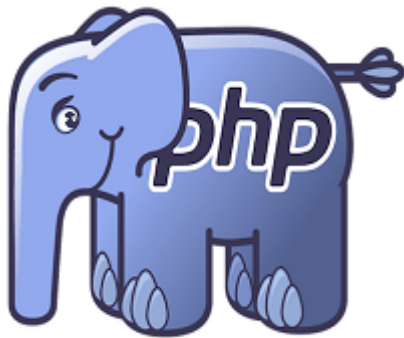


PHP Tutorial



PHP tutorial for beginners and professionals provides in-depth knowledge of PHP scripting language. Our PHP tutorial will help you to learn PHP scripting language easily.

This PHP tutorial covers all the topics of PHP such as introduction, control statements, functions, array, string, file handling, form handling, regular expression, date and time, object-oriented programming in PHP, math, PHP MySQL, PHP with Ajax, PHP with jQuery and PHP with XML.

What is PHP

PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side. PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page.).

PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995. **PHP 7.4.0** is the latest version of PHP, which was released on **28 November**. Some important points need to be noticed about PHP are as followed:

- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.

- PHP is simple and easy to learn language.



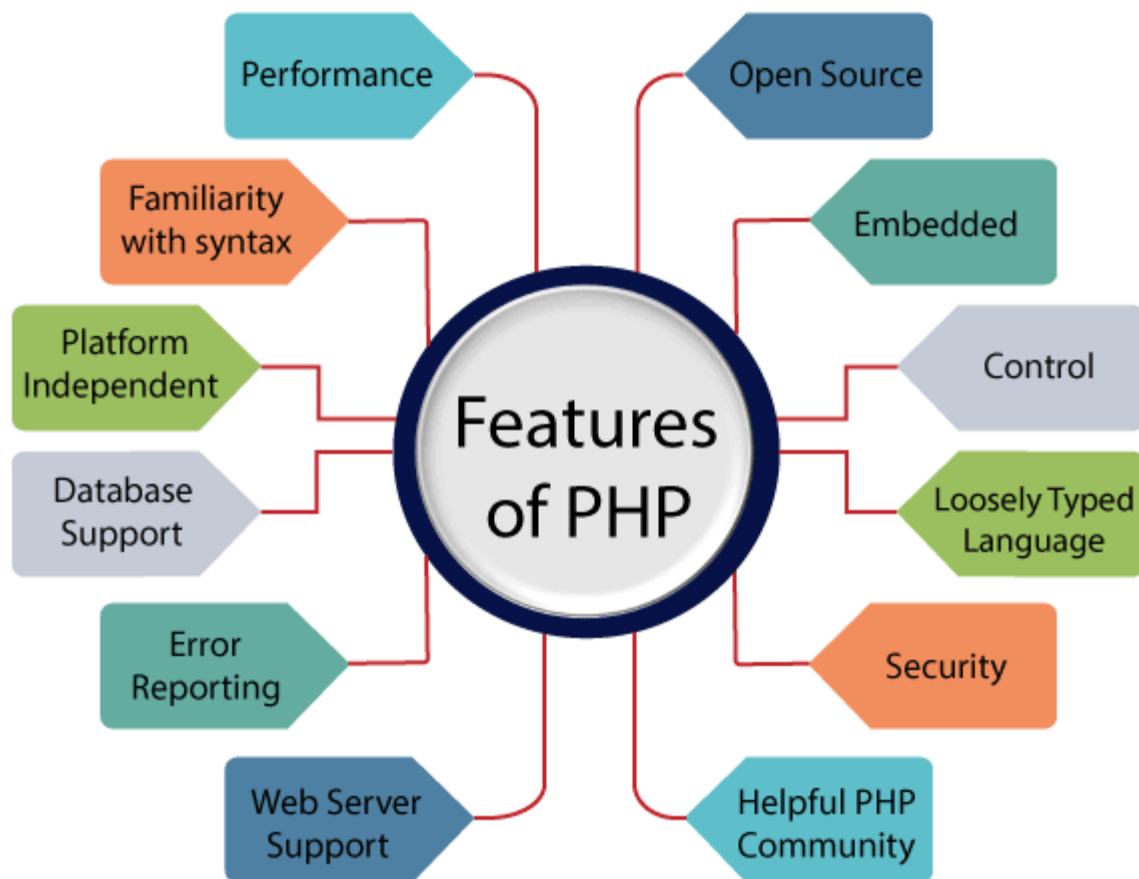
Why use PHP

PHP is a server-side scripting language, which is used to design the dynamic web applications with MySQL database.

- It handles dynamic content, database as well as session tracking for the website.
- You can create sessions in PHP.
- It can access cookies variable and also set cookies.
- It helps to encrypt the data and apply validation.
- PHP supports several protocols such as HTTP, POP3, SNMP, LDAP, IMAP, and many more.
- Using PHP language, you can control the user to access some pages of your website.
- As PHP is easy to install and set up, this is the main reason why PHP is the best language to learn.
- PHP can handle the forms, such as - collect the data from users using forms, save it into the database, and return useful information to the user. **For example** - Registration form.

PHP Features

PHP is very popular language because of its simplicity and open source. There are some important features of PHP given below:



Performance:

PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance.

Open Source:

PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

Familiarity with syntax:

PHP has easily understandable syntax. Programmers are comfortable coding with it.

Embedded:

PHP code can be easily embedded within HTML tags and script.

Platform Independent:

PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

Database Support:

PHP supports all the leading databases such as MySQL, SQLite, ODBC, etc.

Error Reporting -

PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

Loosely Typed Language:

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

Web servers Support:

PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

Security:

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threats and malicious attacks.

Control:

Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

A Helpful PHP Community:

It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

Web Development

PHP is widely used in web development nowadays. PHP can develop dynamic websites easily. But you must have the basic the knowledge of following technologies for web development as well.

- HTML
- CSS
- JavaScript
- Ajax
- XML and JSON
- jQuery

Prerequisite

Before learning PHP, you must have the basic knowledge of **HTML**, **CSS**, and **JavaScript**. So, learn these technologies for better implementation of PHP.

HTML - HTML is used to design static webpage.

CSS - CSS helps to make the webpage content more effective and attractive.

JavaScript - JavaScript is used to design an interactive website.

Audience

Our PHP tutorial is designed to help beginners and professionals. This PHP tutorial will help those who are unaware about the concepts of PHP but have basic knowledge of computer programming.

Problem

We assure you that you will not find any problem in this PHP tutorial. But if there is any mistake or error, please post the error in the contact form.

How to run PHP code in XAMPP

Generally, a PHP file contains HTML tags and some PHP scripting code. It is very easy to create a simple PHP example. To do so, create a file and write HTML tags + PHP code and save this file with .php extension.

Hello First PHP

How to run PHP programs in XAMPP

How to run PHP programs in XAMPP PHP is a popular backend programming language. PHP programs can be written on any editor, such as - Notepad, Notepad++, Dreamweaver, etc. These programs save with **.php** extension, i.e., filename.php inside the htdocs folder.

For example - p1.php.

As I'm using window, and my XAMPP server is installed in D drive. So, the path for the htdocs directory will be "D:\xampp\htdocs".

PHP program runs on a web browser such as - Chrome, Internet Explorer, Firefox, etc. Below some steps are given to run the PHP programs.

Step 1: Create a simple PHP program like hello world.

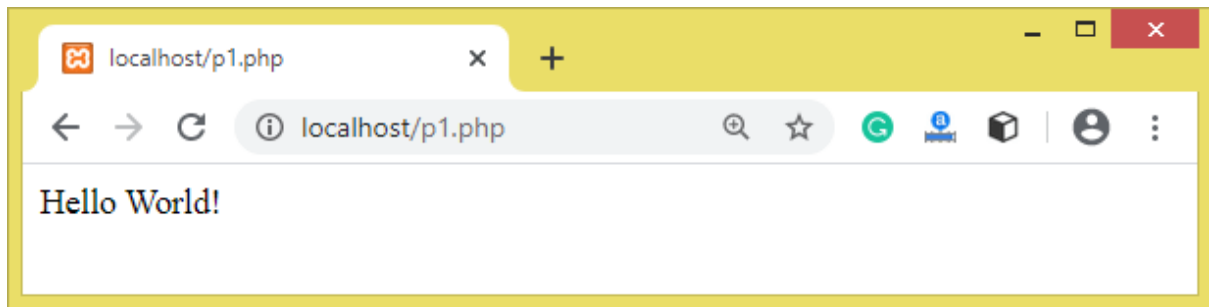
1. `<?php`
2. `echo "Hello World!";`
3. `?>`

Step 2: Save the file with **hello.php** name in the htdocs folder, which resides inside the xampp folder.

Step 3: Run the XAMPP server and start the Apache and MySQL.

Step 4: Now, open the web browser and type localhost `http://localhost/hello.php` on your browser window.

Step 5: The output for the above **hello.php** program will be shown as the screenshot below:



Most of the time, PHP programs run as a web server module. However, PHP can also be run on CLI (Command Line Interface).

PHP Case Sensitivity

In PHP, keyword (e.g., echo, if, else, while), functions, user-defined functions, classes are not case-sensitive. However, all variable names are case-sensitive.

In the below example, you can see that all three echo statements are equal and valid:

1. `<!DOCTYPE>`
2. `<html>`
3. `<body>`
4. `<?php`
5. `echo "Hello world using echo </br>";`
6. `ECHO "Hello world using ECHO </br>";`
7. `EcHo "Hello world using EcHo </br>";`
8. `?>`
9. `</body>`
10. `</html>`

Output:

```
Hello world using echo
Hello world using ECHO
Hello world using EcHo
```

Look at the below example that the variable names are case sensitive. You can see the example below that only the second statement will display the value of the \$color variable. Because it treats \$color, \$CoLoR, and \$COLOR as three different variables:

1. `<html>`
2. `<body>`
3. `<?php`

```
4.      $color = "black";
5.      echo "My car is ". $CoLoR."</br>";
6.      echo "My dog is ". $color."</br>";
7.      echo "My Phone is ". $COLOUR."</br>";
8.      ?>
9.      </body>
10.     </html>
```

Output:

```
Notice: Undefined variable: CoLoR in D:\xampp\htdocs\program\p2.php on line 8
My car is
My dog is black
```

```
Notice: Undefined variable: COLOUR in D:\xampp\htdocs\program\p2.php on line 10
My Phone is
```

Only \$color variable has printed its value, and other variables \$CoLoR and \$COLOUR are declared as undefined variables. An error has occurred in line 5 and line 7.

PHP Echo

PHP echo is a language construct, not a function. Therefore, you don't need to use parenthesis with it. But if you want to use more than one parameter, it is required to use parenthesis.

The syntax of PHP echo is given below:

```
1. void echo ( string $arg1 [, string $... ] )
```

PHP echo statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:

- echo is a statement, which is used to display the output.
- echo can be used with or without parentheses: echo(), and echo.
- echo does not return any value.
- We can pass multiple strings separated by a comma (,) in echo.
- echo is faster than the print statement.

PHP echo: printing string

File: *echo1.php*

Play Video

1. `<?php`
2. `echo "Hello by PHP echo";`
3. `?>`

Output:

```
Hello by PHP echo
```

PHP echo: printing multi line string

File: *echo2.php*

1. `<?php`
2. `echo "Hello by PHP echo`
3. `this is multi line`
4. `text printed by`
5. `PHP echo statement`
6. `";`
7. `?>`

Output:

```
Hello by PHP echo this is multi line text printed by PHP echo statement
```

PHP echo: printing escaping characters

File: *echo3.php*

1. `<?php`
2. `echo "Hello escape \"sequence\" characters";`
3. `?>`

Output:

```
Hello escape "sequence" characters
```

PHP echo: printing variable value

File: *echo4.php*

1. `<?php`
2. `$msg="Hello JavaTpoint PHP";`
3. `echo "Message is: $msg";`
4. `?>`

Output:

```
Message is: Hello JavaTpoint PHP
```

PHP Print

Like PHP echo, PHP print is a language construct, so you don't need to use parenthesis with the argument list. Print statement can be used with or without parentheses: print and print(). Unlike echo, it always returns 1.

The syntax of PHP print is given below:

1. `int print(string $arg)`

PHP print statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:

- print is a statement, used as an alternative to echo at many times to display the output.
- print can be used with or without parentheses.
- print always returns an integer value, which is 1.
- Using print, we cannot pass multiple arguments.
- print is slower than the echo statement.

PHP print: printing string

File: print1.php

1. `<?php`
2. `print "Hello by PHP print ";`
3. `print ("Hello by PHP print()");`
4. `?>`

Output:

```
Hello by PHP print Hello by PHP print()
```

PHP print: printing multi line string

File: *print2.php*

1. `<?php`
2. `print "Hello by PHP print`
3. `this is multi line`
4. `text printed by`
5. `PHP print statement`
6. `";`
7. `?>`

Output:

```
Hello by PHP print this is multi line text printed by PHP print statement
```

PHP print: printing escaping characters

File: *print3.php*

1. `<?php`
2. `print "Hello escape \"sequence\" characters by PHP print";`
3. `?>`

Output:

```
Hello escape "sequence" characters by PHP print
```

PHP print: printing variable value

File: *print4.php*

1. `<?php`
2. `$msg="Hello print() in PHP";`
3. `print "Message is: $msg";`
4. `?>`

Output:

```
Message is: Hello print() in PHP
```

PHP Variables

```
<="" p="" style="color: rgb(51, 51, 51); font-family: inter-regular, system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Helvetica, Arial, sans-serif; font-size: 16px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; text-align: justify; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-thickness: initial; text-decoration-style: initial; text-decoration-color: initial;">
```

In PHP, a variable is declared using a **\$ sign** followed by the variable name. Here, some important points to know about variables:

- As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
- After declaring a variable, it can be reused throughout the code.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

1. `$variablename=value;`

Rules for declaring PHP variable:

- A variable must start with a dollar (\$) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so \$name and \$NAME both are treated as different variable.

PHP Variable: Declaring string, integer, and float

Let's see the example to store string, integer, and float values in PHP variables.

File: variable1.php

1. `<?php`
2. `$str="hello string";`
3. `$x=200;`
4. `$y=44.6;`

5. `echo "string is: $str
";`
6. `echo "integer is: $x
";`
7. `echo "float is: $y
";`
8. `?>`

Output:

```
string is: hello string
integer is: 200
float is: 44.6
```

PHP Variable: Sum of two variables

File: variable2.php

1. `<?php`
2. `$x=5;`
3. `$y=6;`
4. `$z=$x+$y;`
5. `echo $z;`
6. `?>`

Output:

```
11
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COlOr etc.

File: variable3.php

1. `<?php`
2. `$color="red";`
3. `echo "My car is " . $color . "
";`
4. `echo "My house is " . $COLOR . "
";`
5. `echo "My boat is " . $coLOR . "
";`
6. `?>`

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
```

```
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

1. `<?php`
2. `$a="hello";`//letter (valid)
3. `$_b="hello";`//underscore (valid)
- 4.
5. `echo "$a
 $_b";`
6. `?>`

Output:

```
hello
hello
```

File: variableinvalid.php

1. `<?php`
2. `$4c="hello";`//number (invalid)
3. `*$d="hello";`//special symbol (invalid)
- 4.
5. `echo "$4c
 $*d";`
6. `?>`

Output:

```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable
(T_VARIABLE)
or '$' in C:\wamp\www\variableinvalid.php on line 2
```

PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

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:

1. Local variable
2. Global variable
3. 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. Let's understand the local variables with the help of an example:

File: local_variable1.php

```
1. <?php
2.     function local_var()
3.     {
4.         $num = 45; //local variable
5.         echo "Local variable declared inside the function is: ". $num;
6.     }
7.     local_var();
8. ?>
```

Output:

```
Local variable declared inside the function is: 45
```

File: local_variable2.php

```
1. <?php
2.     function mytest()
3.     {
4.         $lang = "PHP";
5.         echo "Web development language: " . $lang;
```

```
6.    }
7.    mytest();
8.    //using $lang (local variable) outside the function will generate an error
9.    echo $lang;
10. ?>
```

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.

Let's understand the global variables with the help of an example:

Example:

File: global_variable1.php

```
1. <?php
2.     $name = "Sanaya Sharma";    //Global Variable
3.     function global_var()
4.     {
5.         global $name;
6.         echo "Variable inside the function: ". $name;
7.         echo "</br>";
8.     }
9.     global_var();
10.    echo "Variable outside the function: ". $name;
11. ?>
```

Output:

```
Variable inside the function: Sanaya Sharma
Variable outside the function: Sanaya Sharma
```


Example:

File: *global_variable2.php*

```
1. <?php
2.     $name = "Sanaya Sharma";    //global variable
3.     function global_var()
4.     {
5.         echo "Variable inside the function: ". $name;
6.         echo "</br>";
7.     }
8.     global_var();
9. ?>
```

Output:

```
Notice: Undefined variable: name in D:\xampp\htdocs\program\p3.php on line
6
Variable inside the function:
```

Using \$GLOBALS instead of global

Another way to use the global variable inside the function is predefined \$GLOBALS array.

Example:

File: *global_variable3.php*

```
1. <?php
2.     $num1 = 5;    //global variable
3.     $num2 = 13;  //global variable
4.     function global_var()
5.     {
6.         $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
7.         echo "Sum of global variables is: " . $sum;
8.     }
9.     global_var();
10. ?>
```

Output:

```
Sum of global variables is: 18
```

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

Example:

File: *global_variable2.php*

```
1. <?php
2.     $x = 5;
3.     function mytest()
4.     {
5.         $x = 7;
6.         echo "value of x: " . $x;
7.     }
8.     mytest();
9. ?>
```

Output:

```
Value of x: 7
```

Static variable

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.

Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

Example:

File: *static_variable.php*

```
1. <?php
2.     function static_var()
3.     {
4.         static $num1 = 3;    //static variable
5.         $num2 = 6;        //Non-static variable
6.         //increment in non-static variable
7.         $num1++;
```

```
8.      //increment in static variable
9.      $num2++;
10.     echo "Static: " . $num1 . "</br>";
11.     echo "Non-static: " . $num2 . "</br>";
12. }
13.
14. //first function call
15. static_var();
16.
17. //second function call
18. static_var();
19. ?>
```

Output:

```
Static: 4
Non-static: 7
Static: 5
Non-static: 7
```

You have to notice that \$num1 regularly increments after each function call, whereas \$num2 does not. This is why because \$num1 is not a static variable, so it freed its memory after the execution of each function call.

PHP Constants

PHP constants are name or identifier that can't be changed during the execution of the script except for [magic constants](#), which are not really constants. PHP constants can be defined by 2 ways:

1. Using define() function
2. Using const keyword

Constants are similar to the variable except once they defined, they can never be undefined or changed. They remain constant across the entire program. PHP constants follow the same PHP variable rules. **For example**, it can be started with a letter or underscore only.

Conventionally, PHP constants should be defined in uppercase letters.

PHP constant: define()

Use the define() function to create a constant. It defines constant at run time. Let's see the syntax of define() function in PHP.

1. define(name, value, **case**-insensitive)
1. **name:** It specifies the constant name.
2. **value:** It specifies the constant value.
3. **case-insensitive:** Specifies whether a constant is case-insensitive. Default value is false. It means it is case sensitive by default.

Let's see the example to define PHP constant using define().

File: constant1.php

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP");
3. echo MESSAGE;
4. **?>**

Output:

```
Hello JavaTpoint PHP
```

Create a constant with **case-insensitive** name:

File: constant2.php

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP",true);//not case sensitive
3. echo MESSAGE, "**</br>**";
4. echo message;
5. **?>**

Output:

```
Hello JavaTpoint PHP
Hello JavaTpoint PHP
```

File: constant3.php

1. **<?php**
2. define("MESSAGE","Hello JavaTpoint PHP",false);//case sensitive
3. echo MESSAGE;

4. echo message;
5. `?>`

Output:

```
Hello JavaTpoint PHP
Notice: Use of undefined constant message - assumed 'message'
in C:\wamp\www\vconstant3.php on line 4
message
```

PHP constant: const keyword

PHP introduced a keyword **const** to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function. The constant defined using const keyword are **case-sensitive**.

File: constant4.php

1. `<?php`
2. const MESSAGE="Hello const by JavaTpoint PHP";
3. echo MESSAGE;
4. `?>`

Output:

```
Hello const by JavaTpoint PHP
```

Constant() function

There is another way to print the value of constants using constant() function instead of using the echo statement.

Syntax

The syntax for the following constant function:

1. constant (name)

File: constant5.php

1. `<?php`
2. define("MSG", "JavaTpoint");
3. echo MSG, "`</br>`";
4. echo constant("MSG");

5. //both are similar
6. ?>

Output:

```
JavaTpoint  
JavaTpoint
```

Constant vs Variables

Constant	Variables
Once the constant is defined, it can never be redefined.	A variable can be undefined as well as redefined easily.
A constant can only be defined using define() function. It cannot be defined by any simple assignment.	A variable can be defined by simple assignment (=) operator.
There is no need to use the dollar (\$) sign before constant during the assignment.	To declare a variable, always use the dollar (\$) sign before the variable.
Constants do not follow any variable scoping rules, and they can be defined and accessed anywhere.	Variables can be declared anywhere in the program, but they follow variable scoping rules.
Constants are the variables whose values can't be changed throughout the program.	The value of the variable can be changed.
By default, constants are global.	Variables can be local, global, or static.

PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types (predefined)

2. Compound Types (user-defined)
3. Special Types

PHP Data Types: Scalar Types

It holds only single value. There are 4 scalar data types in PHP.

1. [boolean](#)
2. [integer](#)
3. [float](#)
4. [string](#)

PHP Data Types: Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

1. [array](#)
2. [object](#)

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. [resource](#)
2. [NULL](#)

PHP Boolean

Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**. It is often used with conditional statements. If the condition is correct, it returns TRUE otherwise FALSE.

Example:

1. `<?php`
2. `if (TRUE)`
3. `echo "This condition is TRUE.";`
4. `if (FALSE)`
5. `echo "This condition is FALSE.";`

6. ?>

Output:

```
This condition is TRUE.
```

PHP Integer

Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional part or decimal points.

Rules for integer:

- An integer can be either positive or negative.
- An integer must not contain decimal point.
- Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
- The range of an integer must be lie between 2,147,483,648 and 2,147,483,647 i.e., -2^{31} to 2^{31} .

Example:

```
1. <?php
2.   $dec1 = 34;
3.   $oct1 = 0243;
4.   $hexa1 = 0x45;
5.   echo "Decimal number: " . $dec1. "<br>";
6.   echo "Octal number: " . $oct1. "<br>";
7.   echo "HexaDecimal number: " . $hexa1. "<br>";
8. ?>
```

Output:

```
Decimal number: 34
Octal number: 163
HexaDecimal number: 69
```

PHP Float

A floating-point number is a number with a decimal point. Unlike integer, it can hold numbers with a fractional or decimal point, including a negative or positive sign.

Example:


```
1. <?php
2.     $n1 = 19.34;
3.     $n2 = 54.472;
4.     $sum = $n1 + $n2;
5.     echo "Addition of floating numbers: " . $sum;
6. ?>
```

Output:

```
Addition of floating numbers: 73.812
```

PHP String

A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.

String values must be enclosed either within **single quotes** or in **double quotes**. But both are treated differently. To clarify this, see the example below:

Example:

```
1. <?php
2.     $company = "Javatpoint";
3.     //both single and double quote statements will treat different
4.     echo "Hello $company";
5.     echo "</br>";
6.     echo 'Hello $company';
7. ?>
```

Output:

```
Hello Javatpoint
Hello $company
```

PHP Array

An array is a compound data type. It can store multiple values of same data type in a single variable.

Example:

```
1. <?php
2.     $bikes = array ("Royal Enfield", "Yamaha", "KTM");
```

3. `var_dump($bikes);` //the `var_dump()` function returns the datatype and values
4. `echo "</br>";`
5. `echo "Array Element1: $bikes[0] </br>";`
6. `echo "Array Element2: $bikes[1] </br>";`
7. `echo "Array Element3: $bikes[2] </br>";`
8. `?>`

Output:

```
array(3) { [0]=> string(13) "Royal Enfield" [1]=> string(6) "Yamaha" [2]=> string(3) "KTM" }
Array Element1: Royal Enfield
Array Element2: Yamaha
Array Element3: KTM
```

You will learn more about array in later chapters of this tutorial.

PHP object

Objects are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.

Example:

1. `<?php`
2. `class bike {`
3. `function model() {`
4. `$model_name = "Royal Enfield";`
5. `echo "Bike Model: " . $model_name;`
6. `}`
7. `}`
8. `$obj = new bike();`
9. `$obj -> model();`
10. `?>`

Output:

```
Bike Model: Royal Enfield
```

This is an advanced topic of PHP, which we will discuss later in detail.

PHP Resource

Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources. **For example** - a database call. It is an external resource.

This is an advanced topic of PHP, so we will discuss it later in detail with examples.

PHP Null

Null is a special data type that has only one value: **NULL**. There is a convention of writing it in capital letters as it is case sensitive.

The special type of data type NULL defined a variable with no value.

Example:

1. <?php
2. \$nl = NULL;
3. echo \$nl; *//it will not give any output*
4. ?>

PHP Operators

PHP Operator is a symbol i.e used to perform operations on operands. In simple words, operators are used to perform operations on variables or values. For example:

1. \$num=10+20;*//+ is the operator and 10,20 are operands*

In the above example, + is the binary + operator, 10 and 20 are operands and \$num is variable.

PHP Operators can be categorized in following forms:

- [Arithmetic Operators](#)
- [Assignment Operators](#)
- [Bitwise Operators](#)
- [Comparison Operators](#)
- [Incrementing/Decrementing Operators](#)
- [Logical Operators](#)
- [String Operators](#)
- [Array Operators](#)

- [Type Operators](#)
- [Execution Operators](#)
- [Error Control Operators](#)

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- **Unary Operators:** works on single operands such as ++, -- etc.
- **Binary Operators:** works on two operands such as binary +, -, *, / etc.
- **Ternary Operators:** works on three operands such as "?:".

Arithmetic Operators

The PHP arithmetic operators are used to perform common arithmetic operations such as addition, subtraction, etc. with numeric values.

Operator	Name	Example	Explanation
+	Addition	\$a + \$b	Sum of operands
-	Subtraction	\$a - \$b	Difference of operands
*	Multiplication	\$a * \$b	Product of operands
/	Division	\$a / \$b	Quotient of operands
%	Modulus	\$a % \$b	Remainder of operands
**	Exponentiation	\$a ** \$b	\$a raised to the power \$b

The exponentiation (**) operator has been introduced in PHP 5.6.

Assignment Operators

The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

Operator	Name	Example	Explanation
=	Assign	<code>\$a = \$b</code>	The value of right operand is assigned to the left operand.
<code>+=</code>	Add then Assign	<code>\$a += \$b</code>	Addition same as <code>\$a = \$a + \$b</code>
<code>-=</code>	Subtract then Assign	<code>\$a -= \$b</code>	Subtraction same as <code>\$a = \$a - \$b</code>
<code>*=</code>	Multiply then Assign	<code>\$a *= \$b</code>	Multiplication same as <code>\$a = \$a * \$b</code>
<code>/=</code>	Divide then Assign (quotient)	<code>\$a /= \$b</code>	Find quotient same as <code>\$a = \$a / \$b</code>
<code>%=</code>	Divide then Assign (remainder)	<code>\$a %= \$b</code>	Find remainder same as <code>\$a = \$a % \$b</code>

Bitwise Operators

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
<code>&</code>	And	<code>\$a & \$b</code>	Bits that are 1 in both <code>\$a</code> and <code>\$b</code> are set to 1, otherwise 0.
<code> </code>	Or (Inclusive or)	<code>\$a \$b</code>	Bits that are 1 in either <code>\$a</code> or <code>\$b</code> are set to 1
<code>^</code>	Xor (Exclusive or)	<code>\$a ^ \$b</code>	Bits that are 1 in either <code>\$a</code> or <code>\$b</code> are set to 0.
<code>~</code>	Not	<code>~\$a</code>	Bits that are 1 set to 0 and bits that are 0 are set to 1

<<	Shift left	<code>\$a << \$b</code>	Left shift the bits of operand \$a \$b steps
>>	Shift right	<code>\$a >> \$b</code>	Right shift the bits of \$a operand by \$b number of places

Comparison Operators

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

Operator	Name	Example	Explanation
<code>==</code>	Equal	<code>\$a == \$b</code>	Return TRUE if \$a is equal to \$b
<code>===</code>	Identical	<code>\$a === \$b</code>	Return TRUE if \$a is equal to \$b, and they are of same data type
<code>!==</code>	Not identical	<code>\$a !== \$b</code>	Return TRUE if \$a is not equal to \$b, and they are not of same data type
<code>!=</code>	Not equal	<code>\$a != \$b</code>	Return TRUE if \$a is not equal to \$b
<code><></code>	Not equal	<code>\$a <> \$b</code>	Return TRUE if \$a is not equal to \$b
<code><</code>	Less than	<code>\$a < \$b</code>	Return TRUE if \$a is less than \$b
<code>></code>	Greater than	<code>\$a > \$b</code>	Return TRUE if \$a is greater than \$b
<code><=</code>	Less than or equal to	<code>\$a <= \$b</code>	Return TRUE if \$a is less than or equal \$b
<code>>=</code>	Greater than or equal to	<code>\$a >= \$b</code>	Return TRUE if \$a is greater than or equal \$b

<=>	Spaceship	\$a <=> \$b	Return -1 if \$a is less than \$b Return 0 if \$a is equal \$b Return 1 if \$a is greater than \$b
-----	-----------	-------------	----------------------------------------------------------------------------------------------------------

Incrementing/Decrementing Operators

The increment and decrement operators are used to increase and decrease the value of a variable.

Operator	Name	Example	Explanation
++	Increment	++\$a	Increment the value of \$a by one, then return \$a
		\$a++	Return \$a, then increment the value of \$a by one
--	decrement	--\$a	Decrement the value of \$a by one, then return \$a
		\$a--	Return \$a, then decrement the value of \$a by one

Logical Operators

The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
and	And	\$a and \$b	Return TRUE if both \$a and \$b are true
Or	Or	\$a or \$b	Return TRUE if either \$a or \$b is true
xor	Xor	\$a xor \$b	Return TRUE if either \$ or \$b is true but not both
!	Not	! \$a	Return TRUE if \$a is not true

&&	And	\$a && \$b	Return TRUE if either \$a and \$b are true
	Or	\$a \$b	Return TRUE if either \$a or \$b is true

String Operators

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

Operator	Name	Example	Explanation
.	Concatenation	\$a . \$b	Concatenate both \$a and \$b
.=	Concatenation and Assignment	\$a .= \$b	First concatenate \$a and \$b, then assign the concatenated string to \$a, e.g. \$a = \$a . \$b

Array Operators

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

Operator	Name	Example	Explanation
+	Union	\$a + \$y	Union of \$a and \$b
==	Equality	\$a == \$b	Return TRUE if \$a and \$b have same key/value pair
!=	Inequality	\$a != \$b	Return TRUE if \$a is not equal to \$b
===	Identity	\$a === \$b	Return TRUE if \$a and \$b have same key/value pair of same type in same order
!==	Non-Identity	\$a !== \$b	Return TRUE if \$a is not identical to \$b

<>	Inequality	\$a <> \$b	Return TRUE if \$a is not equal to \$b
----	------------	------------	----------------------------------------

Type Operators

The type operator **instanceof** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to. It is used in object-oriented programming.

```
1. <?php
2.     //class declaration
3.     class Developer
4.     {}
5.     class Programmer
6.     {}
7.     //creating an object of type Developer
8.     $charu = new Developer();
9.
10.    //testing the type of object
11.    if( $charu instanceof Developer)
12.    {
13.        echo "Charu is a developer.";
14.    }
15.    else
16.    {
17.        echo "Charu is a programmer.";
18.    }
19.    echo "</br>";
20.    var_dump($charu instanceof Developer);        //It will return true.
21.    var_dump($charu instanceof Programmer);        //It will return false.
22. ?>
```

Output:

```
Charu is a developer.
bool(true) bool(false)
```

Execution Operators

PHP has an execution operator **backticks (`)**. PHP executes the content of backticks as a shell command. Execution operator and **shell_exec()** give the same result.

Operator	Name	Example	Explanation
`	backticks	echo `dir`;	Execute the shell command and return the result. Here, it will show the directories available in current folder.

Error Control Operators

PHP has one error control operator, i.e., **at (@) symbol**. Whenever it is used with an expression, any error message will be ignored that might be generated by that expression.

Operator	Name	Example	Explanation
@	at	@file ('non_existent_file')	Intentional file error

PHP Operators Precedence

Let's see the precedence of PHP operators with associativity.

Operators	Additional Information	Associativity
clone new	clone and new	non-associative
[array()	left
**	arithmetic	right
++ -- ~ (int) (float) (string) (array) (object) (bool) @	increment/decrement types and	right

instanceof	types	non-associative
!	logical (negation)	right
* / %	arithmetic	left
+ - .	arithmetic and string concatenation	left
<< >>	bitwise (shift)	left
< <= > >=	comparison	non-associative
== != === !== <>	comparison	non-associative
&	bitwise AND	left
^	bitwise XOR	left
	bitwise OR	left
&&	logical AND	left
	logical OR	left
?:	ternary	left
= += -= *= **= /= .= %= &= = ^= <<= >>= =>	assignment	right
and	logical	left
xor	logical	left
or	logical	left
,	many uses (comma)	left

PHP Comments

PHP comments can be used to describe any line of code so that other developer can understand the code easily. It can also be used to hide any code.

PHP supports single line and multi line comments. These comments are similar to C/C++ and Perl style (Unix shell style) comments.

PHP Single Line Comments

There are two ways to use single line comments in PHP.

- // (C++ style single line comment)
 - # (Unix Shell style single line comment)
1. <?php
 2. // this is C++ style single line comment
 3. # this is Unix Shell style single line comment
 4. echo "Welcome to PHP single line comments";
 5. ?>

Output:

```
Welcome to PHP single line comments
```

PHP Multi Line Comments

In PHP, we can comments multiple lines also. To do so, we need to enclose all lines within /* */. Let's see a simple example ofX PHP multiple line comment.

1. <?php
2. /*
3. Anything placed
4. within comment
5. will not be displayed
6. on the browser;
7. */
8. echo "Welcome to PHP multi line comment";
9. ?>

Output:

```
Welcome to PHP multi line comment
```

PHP If Else

PHP if else statement is used to test condition. There are various ways to use if statement in PHP.

- if
- if-else
- if-else-if
- nested if

PHP If Statement

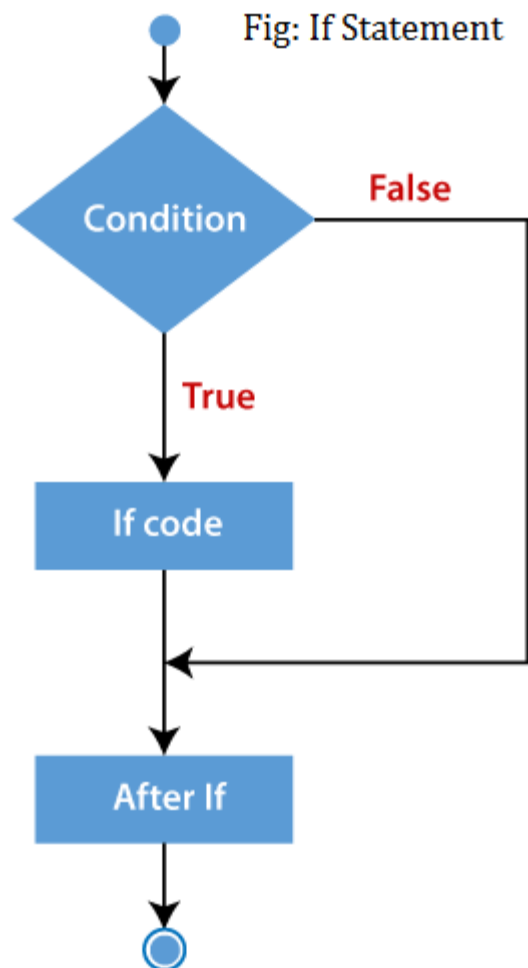
PHP if statement allows conditional execution of code. It is executed if condition is true.

If statement is used to executes the block of code exist inside the if statement only if the specified condition is true.

Syntax`if(condition){`

1. `//code to be executed`
2. `}`

Flowchart



Example

1. <?php
2. \$num=12;
3. **if**(\$num<100){
4. echo "\$num is less than 100";
5. }
6. ?>

Output:

```
12 is less than 100
```

PHP If-else Statement

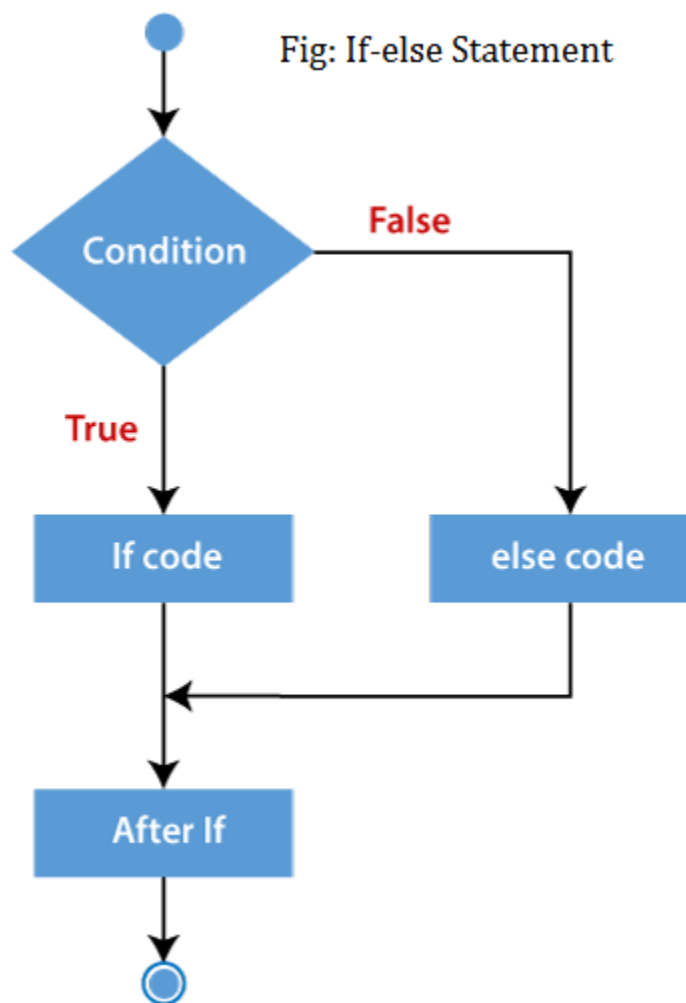
PHP if-else statement is executed whether condition is true or false.

If-else statement is slightly different from if statement. It executes one block of code if the specified condition is **true** and another block of code if the condition is **false**.

Syntax

1. **if**(condition){
2. //code to be executed if true
3. }**else**{
4. //code to be executed if false
5. }

Flowchart



Example

1. <?php
2. \$num=12;
3. **if**(\$num%2==0){
4. echo "\$num is even number";
5. }**else**{
6. echo "\$num is odd number";

7. }
8. ?>

Output:

```
12 is even number
```

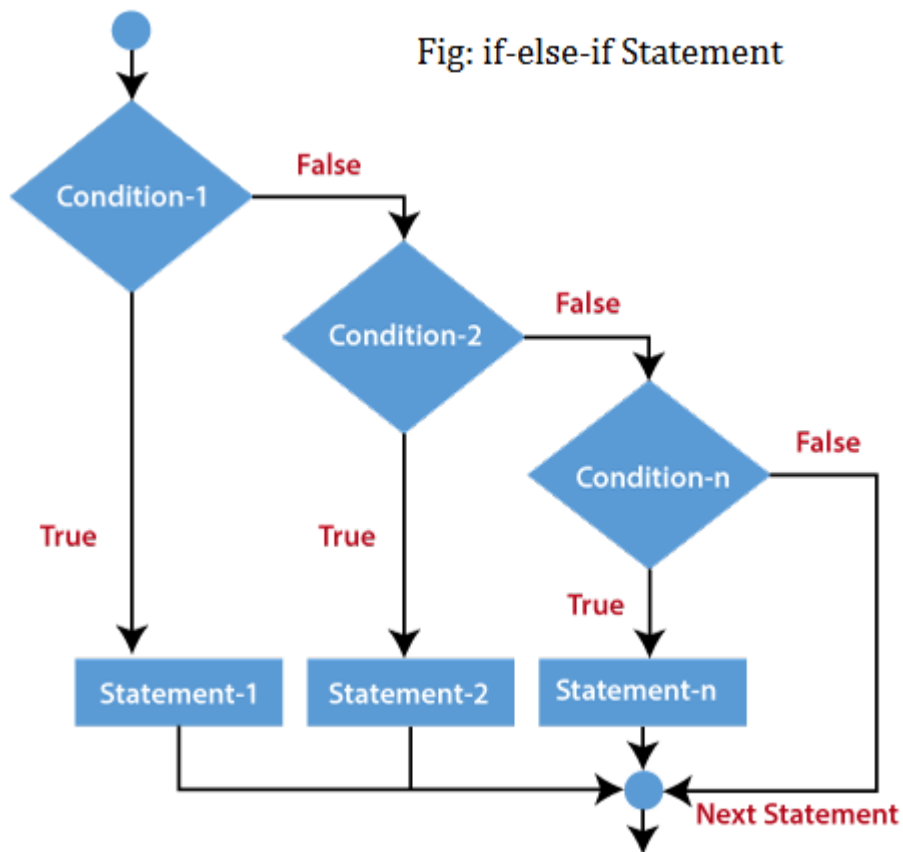
PHP If-else-if Statement

The PHP if-else-if is a special statement used to combine multiple if?.else statements. So, we can check multiple conditions using this statement.

Syntax

1. **if** (condition1){
2. //code to be executed if condition1 is true
3. } **elseif** (condition2){
4. //code to be executed if condition2 is true
5. } **elseif** (condition3){
6. //code to be executed if condition3 is true
7.
8. } **else**{
9. //code to be executed if all given conditions are false
10. }

Flowchart



Example

```

1. <?php
2.   $marks=69;
3.   if ($marks<33){
4.       echo "fail";
5.   }
6.   else if ($marks>=34 && $marks<50) {
7.       echo "D grade";
8.   }
9.   else if ($marks>=50 && $marks<65) {
10.      echo "C grade";
11.  }
12.  else if ($marks>=65 && $marks<80) {
13.      echo "B grade";
14.  }
15.  else if ($marks>=80 && $marks<90) {
16.      echo "A grade";
17.  }
  
```

```
18. else if ($marks>=90 && $marks<100) {  
19.     echo "A+ grade";  
20. }  
21. else {  
22.     echo "Invalid input";  
23. }  
24. ?>
```

Output:

```
B Grade
```

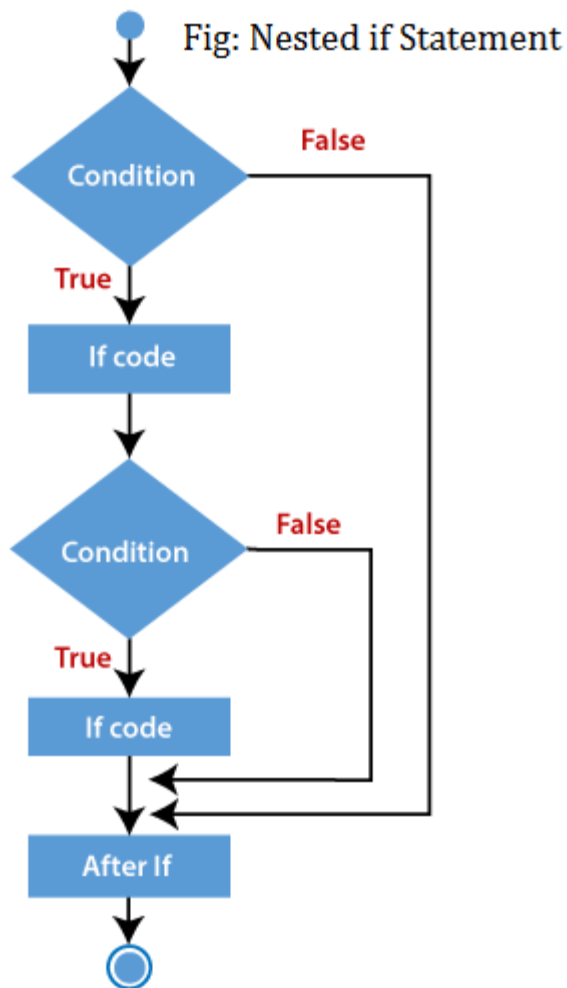
PHP nested if Statement

The nested if statement contains the if block inside another if block. The inner if statement executes only when specified condition in outer if statement is **true**.

Syntax

```
1. if (condition) {  
2.     //code to be executed if condition is true  
3. }  
4. if (condition) {  
5.     //code to be executed if condition is true  
6. }
```

Flowchart



Example

```
1. <?php
2.     $age = 23;
3.     $nationality = "Indian";
4.     //applying conditions on nationality and age
5.     if ($nationality == "Indian")
6.     {
7.         if ($age >= 18) {
8.             echo "Eligible to give vote";
9.         }
10.    else {
11.        echo "Not eligible to give vote";
12.    }
13. }
14. ?>
```

Output:

```
Eligible to give vote
```

PHP Switch Example

```
1. <?php
2.     $a = 34; $b = 56; $c = 45;
3.     if ($a < $b) {
4.         if ($a < $c) {
5.             echo "$a is smaller than $b and $c";
6.         }
7.     }
8. ?>
```

Output:

```
34 is smaller than 56 and 45
```

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

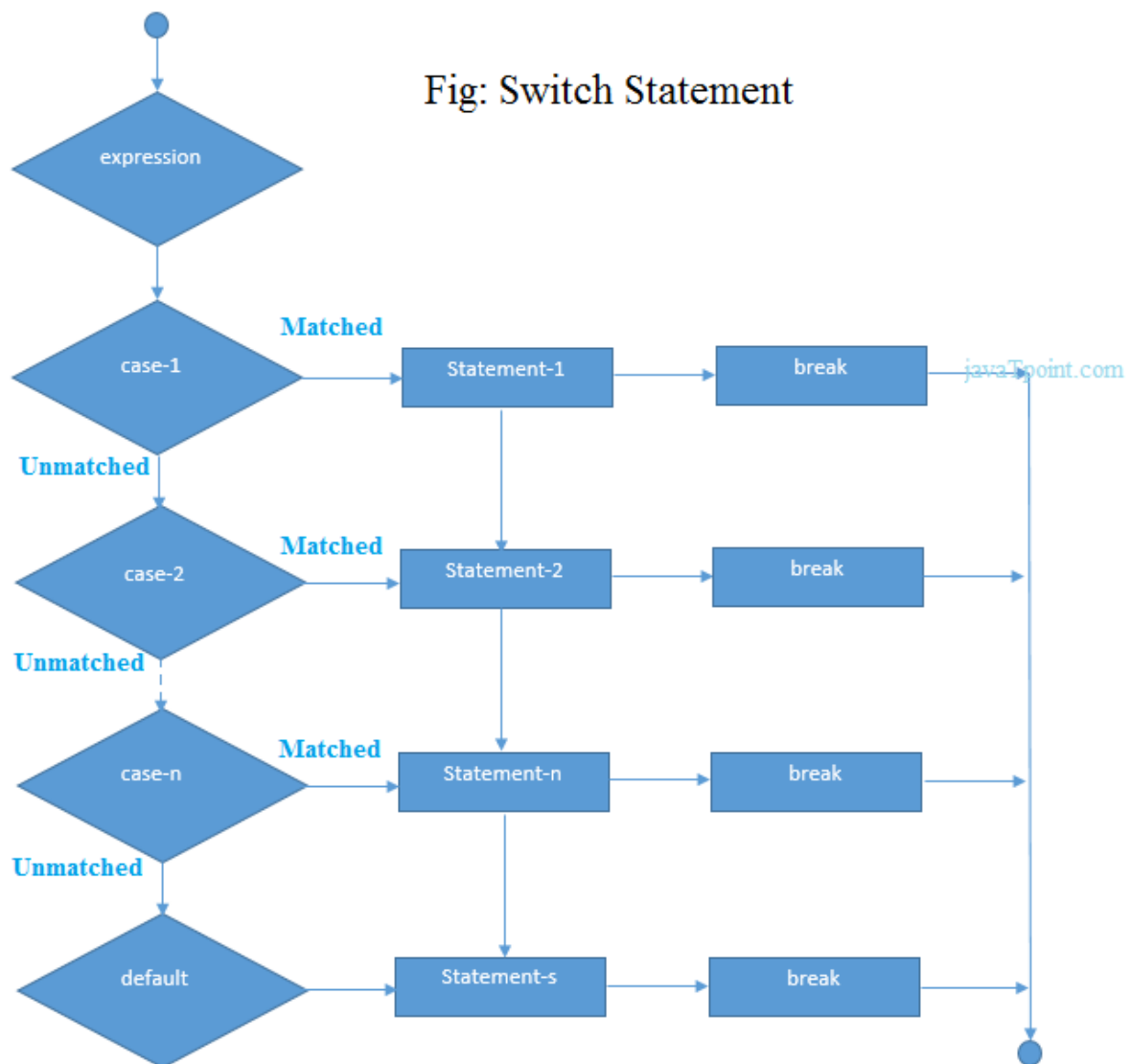
```
1. switch(expression){
2.     case value1:
3.         //code to be executed
4.         break;
5.     case value2:
6.         //code to be executed
7.         break;
8.     .....
9.     default:
10.    code to be executed if all cases are not matched;
11.}
```

Important points to be noticed about switch case:

1. The **default** is an optional statement. Even it is not important, that default must always be the last statement.

2. There can be only one **default** in a switch statement. More than one default may lead to a **Fatal** error.
3. Each case can have a **break** statement, which is used to terminate the sequence of statement.
4. The **break** statement is optional to use in switch. If break is not used, all the statements will execute after finding matched case value.
5. PHP allows you to use number, character, string, as well as functions in switch expression.
6. Nesting of switch statements is allowed, but it makes the program more complex and less readable.
7. You can use semicolon (;) instead of colon (:). It will not generate any error.

PHP Switch Flowchart



PHP Switch Example

```
1. <?php
2. $num=20;
3. switch($num){
4.     case 10:
5.         echo("number is equals to 10");
6.         break;
7.     case 20:
8.         echo("number is equal to 20");
9.         break;
10.    case 30:
11.        echo("number is equal to 30");
12.        break;
13.    default:
14.        echo("number is not equal to 10, 20 or 30");
15.}
16. ?>
```

Output:

```
number is equal to 20
```

PHP switch statement with character

Program to check Vowel and consonant

We will pass a character in switch expression to check whether it is vowel or constant. If the passed character is A, E, I, O, or U, it will be vowel otherwise consonant.

```
1. <?php
2. $ch = 'U';
3. switch ($ch)
4. {
5.     case 'a':
6.         echo "Given character is vowel";
7.         break;
8.     case 'e':
9.         echo "Given character is vowel";
10.        break;
```

```
11.     case 'i':
12.         echo "Given character is vowel";
13.         break;
14.     case 'o':
15.         echo "Given character is vowel";
16.         break;
17.     case 'u':
18.         echo "Given character is vowel";
19.         break;
20.     case 'A':
21.         echo "Given character is vowel";
22.         break;
23.     case 'E':
24.         echo "Given character is vowel";
25.         break;
26.     case 'I':
27.         echo "Given character is vowel";
28.         break;
29.     case 'O':
30.         echo "Given character is vowel";
31.         break;
32.     case 'U':
33.         echo "Given character is vowel";
34.         break;
35.     default:
36.         echo "Given character is consonant";
37.         break;
38. }
39. ?>
```

Output:

```
Given character is vowel
```

PHP switch statement with String

PHP allows to pass string in switch expression. Let's see the below example of course duration by passing string in switch case statement.

```
1. <?php
2.   $ch = "B.Tech";
3.   switch ($ch)
4.   {
5.       case "BCA":
6.           echo "BCA is 3 years course";
7.           break;
8.       case "Bsc":
9.           echo "Bsc is 3 years course";
10.          break;
11.      case "B.Tech":
12.          echo "B.Tech is 4 years course";
13.          break;
14.      case "B.Arch":
15.          echo "B.Arch is 5 years course";
16.          break;
17.      default:
18.          echo "Wrong Choice";
19.          break;
20.  }
21. ?>
```

Output:

```
B.Tech is 4 years course
```

PHP switch statement is fall-through

PHP switch statement is fall-through. It means it will execute all statements after getting the first match, if break statement is not found.

```
1. <?php
2.   $ch = 'c';
3.   switch ($ch)
4.   {
5.       case 'a':
6.           echo "Choice a";
7.           break;
8.       case 'b':
```



```
9.         echo "Choice b";
10.        break;
11.    case 'c':
12.        echo "Choice c";
13.        echo "</br>";
14.    case 'd':
15.        echo "Choice d";
16.        echo "</br>";
17.    default:
18.        echo "case a, b, c, and d is not found";
19.    }
20. ?>
```

Output:

```
Choice c
Choice d
case a, b, c, and d is not found
```

PHP nested switch statement

Nested switch statement means switch statement inside another switch statement. Sometimes it leads to confusion.

```
1. <?php
2.     $car = "Hyundai";
3.     $model = "Tucson";
4.     switch( $car )
5.     {
6.         case "Honda":
7.             switch( $model )
8.             {
9.                 case "Amaze":
10.                    echo "Honda Amaze price is 5.93 - 9.79 Lakh.";
11.                    break;
12.                 case "City":
13.                    echo "Honda City price is 9.91 - 14.31 Lakh.";
14.                    break;
15.            }
```

```
16.         break;
17.     case "Renault":
18.         switch( $model )
19.         {
20.             case "Duster":
21.                 echo "Renault Duster price is 9.15 - 14.83 L.";
22.                 break;
23.             case "Kwid":
24.                 echo "Renault Kwid price is 3.15 - 5.44 L.";
25.                 break;
26.         }
27.         break;
28.     case "Hyundai":
29.         switch( $model )
30.         {
31.             case "Creta":
32.                 echo "Hyundai Creta price is 11.42 - 18.73 L.";
33.                 break;
34.             case "Tucson":
35.                 echo "Hyundai Tucson price is 22.39 - 32.07 L.";
36.                 break;
37.             case "Xcent":
38.                 echo "Hyundai Xcent price is 6.5 - 10.05 L.";
39.                 break;
40.         }
41.         break;
42.     }
43. ?>
```

Output:

```
Hyundai Tucson price is 22.39 - 32.07 L.
```

PHP For Loop

PHP for loop can be used to traverse set of code for the specified number of times.

It should be used if the number of iterations is known otherwise use while loop. This means for loop is used when you already know how many times you want to execute a block of code.

It allows users to put all the loop related statements in one place. See in the syntax given below:

Syntax

1. **for**(initialization; condition; increment/decrement){
2. *//code to be executed*
3. }

Parameters

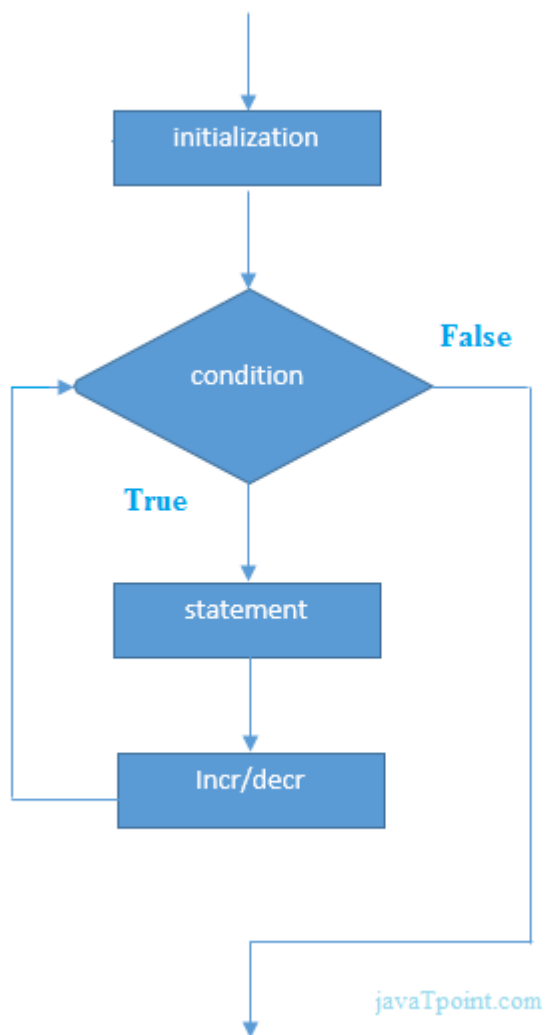
The php for loop is similar to the java/C/C++ for loop. The parameters of for loop have the following meanings:

initialization - Initialize the loop counter value. The initial value of the for loop is done only once. This parameter is optional.

condition - Evaluate each iteration value. The loop continuously executes until the condition is false. If TRUE, the loop execution continues, otherwise the execution of the loop ends.

Increment/decrement - It increments or decrements the value of the variable.

Flowchart



Example

1. `<?php`
2. `for($n=1;$n<=10;$n++){`
3. `echo "$n
";`
4. `}`
5. `?>`

Output:

```
1
2
3
4
5
6
7
8
```

```
9
10
```

Example

All three parameters are optional, but semicolon (;) is must to pass in for loop. If we don't pass parameters, it will execute infinite.

```
1. <?php
2.     $i = 1;
3.     //infinite loop
4.     for (;;) {
5.         echo $i++;
6.         echo "</br>";
7.     }
8. ?>
```

Output:

```
1
2
3
4
.
.
.
```

Example

Below is the example of printing numbers from 1 to 9 in four different ways using for loop.

```
1. <?php
2.     /* example 1 */
3.
4.     for ($i = 1; $i <= 9; $i++) {
5.         echo $i;
6.     }
7.     echo "</br>";
8.
9.     /* example 2 */
10.
11.    for ($i = 1;; $i++) {
12.        if ($i > 9) {
```

```
13.     break;
14.     }
15.     echo $i;
16. }
17. echo "</br>";
18.
19. /* example 3 */
20.
21. $i = 1;
22. for (; ) {
23.     if ($i > 9) {
24.         break;
25.     }
26.     echo $i;
27.     $i++;
28. }
29. echo "</br>";
30.
31. /* example 4 */
32.
33. for ($i = 1, $j = 0; $i <= 9; $j += $i, print $i, $i++);
34. ?>
```

Output:

```
123456789
123456789
123456789
123456789
```

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop. The inner for loop executes only when the outer for loop condition is found **true**.

In case of inner or nested for loop, nested for loop is executed fully for one outer for loop. If outer for loop is to be executed for 3 times and inner for loop for 3 times, inner for loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example

1. <?php
2. **for**(\$i=1;\$i<=3;\$i++){
3. **for**(\$j=1;\$j<=3;\$j++){
4. echo "\$i \$j
";
5. }
6. }
7. ?>

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP For Each Loop

PHP for each loop is used to traverse array elements.

Syntax

1. **foreach**(\$array **as** \$var){
2. //code to be executed
3. }
4. ?>

Example

1. <?php
2. \$season=**array**("summer","winter","spring","autumn");
3. **foreach**(\$season **as** \$arr){
4. echo "Season is: \$arr
";
5. }
6. ?>

Output:

```
Season is: summer
Season is: winter
Season is: spring
```

```
Season is: autumn
```

PHP foreach loop

The foreach loop is used to traverse the array elements. It works only on array and object. It will issue an error if you try to use it with the variables of different datatype.

The foreach loop works on elements basis rather than index. It provides an easiest way to iterate the elements of an array.

In foreach loop, we don't need to increment the value.

Syntax

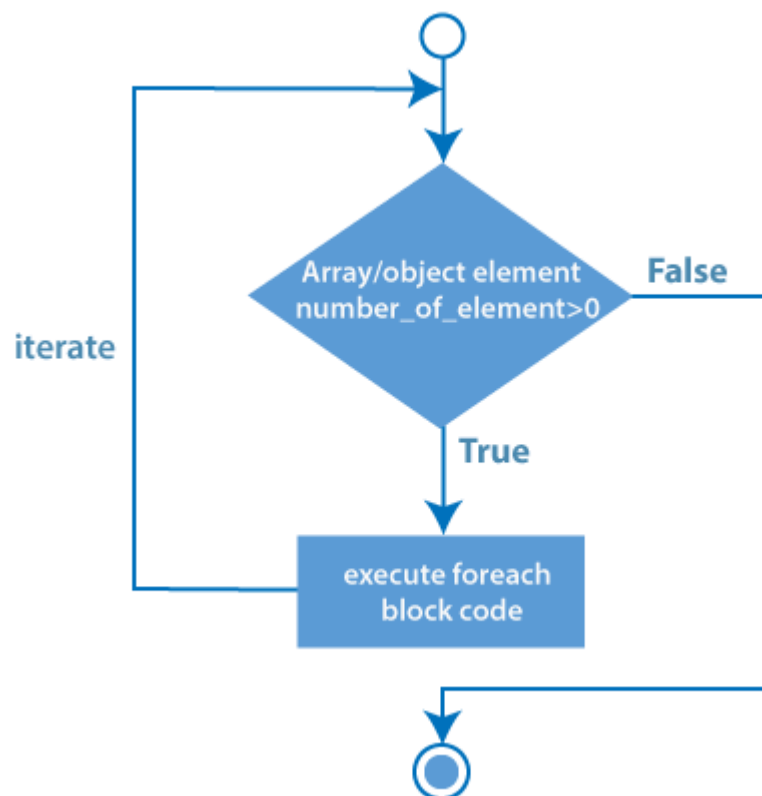
1. **foreach** (\$array **as** \$value) {
2. //code to be executed
3. }

There is one more syntax of foreach loop.

Syntax

1. **foreach** (\$array **as** \$key => \$element) {
2. //code to be executed
3. }

Flowchart



Example 1:

PHP program to print array elements using foreach loop.

```
1. <?php
2. //declare array
3. $season = array ("Summer", "Winter", "Autumn", "Rainy");
4.
5. //access array elements using foreach loop
6. foreach ($season as $element) {
7.     echo "$element";
8.     echo "<br>";
9. }
10. ?>
```

Output:

```
Summer
Winter
Autumn
Rainy
```

Example 2:

PHP program to print associative array elements using foreach loop.

```
1. <?php
2.     //declare array
3.     $employee = array (
4.         "Name" => "Alex",
5.         "Email" => "alex_jtp@gmail.com",
6.         "Age" => 21,
7.         "Gender" => "Male"
8.     );
9.
10.    //display associative array element through foreach loop
11.    foreach ($employee as $key => $element) {
12.        echo $key . " : " . $element;
13.        echo "<br>";
14.    }
15. ?>
```

Output:

```
Name : Alex
Email : alex_jtp@gmail.com
Age : 21
Gender : Male
```

Example 3:

Multi-dimensional array

```
1. <?php
2.     //declare multi-dimensional array
3.     $a = array();
4.     $a[0][0] = "Alex";
5.     $a[0][1] = "Bob";
6.     $a[1][0] = "Camila";
7.     $a[1][1] = "Denial";
8.
9.     //display multi-dimensional array elements through foreach loop
10.    foreach ($a as $e1) {
```

```
11.     foreach ($e1 as $e2) {
12.         echo "$e2\n";
13.     }
14. }
15. ?>
```

Output:

```
Alex Bob Camila Denial
```

Example 4:

Dynamic array

```
1. <?php
2.     //dynamic array
3.     foreach (array('j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't') as $elements) {
4.         echo "$elements\n";
5.     }
6. ?>
```

Output:

```
j a v a t p o i n t
```

PHP While Loop

PHP while loop can be used to traverse set of code like for loop. The while loop executes a block of code repeatedly until the condition is FALSE. Once the condition gets FALSE, it exits from the body of loop.

It should be used if the number of iterations is not known.

The while loop is also called an **Entry control loop** because the condition is checked before entering the loop body. This means that first the condition is checked. If the condition is true, the block of code will be executed.

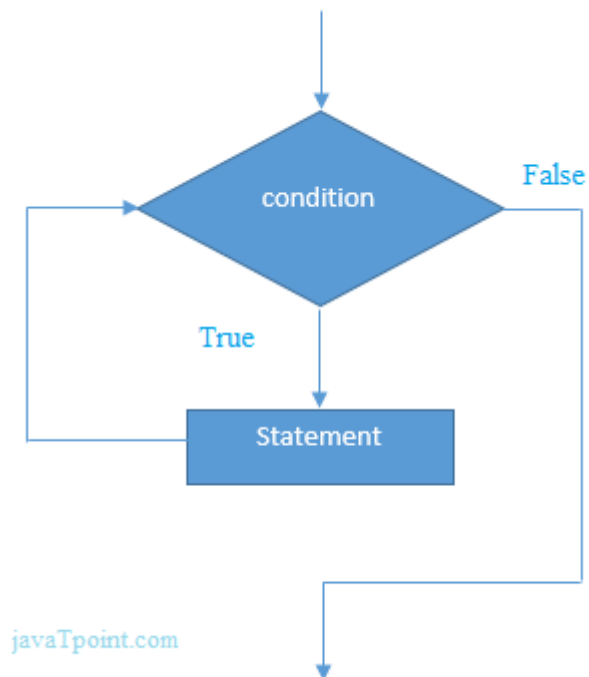
Syntax

```
1. while(condition){
2.     //code to be executed
3. }
```

Alternative Syntax

1. **while**(condition):
2. *//code to be executed*
- 3.
4. **endwhile**;

PHP While Loop Flowchart



PHP While Loop Example

1. `<?php`
2. `$n=1;`
3. **while**(\$n<=10){
4. `echo "$n
";`
5. `$n++;`
6. `}`
7. `?>`

Output:

```
1
2
3
4
5
6
7
8
9
```

10

Alternative Example

1. <?php
2. \$n=1;
3. **while**(\$n<=10):
4. echo "\$n
";
5. \$n++;
6. **endwhile**;
7. ?>

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example

Below is the example of printing alphabets using while loop.

1. <?php
2. \$i = 'A';
3. **while** (\$i < 'H') {
4. echo \$i;
5. \$i++;
6. echo "</br>";
7. }
8. ?>

Output:

```
A
B
C
D
E
F
G
```

PHP Nested While Loop

We can use while loop inside another while loop in PHP, it is known as nested while loop.

In case of inner or nested while loop, nested while loop is executed fully for one outer while loop. If outer while loop is to be executed for 3 times and nested while loop for 3 times, nested while loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example

```
1. <?php
2. $i=1;
3. while($i<=3){
4.     $j=1;
5.     while($j<=3){
6.         echo "$i $j<br/>";
7.         $j++;
8.     }
9.     $i++;
10. }
11. ?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP Infinite While Loop

If we pass **TRUE** in while loop, it will be an infinite loop.

Syntax

```
1. while(true) {
2.     //code to be executed
3. }
```

Example

```
1. <?php
2.     while (true) {
3.         echo "Hello Javatpoint!";
4.         echo "</br>";
5.     }
6. ?>
```

Output:

```
Hello Javatpoint!
Hello Javatpoint!
Hello Javatpoint!
Hello Javatpoint!
.
.
.
.
.
Hello Javatpoint!
Hello Javatpoint!
```

PHP do-while loop

PHP do-while loop can be used to traverse set of code like php while loop. The PHP do-while loop is guaranteed to run at least once.

The PHP do-while loop is used to execute a set of code of the program several times. If you have to execute the loop at least once and the number of iterations is not even fixed, it is recommended to use the **do-while** loop.

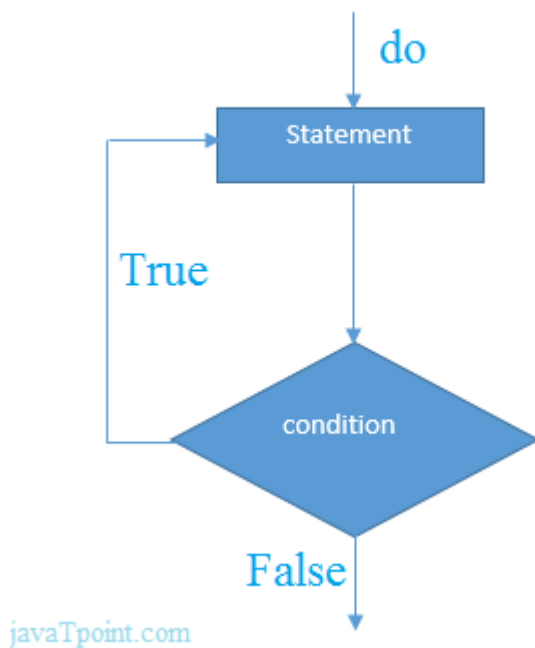
It executes the code at least one time always because the condition is checked after executing the code.

The do-while loop is very much similar to the while loop except the condition check. The main difference between both loops is that while loop checks the condition at the beginning, whereas do-while loop checks the condition at the end of the loop.

Syntax

```
1. do{
2.     //code to be executed
3. }while(condition);
```

Flowchart



Example

1. <?php
2. \$n=1;
3. **do**{
4. echo "\$n
";
5. \$n++;
6. }**while**(\$n<=10);
7. ?>

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example

A semicolon is used to terminate the do-while loop. If you don't use a semicolon after the do-while loop, it is must that the program should not contain any other statements after the do-while loop. In this case, it will not generate any error.


```
1. <?php
2.     $x = 5;
3.     do {
4.         echo "Welcome to javatpoint! </br>";
5.         $x++;
6.     } while ($x < 10);
7. ?>
```

Output:

```
Welcome to javatpoint!
Welcome to javatpoint!
Welcome to javatpoint!
Welcome to javatpoint!
Welcome to javatpoint!
```

Example

The following example will increment the value of \$x at least once. Because the given condition is false.

```
1. <?php
2.     $x = 1;
3.     do {
4.         echo "1 is not greater than 10.";
5.         echo "</br>";
6.         $x++;
7.     } while ($x > 10);
8.     echo $x;
9. ?>
```

Output:

```
1 is not greater than 10.
2
```

Difference between while and do-while loop

while Loop	do-while loop
The while loop is also named as entry control loop .	The do-while loop is also named as exit control loop .

The body of the loop does not execute if the condition is false.	The body of the loop executes at least once, even if the condition is false.
Condition checks first, and then block of statements executes.	Block of statements executes first and then condition checks.
This loop does not use a semicolon to terminate the loop.	Do-while loop use semicolon to terminate the loop.

PHP Break

PHP break statement breaks the execution of the current for, while, do-while, switch, and for-each loop. If you use break inside inner loop, it breaks the execution of inner loop only.

The **break** keyword immediately ends the execution of the loop or switch structure. It breaks the current flow of the program at the specified condition and program control resumes at the next statements outside the loop.

The break statement can be used in all types of loops such as while, do-while, for, foreach loop, and also with switch case.

Syntax

1. jump statement;
2. **break**;

Flowchart

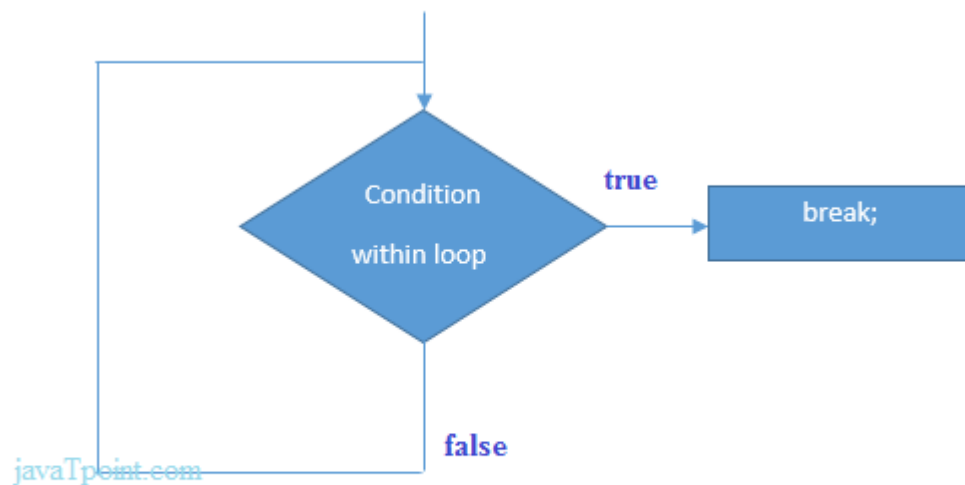


Figure: Flowchart of break statement

PHP Break: inside loop

Let's see a simple example to break the execution of for loop if value of i is equal to 5.

```
1. <?php
2. for($i=1;$i<=10;$i++){
3. echo "$i <br/>";
4. if($i==5){
5. break;
6. }
7. }
8. ?>
```

Output:

```
1
2
3
4
5
```

PHP Break: inside inner loop

The PHP break statement breaks the execution of inner loop only.

```
1. <?php
2. for($i=1;$i<=3;$i++){
3.     for($j=1;$j<=3;$j++){
4.         echo "$i $j<br/>";
5.         if($i==2 && $j==2){
6.             break;
7.         }
8.     }
9. }
10. ?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

PHP Break: inside switch statement

The PHP break statement breaks the flow of switch case also.

```
1. <?php
2. $num=200;
3. switch($num){
4.     case 100:
5.         echo("number is equals to 100");
6.         break;
7.     case 200:
8.         echo("number is equal to 200");
9.         break;
10.    case 50:
11.        echo("number is equal to 300");
12.        break;
13.    default:
14.        echo("number is not equal to 100, 200 or 500");
15. }
```

16. ?>

Output:

```
number is equal to 200
```

PHP Break: with array of string

```
1. <?php
2. //declare an array of string
3. $number = array ("One", "Two", "Three", "Stop", "Four");
4. foreach ($number as $element) {
5.     if ($element == "Stop") {
6.         break;
7.     }
8.     echo "$element </br>";
9. }
10. ?>
```

Output:

```
One
Two
Three
```

You can see in the above output, after getting the specified condition true, break statement immediately ends the loop and control is came out from the loop.

PHP Break: switch statement without break

It is not essential to break out of all cases of a switch statement. But if you want that only one case to be executed, you have to use break statement.

```
1. <?php
2. $car = 'Mercedes Benz';
3. switch ($car) {
4.     default:
5.         echo '$car is not Mercedes Benz<br>';
6.         case 'Orange':
7.             echo '$car is Mercedes Benz';
8.         }
9. ?>
```

Output:

```
$car is not Mercedes Benz  
$car is Mercedes Benz
```

PHP Break: using optional argument

The break accepts an optional numeric argument, which describes how many nested structures it will exit. The default value is 1, which immediately exits from the enclosing structure.

```
1. <?php  
2. $i = 0;  
3. while (++$i) {  
4.     switch ($i) {  
5.         case 5:  
6.             echo "At matched condition i = 5<br />\n";  
7.             break 1; // Exit only from the switch.  
8.         case 10:  
9.             echo "At matched condition i = 10; quitting<br />\n";  
10.            break 2; // Exit from the switch and the while.  
11.        default:  
12.            break;  
13.    }  
14. }?>
```

Output:

```
At matched condition i = 5  
At matched condition i = 10; quitting
```

PHP continue statement

The PHP continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

The continue statement is used within looping and switch control structure when you immediately jump to the next iteration.

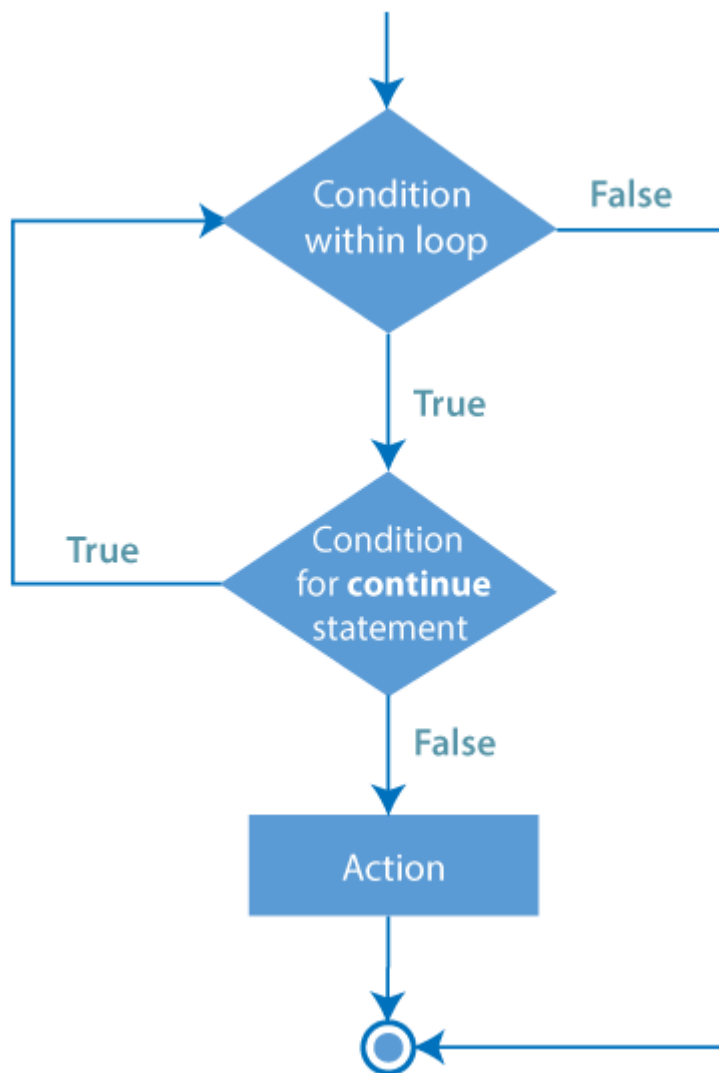
The continue statement can be used with all types of loops such as - for, while, do-while, and foreach loop. The continue statement allows the user to skip the execution of the code for the specified condition.

Syntax

The syntax for the continue statement is given below:

1. jump-statement;
2. **continue**;

Flowchart:



PHP Continue Example with for loop

Example

In the following example, we will print only those values of i and j that are same and skip others.

1. `<?php`
2. `//outer loop`

```
3.  for ($i = 1; $i <= 3; $i++) {
4.      //inner loop
5.      for ($j = 1; $j <= 3; $j++) {
6.          if (!($i == $j)) {
7.              continue;    //skip when i and j does not have same values
8.          }
9.          echo $i.$j;
10.         echo "</br>";
11.     }
12. }
13. ?>
```

Output:

```
11
22
33
```

PHP continue Example in while loop

Example

In the following example, we will print the even numbers between 1 to 20.

```
1.  <?php
2.      //php program to demonstrate the use of continue statement
3.
4.      echo "Even numbers between 1 to 20: </br>";
5.      $i = 1;
6.      while ($i <= 20) {
7.          if ($i % 2 == 1) {
8.              $i++;
9.              continue; //here it will skip rest of statements
10.         }
11.         echo $i;
12.         echo "</br>";
13.         $i++;
14.     }
15. ?>
```


Output:

```
Even numbers between 1 to 20:  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20
```

PHP continue Example with array of string

Example

The following example prints the value of array elements except those for which the specified condition is true and continue statement is used.

```
1. <?php  
2.   $number = array ("One", "Two", "Three", "Stop", "Four");  
3.   foreach ($number as $element) {  
4.       if ($element == "Stop") {  
5.           continue;  
6.       }  
7.       echo "$element </br>";  
8.   }  
9. ?>
```

Output:

```
One  
Two  
Three  
Four
```

PHP continue Example with optional argument

The continue statement accepts an optional numeric value, which is used accordingly. The numeric value describes how many nested structures it will exit.

Example

Look at the below example to understand it better:

```
1. <?php
2.     //outer loop
3.     for ($i = 1; $i <= 3; $i++) {
4.         //inner loop
5.         for ($j = 1; $j <= 3; $j++) {
6.             if (($i == $j)) { //skip when i and j have same values
7.                 continue 1; //exit only from inner for loop
8.             }
9.             echo $i.$j;
10.            echo "</br>";
11.        }
12.    }
13. ?>
```

Output:

```
12
13
21
23
31
32
```

PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

Advantage of PHP Functions

Code Reusability: PHP functions are defined only once and can be invoked many times, like in other programming languages.

Less Code: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

Easy to understand: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare user-defined functions.

Syntax

1. **function** functionname(){
2. *//code to be executed*
3. }

PHP Functions Example

File: function1.php

1. <?php
2. **function** sayHello(){
3. echo "Hello PHP Function";
4. }
5. sayHello();//calling function
6. ?>

Output:

```
Hello PHP Function
```

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**.

Let's see the example to pass single argument in PHP function.

File: functionarg.php

```
1. <?php
2. function sayHello($name){
3.     echo "Hello $name<br/>";
4. }
5. sayHello("Sonoo");
6. sayHello("Vimal");
7. sayHello("John");
8. ?>
```

Output:

```
Hello Sonoo
Hello Vimal
Hello John
```

Let's see the example to pass two argument in PHP function.

File: *functionarg2.php*

```
1. <?php
2. function sayHello($name,$age){
3.     echo "Hello $name, you are $age years old<br/>";
4. }
5. sayHello("Sonoo",27);
6. sayHello("Vimal",29);
7. sayHello("John",23);
8. ?>
```

Output:

```
Hello Sonoo, you are 27 years old
Hello Vimal, you are 29 years old
Hello John, you are 23 years old
```

PHP Call By Reference

Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Let's see a simple example of call by reference in PHP.

File: *functionref.php*

```
1. <?php
2. function adder(&$str2)
3. {
4.     $str2 .= 'Call By Reference';
5. }
6. $str = 'Hello ';
7. adder($str);
8. echo $str;
9. ?>
```

Output:

```
Hello Call By Reference
```

PHP Function: Default Argument Value

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

File: *functiondefaultarg.php*

```
1. <?php
2. function sayHello($name="Sonoo"){
3.     echo "Hello $name<br/>";
4. }
5. sayHello("Rajesh");
6. sayHello();//passing no value
7. sayHello("John");
8. ?>
```

Output:

```
Hello Rajesh
Hello Sonoo
Hello John
```

PHP Function: Returning Value

Let's see an example of PHP function that returns value.

File: *functiondefaultarg.php*

```
1. <?php
2. function cube($n){
3.     return $n*$n*$n;
4. }
5. echo "Cube of 3 is: ".cube(3);
6. ?>
```

Output:

```
Cube of 3 is: 27
```

PHP Call By Value

PHP allows you to call function by value and reference both. In case of PHP call by value, actual value is not modified if it is modified inside the function.

Let's understand the concept of call by value by the help of examples.

Example 1

In this example, variable \$str is passed to the adder function where it is concatenated with 'Call By Value' string. But, printing \$str variable results 'Hello' only. It is because changes are done in the local variable \$str2 only. It doesn't reflect to \$str variable.

```
1. <?php
2. function adder($str2)
3. {
4.     $str2 .= 'Call By Value';
5. }
6. $str = 'Hello ';
7. adder($str);
8. echo $str;
9. ?>
```

Output:

```
Hello
```

Example 2

Let's understand PHP call by value concept through another example.

```
1. <?php
2. function increment($i)
3. {
4.     $i++;
5. }
6. $i = 10;
7. increment($i);
8. echo $i;
9. ?>
```

Output:

```
10
```

PHP Call By Reference

In case of PHP call by reference, actual value is modified if it is modified inside the function. In such case, you need to use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

Let's understand the concept of call by reference by the help of examples.

Example 1

In this example, variable \$str is passed to the adder function where it is concatenated with 'Call By Reference' string. Here, printing \$str variable results 'This is Call By Reference'. It is because changes are done in the actual variable \$str.

```
1. <?php
2. function adder(&$str2)
3. {
4.     $str2 .= 'Call By Reference';
5. }
6. $str = 'This is ';
7. adder($str);
8. echo $str;
9. ?>
```

Output:

```
This is Call By Reference
```

Example 2

Let's understand PHP call by reference concept through another example.

```
1. <?php
2. function increment(&$i)
3. {
4.     $i++;
5. }
6. $i = 10;
7. increment($i);
8. echo $i;
9. ?>
```

Output:

```
11
```

PHP Default Argument Values Function

PHP allows you to define C++ style default argument values. In such case, if you don't pass any value to the function, it will use default argument value.

Let's see the simple example of using PHP default arguments in function.

Example 1

```
1. <?php
2. function sayHello($name="Ram"){
3.     echo "Hello $name<br/>";
4. }
5. sayHello("Sonoo");
6. sayHello();//passing no value
7. sayHello("Vimal");
8. ?>
```

Output:

```
Hello Sonoo
Hello Ram
Hello Vimal
```


Example 2

```
1. <?php
2. function greeting($first="Sonoo",$last="Jaiswal"){
3. echo "Greeting: $first $last<br/>";
4. }
5. greeting();
6. greeting("Rahul");
7. greeting("Michael","Clark");
8. ?>
```

Output:

```
Greeting: Sonoo Jaiswal
Greeting: Rahul Jaiswal
Greeting: Michael Clark
```

Example 3

```
1. <?php
2. function add($n1=10,$n2=10){
3. $n3=$n1+$n2;
4. echo "Addition is: $n3<br/>";
5. }
6. add();
7. add(20);
8. add(40,40);
9. ?>
```

Output:

```
Addition is: 20
Addition is: 30
Addition is: 80
```

PHP Variable Length Argument Function

PHP supports variable length argument function. It means you can pass 0, 1 or n number of arguments in function. To do so, you need to use 3 ellipses (dots) before the argument name.

The 3 dot concept is implemented for variable length argument since PHP 5.6.

Let's see a simple example of PHP variable length argument function.

```
1. <?php
2. function add(...$numbers) {
3.     $sum = 0;
4.     foreach ($numbers as $n) {
5.         $sum += $n;
6.     }
7.     return $sum;
8. }
9.
10. echo add(1, 2, 3, 4);
11. ?>
```

Output:

10

PHP Arrays

PHP array is an ordered map (contains value on the basis of key). It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

Less Code: We don't need to define multiple variables.

Easy to traverse: By the help of single loop, we can traverse all the elements of an array.

Sorting: We can sort the elements of array.

PHP Array Types

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array

3. Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

1st way:

1. `$season=array("summer","winter","spring","autumn");`

2nd way:

1. `$season[0]="summer";`
2. `$season[1]="winter";`
3. `$season[2]="spring";`
4. `$season[3]="autumn";`

Example

File: *array1.php*

1. `<?php`
2. `$season=array("summer","winter","spring","autumn");`
3. `echo "Season are: $season[0], $season[1], $season[2] and $season[3]";`
4. `?>`

Output:

```
Season are: summer, winter, spring and autumn
```

1. `<?php`
2. `$season[0]="summer";`
3. `$season[1]="winter";`
4. `$season[2]="spring";`
5. `$season[3]="autumn";`
6. `echo "Season are: $season[0], $season[1], $season[2] and $season[3]";`
7. `?>`

Output:

```
Season are: summer, winter, spring and autumn
```

PHP Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

1st way:

1. `$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");`

2nd way:

1. `$salary["Sonoo"]="350000";`
2. `$salary["John"]="450000";`
3. `$salary["Kartik"]="200000";`

Example

File: *arrayassociative1.php*

1. `<?php`
2. `$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");`
3. `echo "Sonoo salary: ".$salary["Sonoo"]."
";`
4. `echo "John salary: ".$salary["John"]."
";`
5. `echo "Kartik salary: ".$salary["Kartik"]."
";`
6. `?>`

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

File: *arrayassociative2.php*

1. `<?php`
2. `$salary["Sonoo"]="350000";`
3. `$salary["John"]="450000";`
4. `$salary["Kartik"]="200000";`
5. `echo "Sonoo salary: ".$salary["Sonoo"]."
";`
6. `echo "John salary: ".$salary["John"]."
";`

7. `echo "Kartik salary: ".$salary["Kartik"]."
";`
8. `?>`

Output:

```
Sonoo salary: 350000  
John salary: 450000  
Kartik salary: 200000
```

PHP Indexed Array

PHP indexed array is an array which is represented by an index number by default. All elements of array are represented by an index number which starts from 0.

PHP indexed array can store numbers, strings or any object. PHP indexed array is also known as numeric array.

Definition

There are two ways to define indexed array:

1st way:

1. `$size=array("Big","Medium","Short");`

2nd way:

1. `$size[0]="Big";`
2. `$size[1]="Medium";`
3. `$size[2]="Short";`

PHP Indexed Array Example

File: *array1.php*

1. `<?php`
2. `$size=array("Big","Medium","Short");`
3. `echo "Size: $size[0], $size[1] and $size[2]";`
4. `?>`

Output:

```
Size: Big, Medium and Short
```

File: *array2.php*

```
1. <?php
2. $size[0]="Big";
3. $size[1]="Medium";
4. $size[2]="Short";
5. echo "Size: $size[0], $size[1] and $size[2]";
6. ?>
```

Output:

```
Size: Big, Medium and Short
```

Traversing PHP Indexed Array

We can easily traverse array in PHP using foreach loop. Let's see a simple example to traverse all the elements of PHP array.

File: array3.php

```
1. <?php
2. $size=array("Big","Medium","Short");
3. foreach( $size as $s )
4. {
5.     echo "Size is: $s<br />";
6. }
7. ?>
```

Output:

```
Size is: Big
Size is: Medium
Size is: Short
```

Count Length of PHP Indexed Array

PHP provides count() function which returns length of an array.

```
1. <?php
2. $size=array("Big","Medium","Short");
3. echo count($size);
4. ?>
```

Output:

PHP Associative Array

PHP allows you to associate name/label with each array elements in PHP using => symbol. Such way, you can easily remember the element because each element is represented by label than an incremented number.

Definition

There are two ways to define associative array:

1st way:

1. `$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");`

2nd way:

1. `$salary["Sonoo"]="550000";`
2. `$salary["Vimal"]="250000";`
3. `$salary["Ratan"]="200000";`

Example

File: arrayassociative1.php

1. `<?php`
2. `$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");`
3. `echo "Sonoo salary: ".$salary["Sonoo"]."
";`
4. `echo "Vimal salary: ".$salary["Vimal"]."
";`
5. `echo "Ratan salary: ".$salary["Ratan"]."
";`
6. `?>`

Output:

```
Sonoo salary: 550000
Vimal salary: 250000
Ratan salary: 200000
```

File: arrayassociative2.php

1. `<?php`
2. `$salary["Sonoo"]="550000";`
3. `$salary["Vimal"]="250000";`
4. `$salary["Ratan"]="200000";`

5. `echo "Sonoo salary: ".$salary["Sonoo"]."
";`
6. `echo "Vimal salary: ".$salary["Vimal"]."
";`
7. `echo "Ratan salary: ".$salary["Ratan"]."
";`
8. `?>`

Output:

```
Sonoo salary: 550000
Vimal salary: 250000
Ratan salary: 200000
```

Traversing PHP Associative Array

By the help of PHP for each loop, we can easily traverse the elements of PHP associative array.

1. `<?php`
2. `$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");`
3. `foreach($salary as $k => $v) {`
4. `echo "Key: ".$k." Value: ".$v."
";`
5. `}`
6. `?>`

Output:

```
Key: Sonoo Value: 550000
Key: Vimal Value: 250000
Key: Ratan Value: 200000
```

PHP Multidimensional Array

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.

Definition

1. `$emp = array`
2. `(`
3. `array(1,"sonoo",400000),`
4. `array(2,"john",500000),`
5. `array(3,"rahul",300000)`

6.);

PHP Multidimensional Array Example

Let's see a simple example of PHP multidimensional array to display following tabular data. In this example, we are displaying 3 rows and 3 columns.

Id	Name	Salary
1	sonoo	400000
2	john	500000
3	rahul	300000

File: multiarray.php

```
1. <?php
2. $emp = array
3. (
4. array(1,"sonoo",400000),
5. array(2,"john",500000),
6. array(3,"rahul",300000)
7. );
8.
9. for ($row = 0; $row < 3; $row++) {
10. for ($col = 0; $col < 3; $col++) {
11. echo $emp[$row][$col]." ";
12. }
13. echo "<br/>";
14. }
15. ?>
```

Output:

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

PHP Array Functions

PHP provides various array functions to access and manipulate the elements of array. The important PHP array functions are given below.

1) PHP array() function

PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.

Syntax

1. **array array** ([mixed \$...])

Example

Play Video

1. <?php
2. \$season=**array**("summer","winter","spring","autumn");
3. echo "Season are: \$season[0], \$season[1], \$season[2] and \$season[3]";
4. ?>

Output:

Season are: summer, winter, spring and autumn

2) PHP array_change_key_case() function

PHP array_change_key_case() function changes the case of all key of an array.

Note: It changes case of key only.

Syntax

1. **array** array_change_key_case (**array** \$array [, int \$case = CASE_LOWER])

Example

1. <?php
2. \$salary=**array**("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
3. print_r(array_change_key_case(\$salary,CASE_UPPER));
4. ?>

Output:

```
Array ( [SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000 )
```

Example

1. <?php
2. \$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
3. print_r(array_change_key_case(\$salary,CASE_LOWER));
4. ?>

Output:

```
Array ( [sonoo] => 550000 [vimal] => 250000 [ratan] => 200000 )
```

3) PHP array_chunk() function

PHP array_chunk() function splits array into chunks. By using array_chunk() method, you can divide array into many parts.

Syntax

1. **array** array_chunk (**array** \$array , int \$size [, bool \$preserve_keys = false])

Example

1. <?php
2. \$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
3. print_r(array_chunk(\$salary,2));
4. ?>

Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 )
)
```

4) PHP count() function

PHP count() function counts all elements in an array.

Syntax

1. `int count (mixed $array_or_countable [, int $mode = COUNT_NORMAL])`

Example

1. `<?php`
2. `$season=array("summer","winter","spring","autumn");`
3. `echo count($season);`
4. `?>`

Output:

```
4
```

5) PHP sort() function

PHP sort() function sorts all the elements in an array.

Syntax

1. `bool sort (array &$array [, int $sort_flags = SORT_REGULAR])`

Example

1. `<?php`
2. `$season=array("summer","winter","spring","autumn");`
3. `sort($season);`
4. `foreach($season as $s)`
5. `{`
6. `echo "$s
";`
7. `}`
8. `?>`

Output:

```
autumn
spring
summer
winter
```

6) PHP array_reverse() function

PHP `array_reverse()` function returns an array containing elements in reversed order.

Syntax

1. **array** `array_reverse (array $array [, bool $preserve_keys = false])`

Example

```
1. <?php
2. $season=array("summer","winter","spring","autumn");
3. $reverseseason=array_reverse($season);
4. foreach( $reverseseason as $s )
5. {
6.     echo "$s<br />";
7. }
8. ?>
```

Output:

```
autumn
spring
winter
summer
```

7) PHP `array_search()` function

PHP `array_search()` function searches the specified value in an array. It returns key if search is successful.

Syntax

1. `mixed array_search (mixed $needle , array $haystack [, bool $strict = false])`

Example

```
1. <?php
2. $season=array("summer","winter","spring","autumn");
3. $key=array_search("spring",$season);
4. echo $key;
5. ?>
```

Output:

8) PHP array_intersect() function

PHP array_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.

Syntax

1. **array** array_intersect (**array** \$array1 , **array** \$array2 [, **array** \$...])

Example

1. <?php
2. \$name1=**array**("sonoo","john","vivek","smith");
3. \$name2=**array**("umesh","sonoo","kartik","smith");
4. \$name3=array_intersect(\$name1,\$name2);
5. **foreach**(\$name3 **as** \$n)
6. {
7. echo "\$n
";
8. }
9. ?>

Output:

```
sonoo  
smith
```

PHP String

PHP string is a sequence of characters i.e., used to store and manipulate text. PHP supports only 256-character set and so that it does not offer native Unicode support. There are 4 ways to specify a string literal in PHP.

1. single quoted
2. double quoted
3. heredoc syntax
4. newdoc syntax (since PHP 5.3)

Single Quoted

We can create a string in PHP by enclosing the text in a single-quote. It is the easiest way to specify string in PHP.

For specifying a literal single quote, escape it with a backslash (\) and to specify a literal backslash (\) use double backslash (\\). All the other instances with backslash such as \r or \n, will be output same as they specified instead of having any special meaning.

For Example

Following some examples are given to understand the single quoted PHP String in a better way:

Example 1

```
1. <?php
2.     $str='Hello text within single quote';
3.     echo $str;
4. ?>
```

Output:

```
Hello text within single quote
```

We can store multiple line text, special characters, and escape sequences in a single-quoted PHP string.

Example 2

```
1. <?php
2. $str1='Hello text
3. multiple line
4. text within single quoted string';
5. $str2='Using double "quote" directly inside single quoted string';
6. $str3='Using escape sequences \n in single quoted string';
7. echo "$str1 <br/> $str2 <br/> $str3";
8. ?>
```

Output:

```
Hello text multiple line text within single quoted string
Using double "quote" directly inside single quoted string
Using escape sequences \n in single quoted string
```

Example 3

1. <?php
2. \$num1=10;
3. \$str1='trying variable \$num1';
4. \$str2='trying backslash n and backslash t inside single quoted string \n \t';
5. \$str3='Using single quote \'my quote\' and \\backslash';
6. echo "\$str1
 \$str2
 \$str3";
7. ?>

Output:

```
trying variable $num1
trying backslash n and backslash t inside single quoted string \n \t
Using single quote 'my quote' and \backslash
```

Double Quoted

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

Example 1

1. <?php
2. \$str="Hello text within double quote";
3. echo \$str;
4. ?>

Output:

```
Hello text within double quote
```

Now, you **can't use double quote directly** inside double quoted string.

Example 2

1. <?php
2. \$str1="Using double "quote" directly inside double quoted string";
3. echo \$str1;
4. ?>

Output:


```
Parse error: syntax error, unexpected 'quote' (T_STRING) in  
C:\wamp\www\string1.php on line 2
```

We **can store multiple line text, special characters and escape sequences** in a double quoted PHP string.

Example 3

1. <?php
2. \$str1="Hello text
3. multiple line
4. text within double quoted string";
5. \$str2="Using double \"quote\" with backslash inside double quoted string";
6. \$str3="Using escape sequences \n in double quoted string";
7. echo "\$str1
 \$str2
 \$str3";
8. ?>

Output:

```
Hello text multiple line text within double quoted string  
Using double "quote" with backslash inside double quoted string  
Using escape sequences in double quoted string
```

In double quoted strings, **variable will be interpreted**.

Example 4

1. <?php
2. \$num1=10;
3. echo "Number is: \$num1";
4. ?>

Output:

```
Number is: 10
```

Heredoc

Heredoc syntax (<<<) is the third way to delimit strings. In Heredoc syntax, an identifier is provided after this heredoc <<< operator, and immediately a new line is started to write any text. To close the quotation, the string follows itself and then again that same identifier is provided. That closing identifier must begin from the new line without any whitespace or tab.

Naming Rules

The identifier should follow the naming rule that it must contain only alphanumeric characters and underscores, and must start with an underscore or a non-digit character.

For Example

Valid Example

1. <?php
2. \$str = <<<Demo
3. It is a valid example
4. Demo; //Valid code as whitespace or tab is not valid before closing identifier
5. echo \$str;
6. ?>

Output:

```
It is a valid example
```

Invalid Example

We cannot use any whitespace or tab before and after the identifier and semicolon, which means identifier must not be indented. The identifier must begin from the new line.

1. <?php
2. \$str = <<<Demo
3. It is Invalid example
4. Demo; //Invalid code as whitespace or tab is not valid before closing identifier
5. echo \$str;
6. ?>

This code will generate an error.

Output:

```
Parse error: syntax error, unexpected end of file in  
C:\xampp\htdocs\xampp\PMA\heredoc.php on line 7
```

Heredoc is similar to the double-quoted string, without the double quote, means that quote in a heredoc are not required. It can also print the variable's value.

Example

```
1. <?php
2.     $city = 'Delhi';
3.     $str = <<<DEMO
4. Hello! My name is Misthi, and I live in $city.
5. DEMO;
6.     echo $str;
7. ?>
```

Output:

```
Hello! My name is Misthi, and I live in Delhi.
```

Example

We can add multiple lines of text here between heredoc syntax.

```
1. <?php
2.     $str = <<<DEMO
3. It is the example
4. of multiple
5. lines of text.
6. DEMO;
7.     echo $str;
8.
9. echo '</br>';
10.
11. echo <<<DEMO // Here we are not storing string content in variable str.
12. It is the example
13. of multiple
14. lines of text.
15. DEMO;
16. ?>
```

Output:

```
It is the example of multiple lines of text.
It is the example of multiple lines of text.
```

Below are the example with class and their variable

Example

```
1. <?php
2. class heredocExample{
3.     var $demo;
4.     var $example;
5.     function __construct()
6.     {
7.         $this->demo = 'DEMO';
8.         $this->example = array('Example1', 'Example2', 'Example3');
9.     }
10. }
11. $heredocExample = new heredocExample();
12. $name = 'Gunjan';
13.
14. echo <<<ECO
15. My name is "$name". I am printing some $heredocExample-
    >demo example.
16. Now, I am printing {$heredocExample->example[1]}.
17. It will print a capital 'A': \x41
18. ECO;
19. ?>
```

Output:

```
My name is "Gunjan". I am printing some DEMO example.
Now, I am printing Example2.
It will print a capital 'A': A
```

Newdoc

Newdoc is similar to the heredoc, but in newdoc parsing is not done. It is also identified with three less than symbols <<< followed by an identifier. But here identifier is enclosed in single-quote, **e.g.** <<<'EXP'. Newdoc follows the same rule as heredocs.

The difference between newdoc and heredoc is that - Newdoc is a **single-quoted string** whereas heredoc is a **double-quoted string**.

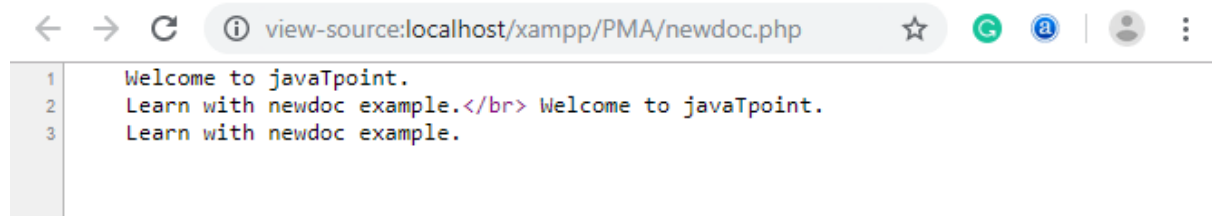
Example-1:

```
1. <?php
2.     $str = <<<'DEMO'
3.     Welcome to javaTpoint.
4.         Learn with newdoc example.
5. DEMO;
6. echo $str;
7. echo '</br>';
8.
9. echo <<< 'Demo' // Here we are not storing string content in variable str.
10. Welcome to javaTpoint.
11.     Learn with newdoc example.
12. Demo;
13. ?>
```

Output:

```
Welcome to javaTpoint. Learn with newdoc example.
Welcome to javaTpoint. Learn with newdoc example.
```

Go to view page source and see the source of the program.



Example

The below example shows that newdoc does not print the variable's value.

```
1. <?php
2. class heredocExample{
3.     var $demo;
4.     var $example;
5.     function __construct()
6.     {
7.         $this->demo = 'DEMO';
8.         $this->example = array('Example1', 'Example2', 'Example3');
9.     }
```

```

10. }
11. $heredocExample = new heredocExample();
12. $name = 'Gunjan';
13.
14. echo <<<ECO
15. My name is "$name". I am printing some $heredocExample-
    >demo example.
16. Now, I am printing {$heredocExample->example[1]}.
17. It will print a capital 'A': \x41
18. ECO;
19. ?>

```

PHP String Functions

PHP provides various string functions to access and manipulate strings.

A list of PHP string functions are given below.

<u>addslashes()</u>	It is used to return a string with backslashes.
<u>addslashes()</u>	It is used to return a string with backslashes.
<u>bin2hex()</u>	It is used to converts a string of ASCII characters to hexadecimal values.
<u>chop()</u>	It removes whitespace or other characters from the right end of a string
<u>chr()</u>	It is used to return a character from a specified ASCII value.
<u>chunk_split()</u>	It is used to split a string into a series of smaller parts.
<u>convert_cyr_string()</u>	It is used to convert a string from one Cyrillic character-set to another.
<u>convert_uudecode()</u>	It is used to decode a uuencoded string.

<u>convert_uuencode()</u>	It is used to encode a string using the uuencode algorithm.
<u>count_chars()</u>	It is used to return information about characters used in a string.
<u>crc32()</u>	It is used to calculate a 32-bit CRC for a string.
<u>crypt()</u>	It is used to create hashing string One-way.
<u>echo()</u>	It is used for output one or more strings.
<u>explode()</u>	It is used to break a string into an array.
<u>fprint()</u>	It is used to write a formatted string to a stream.
<u>get_html_translation_table()</u>	Returns translation table which is used by htmlspecialchars() and htmlentities().
<u>hebrew()</u>	It is used to convert Hebrew text to visual text.
<u>hebrevc()</u>	It is used to convert Hebrew text to visual text and new lines (\n) into .
<u>hex2bin()</u>	It is used to convert string of hexadecimal values to ASCII characters.
<u>htmlentities()</u>	It is used to convert character to HTML entities.
<u>html_entity_decode()</u>	It is used to convert HTML entities to characters.
<u>htmlspecialchars()</u>	Converts the special characters to html entities.
<u>htmlspecialchars_decode()</u>	Converts the html entities back to special characters.
<u>implode()</u>	It is used to return a string from the elements of an array.
<u>Join()</u>	It is the Alias of implode() function.

<u>Levenshtein()</u>	It is used to return the Levenshtein distance between two strings.
<u>Lcfirst()</u>	It is used to convert the first character of a string to lowercase.
<u>localeconv()</u>	Get numeric formatting information
<u>ltrim()</u>	It is used to remove whitespace from the left side of a string.
<u>md5()</u>	It is used to calculate the MD5 hash of a string.
<u>md5_files()</u>	It is used to calculate MD5 hash of a file.
<u>metaphone()</u>	It is used to calculate the metaphone key of a string.
<u>money_format()</u>	It is used to return a string formatted as a currency string.
<u>nl2br()</u>	It is used to insert HTML line breaks in front of each newline in a string.
<u>nl_langinfo()</u>	Query language and locale information
<u>number_format()</u>	It is used to format a number with grouped thousands.
<u>ord()</u>	It is used to return ASCII value of the first character of a string.
<u>parse_str()</u>	It is used to parse a query string into variables.
<u>print()</u>	It is used for output one or more strings.
<u>printf()</u>	It is used to show output as a formatted string.
<u>quoted_printable_decode()</u>	Converts quoted-printable string to an 8-bit string
<u>quoted_printable_encode()</u>	Converts the 8-bit string back to quoted-printable string
<u>quotemeta()</u>	Quote meta characters

<code>rtrim()</code>	It is used to remove whitespace from the right side of a string.
<code>setlocale()</code>	It is used to set locale information.
<code>sha1()</code>	It is used to return the SHA-1 hash of a string.
<code>sha1_file()</code>	It is used to return the SHA-1 hash of a file.
<code>similar_text()</code>	It is used to compare the similarity between two strings.
<code>Soundex()</code>	It is is used to calculate the soundex key of a string.
<code>sprintf()</code>	Return a formatted string
<code>sscanf()</code>	It is used to parse input from a string according to a format.
<code>strcasecmp()</code>	It is used to compare two strings.
<code>strchr()</code>	It is used to find the first occurrence of a string inside another string.
<code>strcmp()</code>	Binary safe string comparison (case-sensitive)
<code>strcoll()</code>	Locale based binary comparison(case-sensitive)
<code>strcspn()</code>	It is used to reverses a string.
<code>stripslashes()</code>	It is used to unquote a string quoted with addslashes().
<code>stripos()</code>	It is used to return the position of the first occurrence of a string inside another string.
<code>stristr()</code>	Case-insensitive strstr
<code>strlen()</code>	It is used to return the length of a string.
<code>strncasecmp()</code>	Binary safe case-insensitive string comparison

<u>strnatcasecmp()</u>	It is used for case-insensitive comparison of two strings using a "natural order" algorithm
<u>strnatcmp()</u>	It is used for case-sensitive comparison of two strings using a "natural order" algorithm
<u>strncmp()</u>	It is used to compare of the first n characters.
<u>strpbrk()</u>	It is used to search a string for any of a set of characters.
<u>strrpos()</u>	It finds the position of the last occurrence of a case-insensitive substring in a string.
<u>strrpos()</u>	It finds the length of the last occurrence of a substring in a string.
<u>strpos()</u>	It is used to return the position of the first occurrence of a string inside another string.
<u>strrchr()</u>	It is used to find the last occurrence of a string inside another string.
<u>strrev()</u>	It is used to reverse a string.
<u>strspn()</u>	Find the initial length of the initial segment of the string
<u>strstr()</u>	Find the occurrence of a string.
<u>strtok()</u>	Splits the string into smaller strings
<u>strtolower()</u>	Convert the string in lowercase
<u>strtoupper()</u>	Convert the strings in uppercase
<u>strtr()</u>	Translate certain characters in a string or replace the substring
<u>str_getcsv()</u>	It is used to parse a CSV string into an array.

<u>str_ireplace()</u>	It is used to replace some characters in a string (case-insensitive).
<u>str_pad()</u>	It is used to pad a string to a new length.
<u>str_repeat()</u>	It is used to repeat a string a specified number of times.
<u>str_replace()</u>	It replaces all occurrences of the search string with the replacement string.
<u>str_rot13()</u>	It is used to perform the ROT13 encoding on a string.
<u>str_shuffle()</u>	It is used to randomly shuffle all characters in a string.
<u>str_split()</u>	It is used to split a string into an array.
<u>strcoll()</u>	It is locale based string comparison.
<u>strip_tags()</u>	It is used to strip HTML and PHP tags from a string.
<u>str_word_count()</u>	It is used to count the number of words in a string.
<u>substr()</u>	Return the part of a string
<u>substr_compare()</u>	Compares two strings from an offset up to the length of characters. (Binary safe comparison)
<u>substr_count()</u>	Count the number of times occurrence of a substring
<u>substr_replace()</u>	Replace some part of a string with another substring
<u>trim()</u>	Remove whitespace or other characters from the beginning and end of the string.
<u>ucfirst()</u>	Make the first character of the string to uppercase
<u>ucwords()</u>	Make the first character of each word in a string to uppercase

vfprintf()	Write a formatted string to a stream
vprintf()	Display the output as a formatted string according to format
vsprintf()	It returns a formatted string
wordwrap()	Wraps a string to a given number of characters

PHP String Function Examples

1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax

1. string strtolower (string \$string)

Example

1. <?php
2. \$str="My name is KHAN";
3. \$str=strtolower(\$str);
4. echo \$str;
5. ?>

Output:

```
my name is khan
```

2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

Syntax

1. string strtoupper (string \$string)

Example

1. <?php

2. `$str="My name is KHAN";`
3. `$str=strtoupper($str);`
4. `echo $str;`
5. `?>`

Output:

```
MY NAME IS KHAN
```

3) PHP ucfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

Syntax

1. `string ucfirst (string $str)`

Example

1. `<?php`
2. `$str="my name is KHAN";`
3. `$str=ucfirst($str);`
4. `echo $str;`
5. `?>`

Output:

```
My name is KHAN
```

4) PHP lcfirst() function

The lcfirst() function returns string converting first character into lowercase. It doesn't change the case of other characters.

Syntax

1. `string lcfirst (string $str)`

Example

1. `<?php`
2. `$str="MY name IS KHAN";`
3. `$str=lcfirst($str);`

4. `echo $str;`
5. `?>`

Output:

```
mY name IS KHAN
```

5) PHP ucwords() function

The `ucwords()` function returns string converting first character of each word into uppercase.

Syntax

1. `string ucwords (string $str)`

Example

1. `<?php`
2. `$str="my name is Sonoo jaiswal";`
3. `$str=ucwords($str);`
4. `echo $str;`
5. `?>`

Output:

```
My Name Is Sonoo Jaiswal
```

6) PHP strrev() function

The `strrev()` function returns reversed string.

Syntax

1. `string strrev (string $string)`

Example

1. `<?php`
2. `$str="my name is Sonoo jaiswal";`
3. `$str=strrev($str);`
4. `echo $str;`
5. `?>`

Output:

lawsiaj oonoS si eman ym

7) PHP strlen() function

The strlen() function returns length of the string.

Syntax

1. int strlen (string \$string)

Example

1. <?php
2. \$str="my name is Sonoo jaiswal";
3. \$str=strlen(\$str);
4. echo \$str;
5. ?>

Output:

24