Unit Three

2. PHP Forms and Statements

2.1. PHP Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts.

- ✓ The syntax could be written as -

```
<form action="url to submit the form filled" method="get" or "post">
           <!-- form contents -->
</form>
```

- ✓ Where **action="…"** is the page that the form should submit its data to, and **method="…"** is the method by which the form data is submitted. If the method is get the data is passed in the url string, if the method is post it is passed as a separate file.

The form variables are available to PHP in the page to which they have been submitted. The variables are available in three superglobal arrays created by PHP called $_POST, $_GET and $_REQUEST.

The GET Method:-

- ✓ Has restriction to send to server/ database parts up to 1024 characters only.
- ✓ Never use GET method for systems which have password or other sensitive information to be sent to the server.
- ✓ GET can't be used to send binary data, like images or word documents, to the server because GET method sends the encoded user information.
- ✓ The data sent by GET method can be accessed using QUERY_STRING environment variable.

Example: - Save it as info.php

```
<?php
  if( $_GET["name"] || $_GET["age"] )
  { echo "Welcome ". $_GET['name']. "<br />";
     echo "You are ". $_GET['age']. " years old.";
     exit();
  }
?>
<html><body>
  <form action="<?php $_PHP_SELF ?>" method="GET">
  Name: <input type="text" name="name" />
```

```
        Age: <input type="text" name="age" />
        <input type="submit" />
        </form>
</body></html>
```

The above code's action attribute value can be represented as the filename itself like:-

```
<form action="info.php" method="Get">
```

The POST Method:

The POST method transfers information via HTTPs headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- ✓ The POST method does not have any restriction on data size to be sent.
- ✓ Relatively secured and can be used in large data requesting and responding
- ✓ The POST method can be used to send ASCII as well as binary data.
- ✓ The data sent by POST method goes through HTTP header so it is secured enough on HTTP protocol. The PHP provides **$_POST** associative array to access all the information sent using POST method.

Example: - Take the above example and change the value of method attribute in the form from GET to POST and the variable from $_GET to $_POST.

The $_REQUEST variable

- ✓ The PHP $_REQUEST variable contains the contents of $_GET, $_POST, and $_COOKIE variables
- ✓ This variable can be used to get the result from form data sent with both the GET and POST methods.

Example1: - Create the following two.

```
<?php
echo "Welcome to the homepage";
?>
```

Save the above file with a filename welcome.php.

```
<html><body>
<form action="" method="post">
Username:<input type="text" name="username"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="Login">
</form>
</body>
</html>
<?php
```

```
$una="abc";
$pwd="abc1234";
$username=$_REQUEST['username'];
$password=$_REQUEST['password'];
if($username==$una&&$password==$pwd)
header("Location:welcome.php");
else
echo "Please enter valid username/password";
?>
```

Save the above file with a filename login.php, run it and see the output.

The PHP **header ()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

The target is specified by the **Location:** header as the argument to the **header ()** function. After calling this function the **exit ()** function can be used to halt parsing of rest of the code.

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

Common Validations

We can categorize validation in to the following groups:

- ✓ Presence Validation
- ✓ String Length Validation
- ✓ Type Validation
- ✓ Inclusion in set Validation
- ✓ Uniqueness Validation
- ✓ Format Validation

Presence Validation: is used to check if there is something in a field or if a variable is not empty.

```
            $value=“”;
     if(!isset($value)||empty($value))
            die("Validation failed");
```

String Length Validation: is used to check if a value is within a certain range.

```
$password=”itec1234”;
 $min=6;
$max=10;
if(strlen($password)<$min&&strlen($password)>$max)
die("Password doesnot fulfill the requirement");
```

Type Validation: is checking whether the given value is number, string or of another type.

```
$value=5;
if(!is_numeric($value))
      die("Validation failed not integer");
```

Inclusion in set Validation: Is used to validate whether the value is in the set

```
$value=$_POST['value'];
 $set=array("M","F");
if(!in_array($value,$set))
die("Not in the list");
```

Uniqueness Validation: Is used to validate whether the value which is going to be submitted to a database is unique or not

```
$username=$_POST['value'];
$usernamefromdb=array();
if(in_array($username,$usernamefromdb))
die("Not unique");
```

Format Validation: Is used to validate whether the value has the right format e.g. email with @ symbol, currency with $ symbol, DateTime with AM or PM

- uses regular expression on the string Syntax: ***preg_match($regexp,$subject)***

```
if(!preg_match("/PHP/","PHP is SSS"))
      die("match not found");
else
      echo "Match Found!";
```

- ***Digits 0-9 only:*** This uses to check whether an input is digit/ number or not. The following is a syntax to check if $age is a number or not. If not number, it display “Please enter numbers only for Age”

```
if (!preg_match("/^[.0-9]*$/",$value))
    die("Please enter numbers only for Age");
or
$age= htmlspecialchars($_POST['age']);
 if (!preg_match("/\D/",$age))
    die("Please enter numbers only for Age");
```

- ***Letters a-z and A-Z only***

```
if (!preg_match("/^[a-zA-Z ]*$/",$value)) {
  die("Only letters and white space allowed");
```

- ***Validate e-mail address:*** Used to check an email is valid, i.e to have valid forms. There is a simple way to check if data entered into input field named "email" is an e-mail address without any unnecessary complications and fancy regular expressions.

```
if (!filter_var($value, FILTER_VALIDATE_EMAIL))
  die("Invalid email format");
Or
if(!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$value))
       die("Invalid email format");
```

- ***URL Address:*** If there is an input field named "website" we can check for a valid URL address like this

```
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$value)) {
  die("Invalid URL");
```

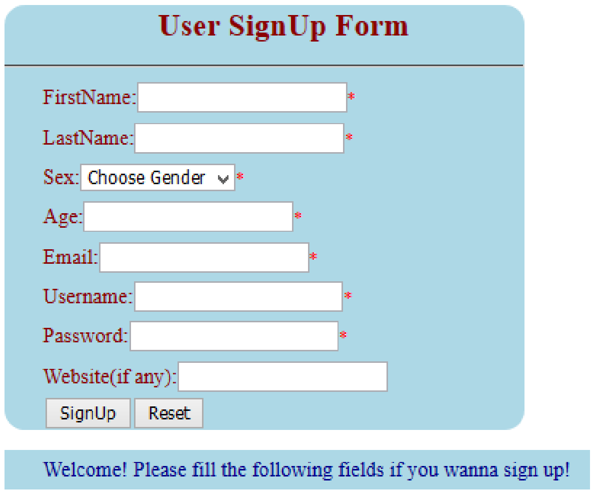
Example: Using form validation. The validation rules here are:

- ✓ Display “Welcome! Please fill the following fields if you wanna sign up!” message while the page is opened for the first time.
- ✓ Display “Please fill all the required fields!” message if all of the fields are empty upon submission.
- ✓ Display “First Name can contain only characters!” message if the user enters characters other than a-z or A-Z in FirstName field.
- ✓ Display “Last Name can contain only characters!” message if the user enters characters other than a-z or A-Z in LastName field.
- ✓ Display “Age can only be number!” message if the user enters characters other than 0-9 in FirstName field.
- ✓ Display “Invalid Email!” message if the user enters an email which does not include @ and . symbols in it.
- ✓ Display “Username already exist!” message if the user enter either of abc,12, xyz usernames
- ✓ Display “Password must be above 6 characters!” message if the user enters a password which consists 6 or less characters.
- ✓ Display “Invalid URL!” message if the user enters a website which does not contain http/s,ftp,www,.domainname.
- ✓ Otherwise display all the values submitted from the form.

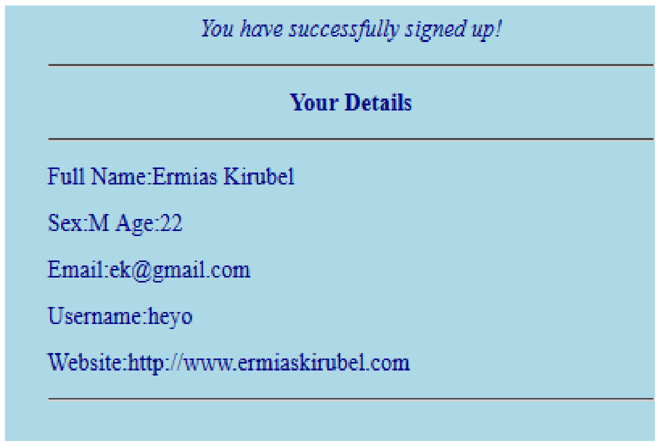
Formvalidation.php

```
<html>
<head><link rel="stylesheet" type="text/css" href="formstyle.css"></head>
<body>
<div id="xx"><div id="xxx"><h1>User SignUp Form</h1></div>
<hr> <div id="xxx">
<form action="" method="post">
FirstName:<input type="text" name="fname"><font>*</font><br>
LastName:<input type="text" name="lname"><font>*</font><br>
Sex:<select name="sex"><option value="" selected>Choose Gender</option><option
value="M">Male</option><option value="F">Female</option></select><font>*</font></br>
Age:<input type="text" name="age"><font>*</font><br>
Email:<input type="text" name="email"><font>*</font><br>
Username:<input type="text" name="username"><font>*</font><br>
Password:<input type="Password" name="password"><font>*</font><br>
Website(if any):<input type="text" name="website"><br>
<input type="submit" name="signup" value="SignUp"><input type="reset" value="Reset">
</form></div></div>
<?php
echo "<div id=xr><div id=xxx>";
if(isset($_POST['signup']))
{
	if(!empty($_POST['fname'])&&!empty($_POST['lname'])&&!empty($_POST['sex'])&&!e
mpty($_POST['age'])&&!empty($_POST['email'])&&!empty($_POST['username'])&&!empty($_PO
ST['password']))
	{
		$fname=$_POST['fname'];
		$lname=$_POST['lname'];
		$sex=$_POST['sex'];
		$age=$_POST['age'];
		$email=$_POST['email'];
		$usernames=array("abc","123","xyz");
		$gender=array("M","F");
		$username=$_POST['username'];
		$password=$_POST['password'];
		$website=$_POST['website'];
		$c=0; $minchar=6;
		if(!preg_match("/^[a-zA-Z ]*$/",$fname)) {
			echo ("First Name can contain only characters<br>");
			$c++;
			}
		if(!preg_match("/^[a-zA-Z ]*$/",$lname)) {
			echo ("Last Name can contain only characters<br>");
			$c++;
			}
		if(!in_array($sex,$gender)){
			echo ("Please select your gender!<br>");
			$c++;
			}
		if(!preg_match("/^[0-9]*$/",$age)) {
			echo ("Age can only be numbers<br>");
			$c++;
			}
		if (!filter_var($email, FILTER_VALIDATE_EMAIL)){
```

```
                        echo ("Invalid email format<br>");
                        $c++;
                        }
                if(in_array($username,$usernames))
                        {
                        echo ("$username already exist!<br>");
                        $c++;
                        }
                if(strlen($password)<$minchar) {
                        echo ("Password must be above 6 characters!<br>");
                        $c++;
                        }
                if($c==0) {
                        echo "<center><i>You have successfully signed up!</i></center>";
                        echo "<hr><b><center>Your Details</center></b><hr>";
                        echo "Full Name:$fname $lname<br>";
                        echo "Sex:$sex    Age:$age<br>";
                        echo "Email:$email<br>Username:$username<br>";
                }
        }else{
                echo "Please fill all the required fields!<br>";
        }
if(!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)&&$c==0)
                                echo "";
                        else if($c>0)
                                echo "Invalid url";
                        else
                                echo "Website:$website<br><hr><br>";
}else{
echo "Welcome! Please fill the following fields if you wanna sign up!</div></div>";
}
?></body></html>
```



Welcome! Please fill the following fields if you wanna sign up!



You have successfully signed up!

Your Details

Full Name:Ermias Kirubel

Sex:M Age:22

Email:ek@gmail.com

Username:heyo

Website:http://www.ermiaskirubel.com

2.2. Decision Statements

The if, else if ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following decision making statements:

The if Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
code to be executed if condition is true;
```

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>
```

Notice that there is no ..else.. in this syntax. The code is executed ***only if the specified condition is true.***

The If...Else Statement

Use this statement, if you want to execute some code if a condition is true and another code if a condition is false.

Syntax

```
if (condition)
code to be executed if condition is true;
else

code to be executed if condition is false;
```

Note*:* If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces.

Example:

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

```
<?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
else
echo "Have a nice day!";
```

```
?>
```

The ElseIf Statement

If you want to execute some code if one of the several conditions is true, then use the elseif statement.

Syntax

```
if (condition)
code to be executed if condition is true;
else if(condition)
code to be executed if condition is true;
else
code to be executed if condition is false;
```

Example:

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!"

```
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..else if..else code.

Syntax:

```
switch (expression)
{
case label1:
code to be executed if expression = label1;
break;
case label2:
code to be executed if expression = label2;
break;
default:
code to be executed if expression is different from both label1 and label2; }
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found, then the code associated with the matching label will be executed. If none of the labels match, then the statement will execute any specified default code.

```
<?php
$d=date("D");
switch ($d)
{
case "Mon":
echo "Today is Monday"; break;
case "Tue":
echo "Today is Tuesday"; break;
case "Wed":
echo "Today is Wednesday"; break;
case "Thu":
echo "Today is Thursday"; break;
case "Fri":
echo "Today is Friday"; break;
case "Sat":
echo "Today is Saturday"; break;
case "Sun":
echo "Today is Sunday"; break;
default:
echo "Wonder which day is this ?";
}
?>
```

2.3. PHP Loops

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

The for Loop Statement

The for loop is used when you know in advance how many times the script should run.

```
Syntax
for (initialization; condition; increment)
  {
  code to be executed;
  }
```

Parameters:

- ✓ *initialization*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- ✓ *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- ✓ *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

Note: Each of the parameters above can be empty, or have multiple expressions (separated by commas).

Example

1. The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>
```

2. The following example displays a product table

```
<html><body>
<?php
echo "<h1>Multiplication table</h1>";
echo "<table border=2 width=50%";
for ($i = 1; $i <= 5; $i++ ) {   //this is the outer loop
  echo "<tr>";
  echo "<td>".$i."</td>";
    for ( $j = 2; $j <= 5; $j++ ) { // inner loop
        echo "<td>".$i * $j."</td>";
    }
   echo "</tr>";
}
echo "</table>";
?>
</body>
</html>
```

The while loop Statement

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

```
while (condition)
{
code to be executed;
}
```

Example: The following example shows how to insert years from 1970-2015 in a form list box and display the selected year upon submit.

```
<html><body>
```

```
<form action="" method="post">
Year:<select name="year">
<?php
$year=1970;
while($year<=2015)
{ echo "<option value=$year>$year</option>";
  $year++;
}
echo "</select>";
?>
<br><input type="submit" value="Display"></form>
<?php
$year=$_POST['year'];
if($year!=null)
echo "Year:$year";
?>
</body></html>
```

The do...while loop Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

Syntax

```
do
  {
  code to be executed;
  }
while (condition);
```

Example

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

```
<html><body>
<?php
$i=1;
echo “The number is:”;
do
  {
  $i++;
  echo "$i ";
  }
while ($i<=5);
```

```
?>
</body></html>
```

This will produce the following result:

```
The number is= 2 3 4 5 6
```

The foreach loop Statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value)
{
code to be executed;
}
```

Example

Try out the following example to list out the values of an array.

```
<?php
$array = array( 1, 2, 3, 4, 5);
echo "Values of the array:”;
foreach( $array as $value )
{
echo “$value ";
}
?>
```

This will produce the following result:

```
Values of the array:1 2 3 4 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example:

```
<?php
$i = 0;
while( $i < 10)
{
$i++;
if( $i == 3 ) break;
}
```

```
echo ("Loop stopped at i = $i" );
?>
```

This will produce the following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<?php
$array = array( 1, 2, 3, 4, 5);
echo “Values of the array:”;
foreach( $array as $value )
{
if( $value == 3 )continue;
echo "$value ";
}
?>
```

This will produce the following result:

```
Values of the array:1 2 4 5
```

2.4. Arrays

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

An array is a special variable that stores one or more similar type of values in a single variable. For example if you want to store 100 numbers then instead of defining 100 variables it’s easy to define an array of 100 length.

An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kinds of arrays:

- ✓ **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- ✓ **Associative array** - An array where each ID key is associated with a value
- ✓ **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, the array index starts from zero.

Example

The following example demonstrates how to create and access numeric arrays. Here we have used **array()** function to create array.

```
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
echo "Value of the array:”;
foreach( $numbers as $value )
{
echo "$value ";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
echo "Value of the array:”;
foreach( $numbers as $value )
{
echo "$value ";
}
?>
```

This will produce the following result:

```
Values of the array: 1 3 2 4 5
Values of the array: one two three four five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in terms of functionality but they are different in terms of their index. When storing data about specific named values, a numerical array is not always the best way to do it. Associative array will have their index as string so that you can establish a strong association between key and values.

Example 1

In this example we use an array to assign credit_hours to the different courses:

```
$credit_hour = array("AIP"=>3, "AP"=>3, "DCN"=>4, "CMT"=>4, "EDP"=>4);
```

This is equivalent with:

```
$credit_hour[‘AIP’] = "3";
$credit_hour[‘AP’] = "3";
$credit_hour[‘DCN’] = "4";
$credit_hour[‘CMT’] = "4";
$credit_hour[‘EDP’] = "4";
<?php
$credit_hour = array("AIP"=>3, "AP"=>3, "DCN"=>4, "CMT"=>4, "EDP"=>4);
$grade = array("AIP"=>'A', "AP"=>'B', "DCN"=>'C', "CMT"=>'A', "EDP"=>'C');
$grade_point= array("AIP"=>12, "AP"=>9, "DCN"=>8, "CMT"=>16, "EDP"=>8);
foreach($credit_hour as $value)
  $total_credit_hour+=$value;
foreach($grade_point as $value)
  $total_grade_point+=$value;
$semester_gpa=number_format(($total_grade_point/$total_credit_hour),2);
echo "GPA:".$semester_gpa;
?>
```

This will produce the following result:

```
GPA: 2.94
```

NOTE: Don't keep associative array inside double quote while printing, otherwise it would not return any value.

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple indexes.

Example:

```
<?php
$grade = array("Abebe"=>array("AIP"=>'A',"AP"=>'B'),"Ermias"=>array("AIP"=>'B',
"AP"=>'C'));
foreach($grade['Abebe'] as $key=>$value)
echo "<br>Abebe scored:".$value." in ".$key;
foreach($grade['Ermias'] as $key=>$value)
echo "<br>Ermias scored:".$value." in ".$key;
?>
```

This will produce the following result:

```
Abebe scored:A in AIP
Abebe scored:B in AP
```

```
Ermias scored:B in AIP
Ermias scored:C in AP
```

The above code is equivalent with:

```
  <?php
$grade['Abebe']['AIP']="A";

$grade['Abebe']['AP']="B";

$grade['Ermias']['AIP']="B";

$grade['Ermias']['AP']="C";

  foreach($grade['Abebe'] as $key=>$value)
  echo "<br>Abebe scored:".$value." in ".$key;
  foreach($grade['Ermias'] as $key=>$value)
  echo "<br>Ermias scored:".$value." in ".$key;"\
  ?>
```

This will produce the same output as the above example.

Adding and Removing Array Elements

PHP comes with four functions that allow us to add or remove elements from the beginning or end of an array: the **array_unshift()** function adds an element to the beginning of an array; the **array_shift()** function removes the first element of an array; the **array_push()** function adds an element to the end of an array; and the **array_pop()** function removes the last element of an array. The following listing illustrates them all in action:

```
<?php
// define array
$movies = array('The Lion King', 'Cars', 'A Bug\'s Life');
// remove element from beginning of array
array_shift($movies);
// remove element from end of array
array_pop($movies);
// add element to end of array
array_push($movies, 'Ratatouille');
// add element to beginning of array
array_unshift($movies, 'The Incredibles');
// print array
// output: ('The Incredibles', 'Cars', 'Ratatouille')
print_r($movies);
?>
```

Notethat:- The array_unshift(), array_shift(), array_push(), and array_pop() functions should be used only with numerically indexed arrays and not with associative arrays. Each of these functions automatically re-indexes the array to account for the value(s).

Sorting Arrays

PHP comes with a number of built-in functions designed for **sorting array elements in different ways**. The first of these is the `sort()` function, which lets to sort numerically indexed arrays alphabetically or numerically, from lowest to highest value.

Some Sorting Functions of PHP which has different sorting orders are:-

- ✓ sort() : sorts the elements in the numeric and alphabetical order.
- ✓ rsort() : sorts the elements in the reverse order.
- ✓ asort() : sorts the elements in the array without changing the indices.
- ✓ ksort() : sorts the arrays by key.

For example:

```
<?php
// define array
$data = array(15,81,14,74,2);
asort($data);
print_r($data);// Array ( [4] => 2 [2] => 14 [0] => 15 [3] => 74 [1] => 81 )
echo "<br/>";
ksort($data);
print_r($data);// Array ( [0] => 15 [1] => 81 [2] => 14 [3] => 74 [4] => 2 )
echo "<br/>";
sort($data);
print_r($data);// Array ( [0] => 2 [1] => 14 [2] => 15 [3] => 74 [4] => 81 )
?>
```

2.5. PHP Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one or more input in the form of parameter and does some processing and returns a value.

- ✓ Functions MUST be defined before they called. Functions can only be executed if they are called. Function headers are of the format:- function functionName($arg_1, $arg_2, …, $arg_n)
- ✓ Unlike variables, function names are not case sensitive (foo(…) == Foo(…) == FoO(…))
- ✓ **Function statements** do the actual work of the function and must be contained within the function braces

i. Creating PHP Function:

We can create a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<?php
function writeMessage()
{
  echo "You are really a nice person, Have a nice time!";
}
writeMessage();
?>
```

ii. PHP Functions with Parameters:

PHP gives us option to pass our parameters inside a function. We can pass as many as parameters we like. These parameters work like variables inside function. Following example takes two integer parameters and add them together and then print them.

```
 <?php
function addFunction($num1, $num2){
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
```

iii. Passing Arguments by Reference:

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition. The following example shows both the cases.

```
    <html><body> <?php
    function addFive(&$num){
       $num += 5;}
    function addSix(&$num){
       $num += 6;}
   $orignum = 10;
addFive($orignum );
   echo "Original Value is $orignum<br />";
addSix($orignum );
   echo "Original Value is $orignum<br />";
?></body></html>
```

iv. PHP Functions returning value:

A function can return a value using the **return** statement in conjunction with a value or object. Return stops the execution of the function and sends the value back to the calling code. It is possible to return more than one value from a function using **return array(1,2,3,4)**.

The following example takes two integer parameters and adds them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html><body>
<?php
function addFunction($num1, $num2){
  $sum = $num1 + $num2;
  return $sum;}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value";
?>
</body></html>
```

We can return multiple values from a function, by placing them all in an array and returning the array. The next example illustrates, by accepting a sentence and returning the individual words, reversed, to the caller as an array:

```
<?php
function add_sub($num1,$num2){
    $add=$num1+$num2;
    $sub=$num1-$num2;
    return array($add,$sub);
}
$result_array=add_sub(15,20);
echo "Add:".$result_array[0]."<br>";
echo "Sub:".$result_array[1]."<br>";
//you can use list function to assign the returned value to variables
list($add_result,$sub_result)=add_sub(30,20);
echo "Add:".$add_result."<br>";
echo "Sub:".$sub_result;
?>
```

v. Setting Default Values for Function Parameters:

We can set a parameter to have a default value if the function's caller doesn't pass it.

```
<?php
function studinfo($department="IT",$year="3rd"){
    return "You are $year year $department student<br>";
}
echo studinfo();
echo studinfo("Software Engineering","1st");
echo studinfo("Electrical",Null);
echo studinfo("2nd");
?>
```

Output
```
You are 3rd year IT student
You are 1st year Software Engineering student
You are year Electrical student
You are 3rd year 2nd student
```

vi. Dynamic Function Calls:

It is possible to assign function names as strings to variables and then treat these variables exactly as the function name itself. Following example depicts this behavior. From the following code, Hello will be displayed.

```
<html><body>
<?php
function sayHello(){
   echo "Hello<br />";}
$function_holder = "sayHello";
$function_holder();
?>
</body></html>
```

vii. Using Recursive Function

Another, slightly more complex idea involving functions is *recursion.* A *recursive function* is a function that calls itself repeatedly until a condition is satisfied. Such a function is typically used to solve complex, iterative calculations, or to process deeply nested structures.

```
<?php
function printValues($arr) {
global $count;
global $out;
if (!is_array($arr)) {
die('ERROR: Input is not an array');
}
foreach ($arr as $a) {
if (is_array($a)) {
printValues($a);
} else {
$out[] = $a;
$count++;
}
}
return array('total' => $count, 'values' => $out);
}
$data = array('o' => array('orange','owl','one'),'t' => array('tea','ten','tag','twentythree' =>
array(array('twenty', 'three'),array('vingt', 'trois', array('red' => 'baron','blue' => 'blood')))));
// count and print values in nested array
$ret = printValues($data);
echo $ret['total'] . ' value(s) found: ';
echo "<br/>";
echo implode(', ', $ret['values']);
?>
```

Output:
```
12 value(s) found:
orange, owl, one, tea, ten, tag, twenty, three, vingt, trois, baron, blood
```