Chapter 1

**\*Loops \***

PHP is used to execute a statement or a block of statements, multiple times until and unless a specific condition is met. This helps the user to save both time and effort of writing the same code multiple times.

PHP supports four types of looping techniques;

1. for loop
2. while loop
3. do-while loop
4. foreach loop

**for loop**: This type of loops is used when the user knows in advance, how many times the block needs to execute. That is, the number of iterations is known beforehand. These type of loops are also known as entry-controlled loops. There are three main parameters to the code, namely the initialization, the test condition and the counter.

Syntax :

for (initialization expression; test condition; update expression) { // code to be executed}

Example -

<?php

// code to illustrate for loop

**for** ($num = 1; $num <= 10; $num += 2) {

echo "$num \n";

}

?>

**while loop**: The while loop is also an entry control loop like for loops i.e., it first checks the condition at the start of the loop and if its true then it enters the loop and executes the block of statements, and goes on executing it as long as the condition holds true.

Syntax :

while (if the condition is true) { // code is executed}

Example -

<?php

// PHP code to illustrate while loops

$num = 2;

**while** ($num < 12) {

$num += 2;

echo $num, "\n";

}

?>

**do-while loop**: This is an exit control loop which means that it first enters the loop, executes the statements, and then checks the condition. Therefore, a statement is executed at least once on using the do…while loop. After executing once, the program is executed as long as the condition holds true.

Syntax :

do { //code is executed} while (if condition is true);

Example -

<?php

// PHP code to illustrate do...while loops

$num = 2;

**do** {

$num += 2;

echo $num, "\n";

} **while** ($num < 12);

?>

**foreach loop**: This loop is used to iterate over arrays. For every counter of loop, an array element is assigned and the next counter is shifted to the next element.

Syntax :

foreach (array\_element as value) { //code to be executed}

|  |
| --- |
| <?php    $arr = **array** (10, 20, 30, 40, 50, 60);  **foreach** ($arr **as** $val) {  echo "$val \n";  }    $arr = **array** ("Ram", "Laxman", "Sita");  **foreach** ($arr **as** $val) {  echo "$val \n";  }    ?> |

Output:

10

\*String Functions\*

## PHP string functions

PHP string functions are used to manipulate string values.

We are now going to look at some of the commonly used string functions in PHP

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Description** | **Example** | **Output** |
| strtolower | Used to convert all string characters to lower case letters | echo strtolower( 'Benjamin'); | outputs benjamin |
| strtoupper | Used to convert all string characters to upper case letters | echo strtoupper('george w bush'); | outputs GEORGE W BUSH |
| strlen | The string length function is used to count the number of character in a string. Spaces in between characters are also counted | echo strlen('united states of america'); | 24 |
| explode | Used to convert strings into an array variable | $settings = explode(';', "host=localhost; db=sales; uid=root; pwd=demo"); print\_r($settings); | Array ( [0] => host=localhost [1] => db=sales [2] => uid=root [3] => pwd=demo ) |
| substr | Used to return part of the string. It accepts three (3) basic parameters. The first one is the string to be shortened, the second parameter is the position of the starting point, and the third parameter is the number of characters to be returned. | $my\_var = 'This is a really long sentence that I wish to cut short';echo substr($my\_var,0, 12).'...'; | This is a re... |
| str\_replace | Used to locate and replace specified string values in a given string. The function accepts three arguments. The first argument is the text to be replaced, the second argument is the replacement text and the third argument is the text that is analyzed. | echo str\_replace ('the', 'that', 'the laptop is very expensive'); | that laptop is very expensive |
| strpos | Used to locate the and return the position of a character(s) within a string. This function accepts two arguments | echo strpos('PHP Programing','Pro'); | 4 |
| sha1 | Used to calculate the SHA-1 hash of a string value | echo sha1('password'); | 5baa61e4c 9b93f3f0 682250b6cf8331b 7ee68fd8 |
| md5 | Used to calculate the md5 hash of a string value | echo md5('password'); | 9f961034ee 4de758 baf4de09ceeb1a75 |
| str\_word\_count | Used to count the number of words in a string. | echo str\_word\_count ('This is a really long sentence that I wish to cut short'); | 12 |
| ucfirst | Make the first character of a string value upper case | echo ucfirst('respect'); | Outputs Respect |
| lcfirst | Make the first character of a string value lower case | echo lcfirst('RESPECT'); | Outputs rESPECT |

\*Control Statements\*

The **if**, **if…else** and **if…elseif…else** construct are one of the most important features of many languages. These conditional statements provides us different actions for the different conditions. When we write code, we perform different actions for different decisions. Like any other languages, PHP is built out of a series of control statements. The control statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing or an empty statement.

## 1. The if Statement

Use the if statement to execute some code only if a specified condition is true.  
The expression is evaluated to its Boolean value. If expression evaluates to TRUE, PHP will execute statement, and if it evaluates to FALSE – it’ll ignore it

**Syntax**  
**if** (condition) {  
code to be executed if condition is true;  
}

The following example would display ” A is bigger than B” if $a is bigger than $b:  
<?php  
if ($a > $b)  
echo "A is bigger than B";  
?>

## 2. The if…else Statement

elseif, as its name suggests, is a combination of if and else. Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

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

For example, the following code would display a is bigger than b, a equal to b or a is smaller than b:  
<?php  
if ($a > $b) {  
echo "a is bigger than b";  
} elseif ($a == $b) {  
echo "a is equal to b";  
} else {  
echo "a is smaller than b";  
}  
?>

## 3. The if…elseif….else Statement

Use the if….elseif…else statement to select one of several blocks of code to be executed.

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

Note: Note that elseif and else if will only be considered exactly the same when using curly brackets as in the above example. When using a colon to define your if/elseif conditions, you must not separate else if into two words, or PHP will fail with a parse error.

## 4. The Switch Statement

The switch statement is similar to IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the switch statement is for.

switch ( )  
{  
case condition1  
break;  
case condition2  
break;  
}

For example, the following code would display $i matched value as 0 or 1 or 2:  
<?php  
switch ($i) {  
case 0:  
echo "i equals 0";  
case 1:  
echo "i equals 1";  
case 2:  
echo "i equals 2";  
}  
?>

\*Arrays in PHP\*

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key.

* **Indexed or Numeric Arrays:** An array with a numeric index where values are stored linearly.
* **Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.
* **Multidimensional Arrays:** An array which contains single or multiple array within it and can be accessed via multiple indices.

Indexed or Numeric Arrays

<?php

// One way to create an indexed array

$name\_one = **array**("Zack", "Anthony", "Ram", "Salim", "Raghav");

// Accessing the elements directly

echo "Accessing the 1st array elements directly:\n";

echo $name\_one[2], "\n";

echo $name\_one[0], "\n";

echo $name\_one[4], "\n";

// Second way to create an indexed array

$name\_two[0] = "ZACK";

$name\_two[1] = "ANTHONY";

$name\_two[2] = "RAM";

$name\_two[3] = "SALIM";

$name\_two[4] = "RAGHAV";

// Accessing the elements directly

echo "Accessing the 2nd array elements directly:\n";

echo $name\_two[2], "\n";

echo $name\_two[0], "\n";

echo $name\_two[4], "\n";

?>

Traversing over a indexed arrays

$name\_one = **array**("Zack", "Anthony", "Ram", "Salim", "Raghav");

// One way of Looping through an array usign foreach

echo "Looping using foreach: \n";

**foreach** ($name\_one **as** $val){

echo $val. "\n";

}

**Associative Arrays**

These type of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

$name\_one = **array**("Zack"=>"Zara", "Anthony"=>"Any",

"Ram"=>"Rani", "Salim"=>"Sara",

"Raghav"=>"Ravina");

echo "Accessing the elements directly:\n";

echo $name\_two["zack"], "\n";

echo $name\_two["salim"], "\n";

echo $name\_two["anthony"], "\n";

echo $name\_one["Ram"], "\n";

echo $name\_one["Raghav"], "\n";

Traversing over asssociative arrays

$name\_one = **array**("Zack"=>"Zara", "Anthony"=>"Any",

"Ram"=>"Rani", "Salim"=>"Sara",

"Raghav"=>"Ravina");

// Looping through an array using foreach

echo "Looping using foreach: \n";

**foreach** ($name\_one **as** $val => $val\_value){

echo "Husband is ".$val." and Wife is ".$val\_value."\n";

}

**Multidimensional Arrays**

Multi-dimensional arrays are such arrays which stores an another array at each index instead of single element. In other words, we can define multi-dimensional arrays as array of arrays.

<?php

// Defining a multidimensional array

$favorites = **array**(

**array**(

"name" => "Dave Punk",

"mob" => "5689741523",

"email" => "[davepunk@gmail.com](mailto:davepunk@gmail.com)",

),

**array**(

"name" => "Monty Smith",

"mob" => "2584369721",

"email" => "[montysmith@gmail.com](mailto:montysmith@gmail.com)",

),

**array**(

"name" => "John Flinch",

"mob" => "9875147536",

"email" => "[johnflinch@gmail.com](mailto:johnflinch@gmail.com)",

)

);

// Accessing elements

echo "Dave Punk email-id is: " . $favorites[0]["email"], "\n";

echo "John Flinch mobile number is: " . $favorites[1]["mob"];

?>

Traversing over multidimensional arrays

$keys = array\_keys($favorites);

**for**($i = 0; $i < count($favorites); $i++) {

echo $keys[$i] . "\n";

**foreach**($favorites[$keys[$i]] **as** $key => $value) {

echo $key . " : " . $value . "\n";

}

echo "\n";

}

----- IMPORTANT -------

[**https://www.w3schools.com/php/php\_arrays\_sort.asp**](https://www.w3schools.com/php/php_arrays_sort.asp)

\*Basics \*

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

Remember that PHP variable names are case-sensitive!

## PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

* local
* global
* static

<?php  
$x = 5; // global scope  
  
function myTest() {  
 // using x inside this function will generate an error  
 echo "<p>Variable x inside function is: $x</p>";  
}   
myTest();  
  
echo "<p>Variable x outside function is: $x</p>";  
?>

## PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
 global $x, $y;  
 $y = $x + $y;  
}  
  
myTest();  
echo $y; // outputs 15  
?>

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
 $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];  
}   
  
myTest();  
echo $y; // outputs 15  
?>

## PHP The static Keyword

Normally, 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.

<?php  
function myTest() {  
 static $x = 0;  
 echo $x;  
 $x++;  
}  
  
myTest();  
myTest();  
myTest();  
?>

Output :

0  
1  
2

**\*Error Handling (Exception Handling)\***

<https://www.tutorialspoint.com/php/php_error_handling.htm>

**\*Date and time functions \***

<https://www.w3schools.com/php/php_date.asp>