to check whether the variable is a number or a numeric string.

**Example 1:**

| <?php  $variable = "12345";   echo is\_int($variable)."\n";//returns false  echo is\_integer($variable)."\n";//returns false  echo is\_numeric($variable)."\n";//return true ?> |
| --- |

**Example 2:**

| <?php  $variable = 12345;   echo is\_int($variable)."\n";//returns true  echo is\_integer($variable)."\n";//returns true  echo is\_numeric($variable)."\n";//return true ?> |
| --- |

**Task - 4**

**Explain using examples,ways to convert 'int' variable to 'string'.**

Any variable can be converted to a string using the explicit ‘**(string)**’ cast or the ‘**strval()**’ function. String conversion is automatically done in the scope of an expression where a string is needed. This happens when using the echo or print functions, or when a variable is compared to a string.

Converting integer variables to string:

1. **Using strval() function:**

| // Function call  $var = 5; $var\_string = strval($var); // $var\_string = "5";  var\_dump($var\_string); |
| --- |

1. **Explicit cast:**

| // Type cast $var = 5; $var\_string = (string)$var; // $var\_string = "5"; var\_dump($var\_string); |
| --- |

1. **Implicit cast**
2. **During Assignment:**

| // Implicit cast during assignment $var = 0; $var\_string='5'; $var = $var\_string; // $var = "5"; var\_dump($var); |
| --- |

1. **During String concatenation:**

| $var = 5; echo "I'd like ".$var." waffles"; // I'd like 5 waffles |
| --- |

1. **During Inline variable parsing:**

| $var = 5; echo "I'd like {$var} waffles"; // = I'd like 5 waffles |
| --- |

**Task - 5**

**Create a function to floor decimal numbers with any provided precision.**

ex: convert(2.99999,2), convert(199.99999,4)

**Question: Should we use ‘ceil’ or ‘floor’ inside the convert() function?**

| <?php  //Create a function to floor decimal numbers with any provided precision.  //ex: convert(2.99999,2), convert(199.99999,4)    function convert($var\_dec, $precision)  {  return floor($var\_dec\*(10\*\*$precision))/10\*\*$precision;  }   echo   "  <form method='POST' action='/Week2/testCodes/task5.php' >  <label for='num'>Decimal value</label>  <input type='number' name='num' step='any'></input><br><br>  <label for='precision'>Precision value</label>  <input type='number' name='precision'></input><br><br>  <input type='submit' name='submit' value='Submit'>  </form>  ";  if(isset($\_POST['submit']))   {   $num = $\_POST['num'];  $precision = $\_POST['precision'];   $result = convert($num, $precision);   echo "Decimal value $num <br>";  echo "Precision $precision <br>";  echo "Precised value = $result <br>";  }  ?> |
| --- |

**Task - 6**

**Return the lowest and the largest integer from an array of numbers (don't use min() or max()).**

| <?php  function sortarray($num\_array)  {  sort($num\_array);  return $num\_array;  }  function getmin($sorted\_array)  {  return $sorted\_array[0];  }  function getmax($sorted\_array)  {  return $sorted\_array[count($sorted\_array)-1];  }   $num\_array = array(50,10,9,70,13,25,100,75);  $sorted\_array = sortarray($num\_array);   echo "Maximum number in array is ".getmax($sorted\_array)."\n";  echo "Minimum number in array is ".getmin($sorted\_array)."\n"; ?> |
| --- |

**Task - 7**

**Write a code that creates random numbers from a certain range of values.**

| <?php  //Code that creates random numbers from a certain range of values  function random\_generator($first\_num,$last\_num)  {  $list = range($first\_num,$last\_num);  shuffle($list);  return $list;  }   $result = random\_generator(10,20);    echo "Unique random numbers within 10 and 20 \n";   foreach ($result as $item)  {  echo $item."\n";  }  ?> |
| --- |

**Task - 8**

**Write a code to list the numbers between any two numbers that are divisible by 2 (without using conditional statements).**

| <?php  //list the numbers between any two numbers that are divisible by 2 without using conditional statement  function getEvenNumbers($first\_num , $last\_num)  {  $first\_num%2==0 ? $even\_numbers=range($first\_num,$last\_num,2) : $even\_numbers=range($first\_num+1,$last\_num,2);  return $even\_numbers;  }   function printArray($array\_list)  {  foreach ($array\_list as $item)  {  echo $item."\n";  }  }   $even\_numbers = getEvenNumbers(11,55);  echo "Even numbers between 11 and 55\n";  printArray($even\_numbers);  $even\_numbers2 = getEvenNumbers(50,100);  echo "Even numbers between 50 and 100\n";  printArray($even\_numbers2); ?> |
| --- |

**Task - 9**

**Write a code explaining the use of the 'switch' statement.**

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

We use switch-case over if-else condition handlers when we have a single variable to be checked and that variable is bound to limited constant values. Using switch-case over if-else in this case makes the condition block less chaotic and more readable.

Example 1

| $day =2;  switch($day)  {  case '1':  case '7':  echo 'Weekend \n';  break;   case '2':  echo 'Monday \n';  break;   case '3':  echo 'Tuesday \n';  break;    case '4':  echo 'Wednesday \n';  break;   case '5':  echo 'Thursday \n';  break;   case '6':  echo 'Friday \n';  break;  } |
| --- |

Example 2

| $day = 2;   if ($day == 1 || $day == 7)  {  echo "Weekend \n";   }  else if ($day == 2)  {  echo "Monday \n";  }  else if ($day == 3)  {  echo "Tuesday \n";  }  else if ($day == 4)  {  echo "Wednesday \n";  }  else if ($day == 5)  {  echo "Thursday \n";  }  else if ($day == 6)  {  echo "Friday \n";  } |
| --- |

As we can see in above examples, the switch block seems much less chaotic and thus more readable. There are a few reasons for that.

The first is that we’re checking for the value of a single variable, and that’s exactly what a switch was made for. The OR check is way more elegant in the switch statement.

The second reason is that as the logic chain gets larger and with five else-ifs, we can say it’s pretty large, the if-block tends to get really messy. Part of this has to do with the number of curly brackets that we need to use to get this piece of code to work. When it comes to large logic chains, the switch statement seems to be much more readable.

**Task - 10**

**Write a code to display dates in the provided format?**

**// input: 'Sep 20 2021' and '09092021'**

**// output: 2021-09-20 and 'Sep-09-2021'**

| <?php  //Code to display dates in the provided format  // input: 'Sep 20 2021' and '09092021'  // output: 2021-09-20 and 'Sep-09-2021'   function format1($date)  {  $date = strtotime($date);  $date = date('Y-m-d',$date);  return $date;  }   function format2($date)  {  $date = strtotime($date);  $date = date('M-d-Y',$date);  return $date;  }   $date1 = 'Sep 20 2021';  $date1 = format1($date1);  echo "Formatted date 1 ".$date1."\n";   $date2 = '09.09.2021';  $date2 = format2($date2);  echo "Formatted date 2 ".$date2."\n";  ?> |
| --- |

**Task - 11**

**Write a code to calculate a number of days between any two dates.**

| <?php  //Code to calculate a number of days between any two dates  function calcDays($sdate, $edate)  {  $diff = abs($edate-$sdate);  $days = $diff/(60\*60\*24);  return $days;  }   $sdate = '1997-09-24';  $edate = '2021-01-01';  $sdate = strtotime($sdate);  $edate = strtotime($edate);   $days = calcDays($sdate, $edate);   echo "Number of days between given dates is $days \n";  ?> |
| --- |

**Task - 12  
Write a code to calculate the current age of a person in 'days'.**

| <?php  //Code to calculate the current age of a person in 'days'   function calcDays($sdate, $edate)  {  $diff = abs($edate-$sdate);  $days = $diff/(60\*60\*24);  return $days;  }   $birth\_date = '1997-09-24';  $edate = date('Y-m-d');  $birth\_date = strtotime($birth\_date);  $edate = strtotime($edate);   $days = calcDays($birth\_date, $edate);   echo "You are $days days old \n";  ?> |
| --- |

**Task - 13**

**Write a code to show the number of days of the previous and current month.**

| <?php  //Code to show the number of days of the previous and current month   $today = strtotime(date('Y-m-d'));  $one\_month\_ago = strtotime("-1 months",$today);   echo "There are ".date('t',$today)." days in ".date('M',$today)."\n";   echo "There were ".date('t',$one\_month\_ago)." days in ".date('M',$one\_month\_ago)."\n";  ?> |
| --- |

**Task -14**

**Write a code explaining 'overloading' and 'overriding'.**

**Method Overloading in PHP**

You cannot overload PHP functions. Also, class method overloading is different in PHP than in many other languages. We have to utilize PHP's magic methods \_\_call() to achieve method overloading. If an object of a class calls a method that doesn't exist then \_\_call() is called and after that we can define this method in such a way that can handle functions with different number of parameters or different return type as shown in example below:

| <?php  //Code to explain 'overloading'   class Shape  {  public $area;  function \_\_call($name\_of\_function, $arguments)  {  if ($name\_of\_function=='calcArea')  {  switch (count($arguments))  {  case 1:  $area\_of\_circle = 3.14\*($arguments[0]\*\*2);  $this->area = $area\_of\_circle;  break;  case 2:  $area\_of\_rectangle = $arguments[0] \* $arguments[1] ;  $this->area = $area\_of\_rectangle;  break;  }  }  }  }   $circle = new Shape();  $circle->calcArea(0.4);  echo "Area of circle with radius 0.4 m is ".$circle->area."\n";   $rectangle = new Shape();  $rectangle->calcArea(5,6);  echo "Area of rectangle with length 5 m and breadth 6 m is ".$rectangle->area."\n"; ?> |
| --- |

In the above example, objects(‘circle’ and ‘rectangle’) of the same class are calling the function(‘area’) with a different number of arguments(area(0.4) and area(5,6)).

**Method Overriding in PHP**

| <?php  //Code to explain 'overriding'  class Bases  {  public function printSomething()  {  echo "Function to be overridden \n";  }  }   class Derived extends Bases  {  public function printSomething()  {  echo "Function overridden \n";  }  }   $base = new Bases();  $derived = new Derived();   $base->printSomething();  $derived->printSomething();   ?> |
| --- |

**Task - 15**

**Write a code, explaining similarities/differences using 'abstract classes' and 'interfaces'.**

Similarities between ‘abstract classes’ and ‘interfaces’ are:

1. Interface can not be instantiated. Same way, you can not instantiate abstract class.
2. They both provide a certain level of abstraction while developers follow an OOP approach to develop an app.

The differences between ‘abstract classes’ and ‘interfaces’ are:

1. Interface support multiple inheritance while abstract class does not support multiple inheritance.

| interface Interface\_A { } interface Interface\_B { } interface Interface\_C { }  interface MyInterface extends Interface\_A, Interface\_B, Interface\_C { } |
| --- |

The above code will work fine but the below code will give an error.

| class Class\_A { } class Class\_B { }  abstract class MyAbstractClass extends Class\_A, Class\_B { } |
| --- |

1. Interface doesn’t contain data member while abstract class contains data member.

| interface MyInterface {  public $foo = null; } |
| --- |

The above code will not work while below example works fine.

| abstract class MyAbstractClass {  public $foo = null; } |
| --- |

1. Interfaces do not need to have constructors while abstract classes contain constructors.

| interface MyInterface {  public function \_\_construct(); } |
| --- |

The above code is redundant as it does not make sense to have a constructor. You can have common initialization code in constructor of an abstract class or you can define constructor as below.

| abstract class MyAbstractClass {  abstract public function \_\_construct(); } |
| --- |

1. An interface contains only incomplete member (signature of member) while an abstract class contains both incomplete (abstract) and complete member.

| interface Animal  {  function prey();  function run();  } |
| --- |

| abstract class Animal {  // child classes must implement this  abstract function prey();   public function run() {  echo 'I am running!';  } }  class Dog extends Animal {  public function prey() {  echo 'Cat !';  } }  class Cat extends Animal {  public function prey() {  echo 'Rat !';  } }  $dog = new Dog(); $cat = new Cat();  $dog->prey(); // Cat ! $cat->prey(); // Rat !  $dog->run(); // I am running! $cat->run(); // I am running! |
| --- |

1. An interface cannot have access modifiers, by default everything is assumed as public while an abstract class can contain access modifiers for the methods, properties.
2. Members of the interface can not be static. Only complete members of the abstract class can be static.

| interface MyInterface  { static function foo(); } |
| --- |

The above code is possible because php is a loosely coupled language but it does not make sense to make methods inside the interface static.

| abstract class MyAbstractClass {  public static $foo = null;   public static function foo() {} } |
| --- |

**Task - 16**

**Create an example that deploys 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.

**Multiple Inheritance using interface:**

| <?php   interface father{   function message();   }    interface mother{   function dayin($my);   }   interface fam extends father,mother{   function cook($name);   }   class test implements fam{   function dayin($my){   Echo "My name is:".$my."\n";   }   function message(){   Echo "Interface inheritance, to achieve two abstraction methods\n";  }   function cook($name){   echo "People who often cook often:".$name."\n";  }   }   $t=new test();   $t->message();   $t-> DAYIN ("Kush");  $t-> Cook ("Mom"); ?> |
| --- |

**Multiple Inheritance using Traits:**

The trait is a type of class which enables multiple inheritance.

Classes, case classes, objects, and traits can all extend no more than one class but can extend multiple traits at the same time.

| <?php   // trait ABC  trait ABC {  public function sayhello() {  echo "Hello";  }  }    // trait DEF  trait DEF {  public function sayfor() {  echo " There";  }  }    class Sample {  use ABC;  use DEF;  public function message() {  echo "\nhuman";  }   }    $test = new Sample();  $test->sayhello();  $test->sayfor();  $test->message(); ?> |
| --- |

**Task - 17**

**What is 'namespaces' in PHP, explain using an example?**

Namespaces are qualifiers that solve two different problems:

1. They allow for better organization by grouping classes that work together to perform a task
2. They allow the same name to be used for more than one class

**Defining namespace**

| <?php namespace ABC; // code which is defined here belongs to the ABC namespace ?> |
| --- |

**Defining sub-namespace**

| <?php // example.php namespace ABC\DEF;  // define a class class ExampleClass {  public function \_\_construct() {  echo "Fully qualified class name: ".\_\_CLASS\_\_."\n";    }  }  // define a function function exampleFunction() {  echo "Fully qualified function name: ".\_\_FUNCTION\_\_."\n"; }  // define a constant const EXAMPLE\_CONSTANT = 50; ?> |
| --- |

**Using namespace**

| <?php // use\_namespace.php use ABC\DEF as NS; use ABC\DEF\ExampleClass as ExampleClass;  require "example.php";  // instantiate the Tutorial class with namespace alias TC $obj1 = new NS\ExampleClass();  // instantiate the Tutorial class with class alias Tut $obj2 = new ExampleClass();  // call the function echo NS\exampleFunction();  // display the constant echo NS\EXAMPLE\_CONSTANT |
| --- |

**Develop a Web application "User Registration System".**

Please go to link : <https://github.com/gauravkarki78/User-Registration-System>

All codes are available in :

<https://github.com/gauravkarki78/Week-2-Codes>