FUNCTIONS

Functions

- A function is a self-contained block of code that performs a specific task.

- PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task.

```
<?php
function sendmsg() {
  echo "Hey there";
}
sendmsg();
?>
```

OUTPUT:

Hey there

PHP User-Defined Functions

- PHP also allows you to define your own functions.
- It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately form main program.
- Here are some advantages of using functions:

Advantages

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.

- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.

Advantages(contd.)

- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.

- **Functions can be reused in other application** — Because a function is separated from the rest of the script, it's easy to reuse the same function in other applications just by including the php file containing those functions.

PHP Function Arguments

- We can pass the information in PHP function through arguments which is separated by comma.

- PHP supports Call by Value (default), Call by Reference, Default argument values and Variable-length argument list.

PHP Function Arguments(contd.)

```
<?php
function cse($course, $credit) {
  echo "$course with credit $credit<br>";
}

cse("HTML", 3);
cse("CSS",2);
cse("JavaScript", 3);
cse("PHP", 4);
cse("Node js", 5);
?>
```

OUTPUT:

HTML with credit 3
CSS with credit 2
Javascript with credit 3
PHP with credit 4
Node js with credit 5

Loosely Typed Language

- Notice that we did not have to tell PHP which data type the variable is.

- PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

- In the following example, if we try to send both a number and a string to the function, we get fatal error.

Loosely Typed Language(contd.)

```
<?php
function add(int $a, int $b) {
  echo $a + $b;
}
add(2, "3 semesters");  //error
?>
```

OUTPUT:

Fatal error

PHP Default Argument Value

- The following example shows how to use a default parameter.
- If we call the function setchildren() without arguments it takes the default value as argument:

```
<?php
function setchildren(int $children = 0) {
  echo "Total number of children are : $children <br>";
}

setchildren(2);
setchildren(1);
setchildren();
setchildren(3);
?>
```

OUTPUT:
Total number of children are : 2
Total number of children are : 1
Total number of children are : 0
Total number of children are : 3

Returning values

- A function can return a value back to the script that called the function using the return statement.

```
<?php
function add(int $a, int $b) {
  return $a+$b;
}

echo "5 + 10 = " . add(5,10) . "<br>";
echo "7 + 13 = " . add(7,13) . "<br>";
echo "2 + 4 = " . add(2,4);
?>
```

OUTPUT:
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6

Return Type Declarations

- PHP 7 also supports Type Declarations for the return statement.

Return Type Declarations(contd.)

```
<?php
function add(float $a, float $b) {
  return $a + $b;
}
echo add(1.2, 5.2);
?>
```

OUTPUT:

6.4

Return Type Declarations(contd.)

```
<?php
function add(float $a, float $b) {
  return (int)($a + $b);
}
echo add(1.2, 5.2);
?>
```

OUTPUT:

6

Passing Arguments to a Function by Reference

- In PHP there are two ways you can pass arguments to a function: by value and by reference.
- By default, function arguments are passed by value so that if the value of the argument within the function is changed, it does not get affected outside of the function.
- However, to allow a function to modify its arguments, they must be passed by reference.
- Passing an argument by reference is done by prepending an ampersand (&) to the argument name in the function definition, as shown in the example below:

Passing Arguments to a Function by Reference(contd.)

```
<?php
/* Defining a function that multiply a number
by itself and return the new value */
function selfMultiply(&$number){
    $number *= $number;
    return $number;
}

$mynum = 5;
echo $mynum; // Outputs: 5
echo "<br>";
selfMultiply($mynum);
echo $mynum; // Outputs: 25
?>
```

OUTPUT:
525

Passing Arguments to a Function by Reference(contd.)

```
<?php
function product(&$value) {
  $value *= 5;
}

$num = 2;
product($num);
echo $num;
?>
```

OUTPUT:

10

Passing Arguments to a Function by Reference(contd.)

```
<?php
function adder(&$str2)
{
    $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

OUTPUT:

Hello Call By Reference

Understanding the Variable Scope

- However, you can declare the variables anywhere in a PHP script.

- But, the location of the declaration determines the extent of a variable's visibility within the PHP program i.e. where the variable can be used or accessed. This accessibility is known as variable scope.

- By default, variables declared within a function are local and they cannot be viewed or manipulated from outside of that function, as demonstrated in the example below:

PHP Variable Scope

- The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

- PHP has three types of variable scopes:

- Local variable
- Global variable
- Static variable

Local variable

- The variables that are declared within a function are called local variables for that function.
- These local variables have their scope only in that particular function in which they are declared.
- This means that these variables cannot be accessed outside the function, as they have local scope.
- A variable declaration outside the function with the same name is completely different from the variable declared inside the function.

Local variable(contd.)

```
<?php
    function mytest()
    {
        $lang = "PHP";   //Local variable
        echo "Web development language: " .$lang;
    }
    mytest();
?>
```

OUTPUT:

Web development language: PHP

Local variable(contd.)

```
<?php
    function mytest()
    {
        $lang = "PHP";
        echo "Web development language: " .$lang;
    }
    mytest();
    //using $lang (local variable) outside the function will generate an error
    echo $lang;
?>
```

OUTPUT:

Web development language: PHP

Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 28

Global variable

- The global variables are the variables that are declared outside the function.
- These variables can be accessed anywhere in the program.
- To access the global variable within a function, use the GLOBAL keyword before the variable.
- However, these variables can be directly accessed or used outside the function without any keyword.
- Therefore there is no need to use any keyword to access a global variable outside the function.

Global variable(contd.)

```
<?php
        $lang = "PHP";
    function mytest()
    {
        global $lang;
        echo $lang;
                    echo "<br>";
    }
    mytest();
    echo $lang;
?>
```

OUTPUT:
PHP
PHP

Global variable(contd.)

```
<?php
        $lang = "PHP";
    function mytest()
    {
        $lang = "Java";
        echo $lang;
                    echo "<br>";
    }
    mytest();
    echo $lang;
?>
```

OUTPUT:

Java
PHP

Global variable(contd.)

Note: Without using the global keyword, if you try to access a global variable inside the function, it will generate an error that the variable is undefined.

Global variable(contd.)

```
<?php
        $lang = "PHP";
    function mytest()
    {
        echo $lang;
            echo "<br>";
    }
    mytest();
?>
```

OUTPUT:

Warning: Undefined variable $lang in C:\xampp\htdocs\a.php on line 5

Global variable(contd.)

- **Using $GLOBALS instead of global** - Another way to use the global variable inside the function is predefined $GLOBALS array.

```
<?php
        $lang = "PHP";
   function mytest()
   {
                $l = $GLOBALS['lang'];
      echo $l;
                echo "<br>";
   }
   mytest();
?>
```

OUTPUT:
PHP

Global variable(contd.)

```
<?php
    $num1 = 5;       //global variable
    $num2 = 13;      //global variable
    function global_var()
    {
            $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
            echo "Sum of global variables is: " .$sum;
    }
    global_var();
?>
```

OUTPUT:

Sum of global variables is: 18

Global variable(contd.)

- If two variables, local and global, have the same name, then the local variable has higher priority than the global variable inside the function.

```
<?php
    $x = 5;
    function mytest()
    {
        $x = 7;
        echo "value of x: " .$x;
    }
    mytest();
?>
```

OUTPUT:

Value of x: 7

Static variable

- When a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

- From the example is next slide, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Static variable(contd.)

```
<?php
   function static_var()
   {
      static $num1 = 2;       //static variable
      $num2 = 5;          //Non-static variable
      //increment in non-static variable
      $num1++;
      //increment in static variable
      $num2++;
      echo "Static: " .$num1 ."</br>";
      echo "Non-static: " .$num2 ."</br>";
   }

//first function call
   static_var();

   //second function call
   static_var();
?>
```

OUTPUT:
Static: 3
Non-static: 6
Static: 4
Non-static: 6

Static variable(contd.)

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
  static $x = 1;
  echo $x;
  $x++;
}

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
?>

</body>
</html>
```

OUTPUT:
1
2
3