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 threads 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

# 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

```
<?php
echo "Hello by PHP echo";
?>
```

**Output:**

```
Hello by PHP echo
```

```
<?php
echo "Hello by PHP echo
this is multi line
text printed by
PHP echo statement
";
?>
```

**Output:**

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

```
<?php
echo "Hello escape \"sequence\" characters";
?>
```

**Output:**

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

## PHP echo: printing variable value

```php
<?php
$msg="Hello JavaTpoint PHP";
echo "Message is: $msg";
?>
```

**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

```php
<?php
print "Hello by PHP print ";
print ("Hello by PHP print()");
?>
```

**Output:**

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

## PHP print: printing multi line string

```
<?php
print "Hello by PHP print
this is multi line
text printed by
PHP print statement
";
?>
```

**Output:**

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

## PHP print: printing escaping characters

```
<?php
print "Hello escape \"sequence\" characters by PHP print";
?>
```

**Output:**

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

# PHP print: printing variable value

```
<?php
$msg="Hello print() in PHP";
print "Message is: $msg";
?>
```

**Output:**

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

# Difference between echo and print

## echo

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

- echo is faster than print statement.

## print

- print is also 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 echo statement.

# 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.

```
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

**Output:**

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

## PHP Variable: Sum of two variables

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

**Output:**

```
11
```

## PHP Variable: case sensitive

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

```
<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
```

```
?>
```

**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.

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)

echo "$a <br/> $_b";
?>
```

**Output:**

```
hello
hello
```

```
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$4c <br/> $*d";
?>
```

**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:

```
<?php
  function local_var()
  {
    $num = 45;  //local variable
    echo "Local variable declared inside the function is: ". $num;
  }
  local_var();
?>
```

**Output:**

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

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

```
    }
    mytest();
    //using $lang (local variable) outside the function will generate an error
    echo $lang;
?>
```

**Output:**

```
Web development language: PHP
Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 28
```

# Global variable

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

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

## Example:

```
<?php
    $name = "Sanaya Sharma";       //Global Variable
    function global_var()
    {
        global $name;
        echo "Variable inside the function: ". $name;
        echo "</br>";
    }
    global_var();
    echo "Variable outside the function: ". $name;
?>
```

**Output:**

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

## Example:

```
<?php
```

```
    $name = "Sanaya Sharma";       //global variable
    function global_var()
    {
        echo "Variable inside the function: ". $name;
        echo "</br>";
    }
    global_var();
?>
```

**Output:**

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

## 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:

```
<?php
    function static_var()
    {
        static $num1 = 3;       //static variable
        $num2 = 6;          //Non-static variable
        //increment in non-static variable
        $num1++;
        //increment in static variable
        $num2++;
        echo "Static: " .$num1 ."</br>";
        echo "Non-static: " .$num2 ."</br>";
    }

//first function call
```

```
    static_var();

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

**Output:**

```
Static: 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 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:**

```
<?php
  if (TRUE)
    echo "This condition is TRUE.";
  if (FALSE)
    echo "This condition is FALSE.";
?>
```

**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:**

```php
<?php
   $dec1 = 34;
   $oct1 = 0243;
   $hexa1 = 0x45;
   echo "Decimal number: " .$dec1. "</br>";
   echo "Octal number: " .$oct1. "</br>";
   echo "HexaDecimal number: " .$hexa1. "</br>";
?>
```

**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:**

```php
<?php
   $n1 = 19.34;
   $n2 = 54.472;
   $sum = $n1 + $n2;
   echo "Addition of floating numbers: " .$sum;
?>
```

**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:**

```php
<?php
   $company = "Javatpoint";
   //both single and double quote statements will treat different
   echo "Hello $company";
   echo "</br>";
   echo 'Hello $company';
?>
```

**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.

```php
<?php
   $bikes = array ("Royal Enfield", "Yamaha", "KTM");
   var_dump($bikes);   //the var_dump() function returns the datatype and values
   echo "</br>";
   echo "Array Element1: $bikes[0] </br>";
   echo "Array Element2: $bikes[1] </br>";
   echo "Array Element3: $bikes[2] </br>";
?>
```

**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
```

## PHP object

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

**Example:**

```php
<?php
```

```
    class bike {
        function model() {
            $model_name = "Royal Enfield";
            echo "Bike Model: " .$model_name;
          }
    }
    $obj = new bike();
    $obj -> model();
?>
```

**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:**

```
<?php
    $nl = NULL;
    echo $nl;   //it will not give any output
?>
```

# 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 | $a = $b | The value of right operand is assigned to the left op |
| += | Add then Assign | $a += $b | Addition same as $a = $a + $b |
| -= | Subtract then Assign | $a -= $b | Subtraction same as $a = $a - $b |
| *= | Multiply then Assign | $a *= $b | Multiplication same as $a = $a * $b |
| /= | Divide then Assign (quotient) | $a /= $b | Find quotient same as $a = $a / $b |
| %= | Divide then Assign (remainder) | $a %= $b | Find remainder same as $a = $a % $b |

## 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 |
|---|---|---|---|
| & | And | $a & $b | Bits that are 1 in both $a and $b are set to 1, otherwis |
| \| | Or (Inclusive or) | $a \| $b | Bits that are 1 in either $a or $b are set to 1 |
| ^ | Xor (Exclusive or) | $a ^ $b | Bits that are 1 in either $a or $b are set to 0. |
| ~ | Not | ~$a | Bits that are 1 set to 0 and bits that are 0 are set to 1 |

| | | | |
|---|---|---|---|
| << | Shift left | $a << $b | Left shift the bits of operand $a $b steps |
| >> | Shift right | $a >> $b | Right shift the bits of $a operand by $b number of pla |

## 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 |
|---|---|---|---|
| == | Equal | $a == $b | Return TRUE if $a is equal to $b |
| === | Identical | $a === $b | Return TRUE if $a is equal to $b, and they are of same |
| !== | Not identical | $a !== $b | Return TRUE if $a is not equal to $b, and they are not type |
| != | Not equal | $a != $b | Return TRUE if $a is not equal to $b |
| <> | Not equal | $a <> $b | Return TRUE if $a is not equal to $b |
| < | Less than | $a < $b | Return TRUE if $a is less than $b |
| > | Greater than | $a > $b | Return TRUE if $a is greater than $b |
| <= | Less than or equal to | $a <= $b | Return TRUE if $a is less than or equal $b |
| >= | Greater than or equal to | $a >= $b | Return TRUE if $a is greater than or equal $b |
| <=> | Spaceship | $a <=>$b | Return -1 if $a is less t<br>Return 0 if $a is equ<br>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 |
|---|---|---|---|
| | | ++$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 concate 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 ir |
| !== | 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.

```php
<?php
   //class declaration
   class Developer
   {}
   class Programmer
   {}
   //creating an object of type Developer
   $charu = new Developer();

   //testing the type of object
   if( $charu instanceof Developer)
   {
      echo "Charu is a developer.";
   }
   else
   {
      echo "Charu is a programmer.";
   }
   echo "</br>";
   var_dump($charu instanceof Developer);          //It will return true.
   var_dump($charu instanceof Programmer);       //It will return false.
?>
```

**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 t Here, it will show the directories available in current folder. |

***Note: Note that backticks (``) are not single-quotes.***

---

## 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.

# 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)

```
<?php
// this is C++ style single line comment
```

```
# this is Unix Shell style single line comment
echo "Welcome to PHP single line comments";
?>
```

```
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 of 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

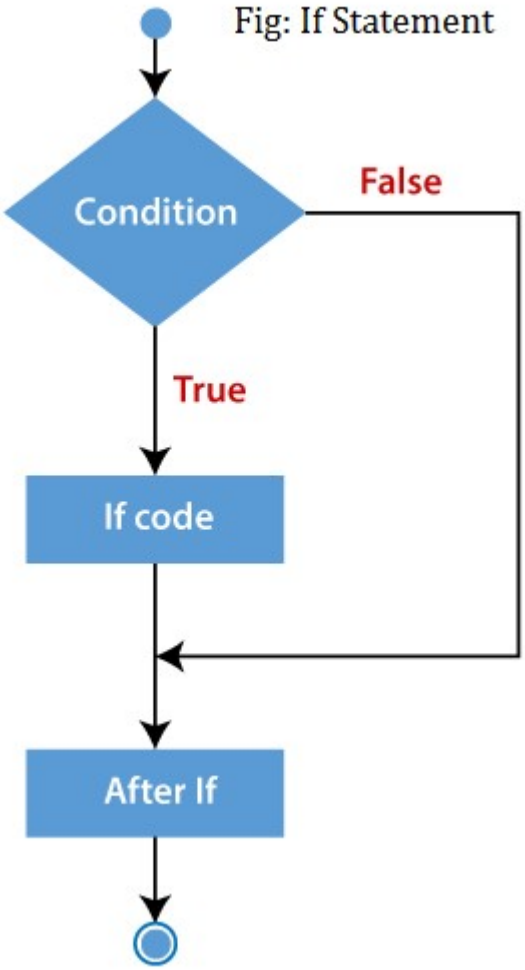
PHP if statement allows conditional execution of code. It is executed if condition is true.

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

**Syntax**

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

**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

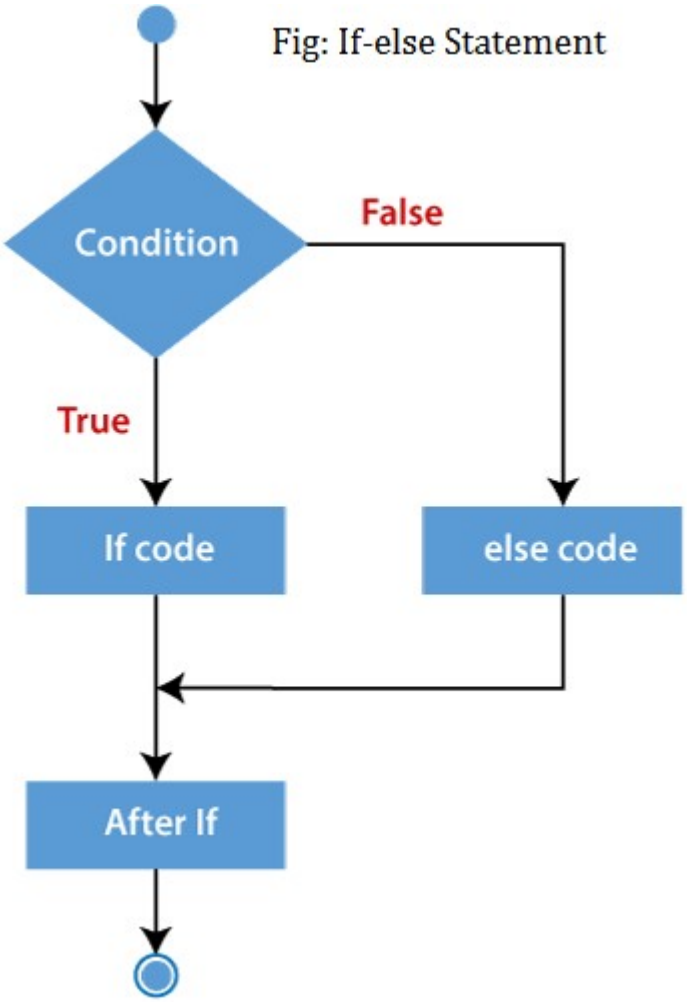
PHP if-else statement is executed whether condition is true or false.

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
```

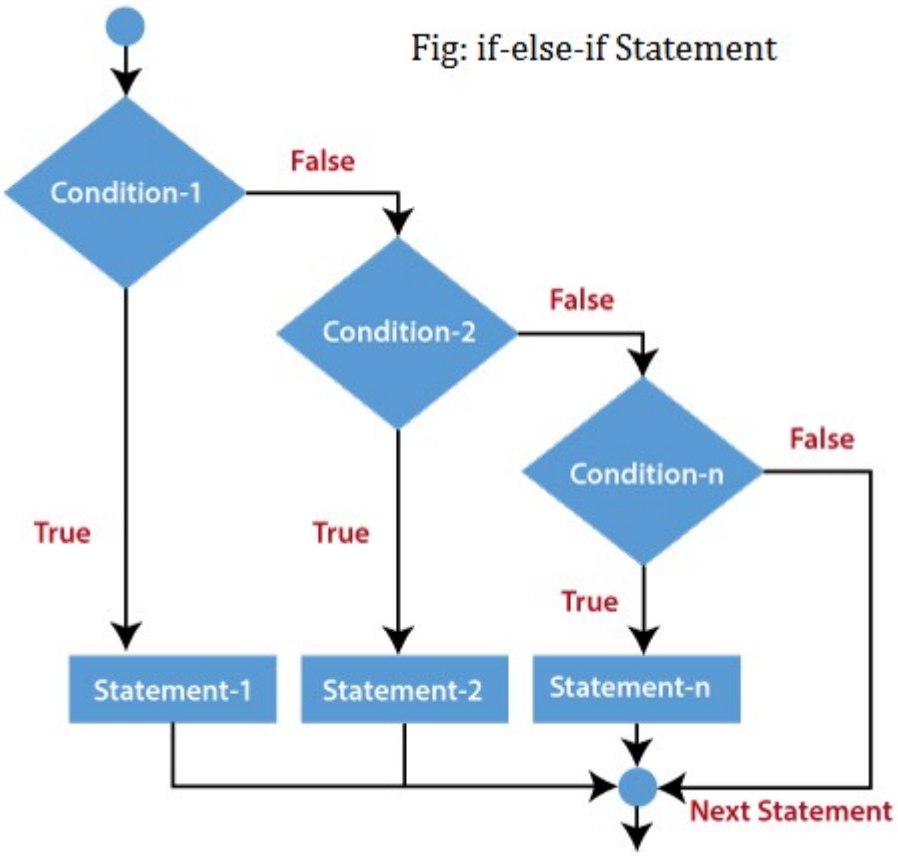
## PHP If-else-if Statement

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**

```
if (condition1){
//code to be executed if condition1 is true
} elseif (condition2){
//code to be executed if condition2 is true
} elseif (condition3){
//code to be executed if condition3 is true
....
}  else{
//code to be executed if all given conditions are false
}
```

**Flowchart**




**Example**

```
<?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

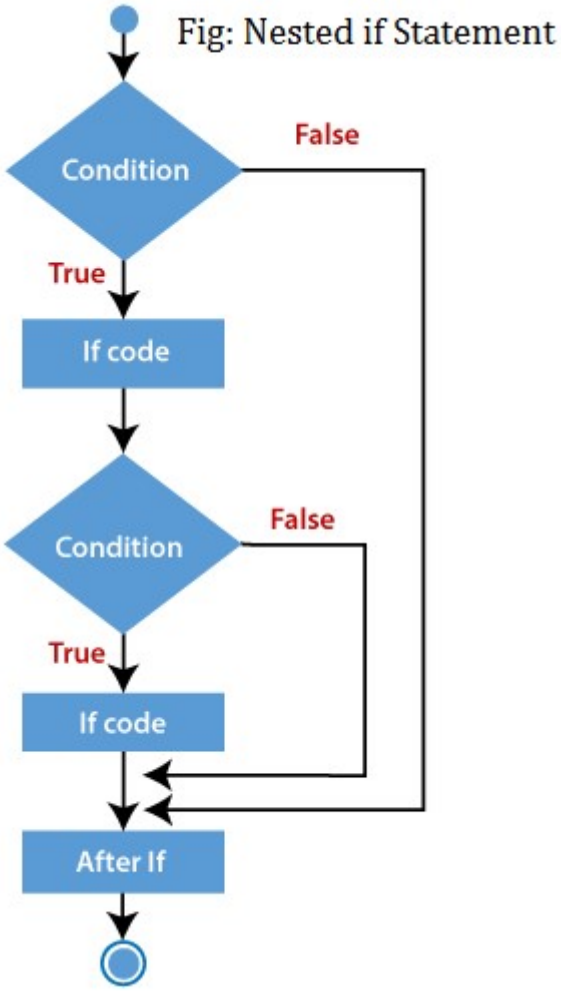
The nested if statement contains the if block inside another if block. The inner if statement executes only when specified condition in outer if statement is **true**.

**Syntax**

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

```
}
}
```

**Flowchart**




**Example**

```
<?php
        $age = 23;
    $nationality = "Indian";
    //applying conditions on nationality and age
    if ($nationality == "Indian")
    {
        if ($age >= 18) {
            echo "Eligible to give vote";
        }
        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.

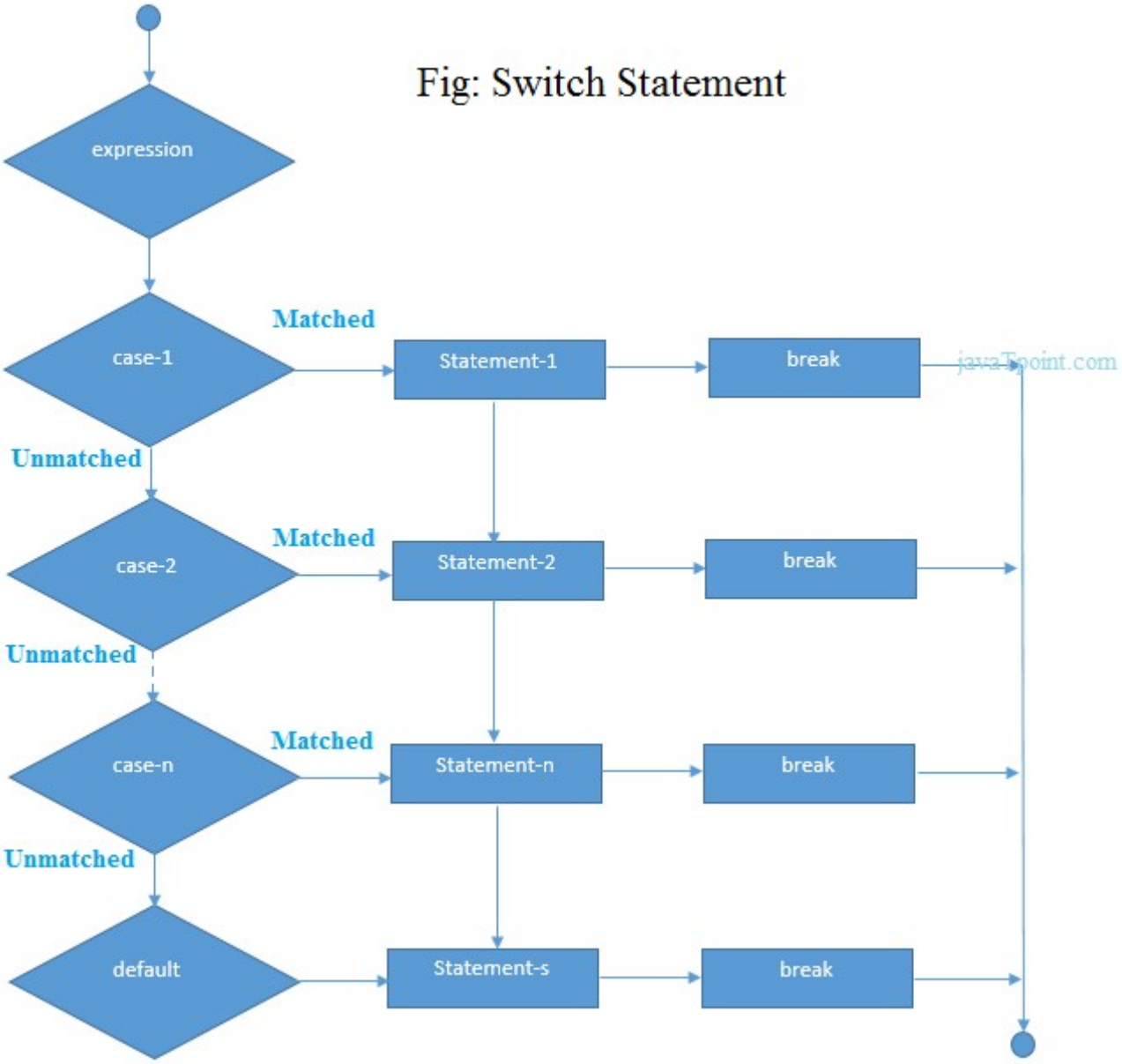
## Syntax

```
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

```
<?php
$num=20;
switch($num){
case 10:
echo("number is equals to 10");
break;
case 20:
echo("number is equal to 20");
break;
```

```
case 30:
echo("number is equal to 30");
break;
default:
echo("number is not equal to 10, 20 or 30");
}
?>
```

**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.

```
<?php
  $ch = "B.Tech";
  switch ($ch)
  {
    case "BCA":
      echo "BCA is 3 years course";
      break;
    case "Bsc":
      echo "Bsc is 3 years course";
      break;
```

```
    case "B.Tech":
        echo "B.Tech is 4 years course";
        break;
    case "B.Arch":
        echo "B.Arch is 5 years course";
        break;
    default:
        echo "Wrong Choice";
        break;
}
?>
```

**Output:**

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

# 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

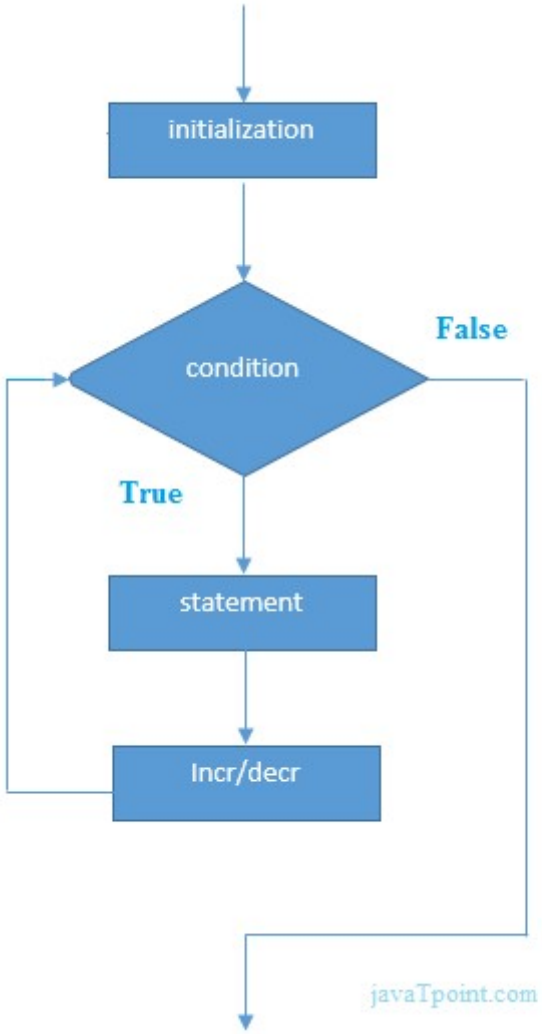
The php for loop is similar to the java/C/C++ for loop. The parameters of for loop have the following meanings:


**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<br/>";
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.

```php
<?php
    $i = 1;
    //infinite loop
    for (;;) {
        echo $i++;
        echo "</br>";
    }
?>
```

**Output:**

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

## Example

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

```php
<?php
    /* example 1 */

    for ($i = 1; $i <= 9; $i++) {
    echo $i;
    }
    echo "</br>";

    /* example 2 */

    for ($i = 1; ; $i++) {
        if ($i > 9) {
            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**

```php
<?php
for($i=1;$i<=3;$i++){
for($j=1;$j<=3;$j++){
echo "$i   $j<br/>";
}
}
?>
```

**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**

```php
foreach( $array as $var ){
 //code to be executed
}
?>
```

**Example**

```php
<?php
$season=array("summer","winter","spring","autumn");
foreach( $season as $arr ){
  echo "Season is: $arr<br />";
}
?>
```

# 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

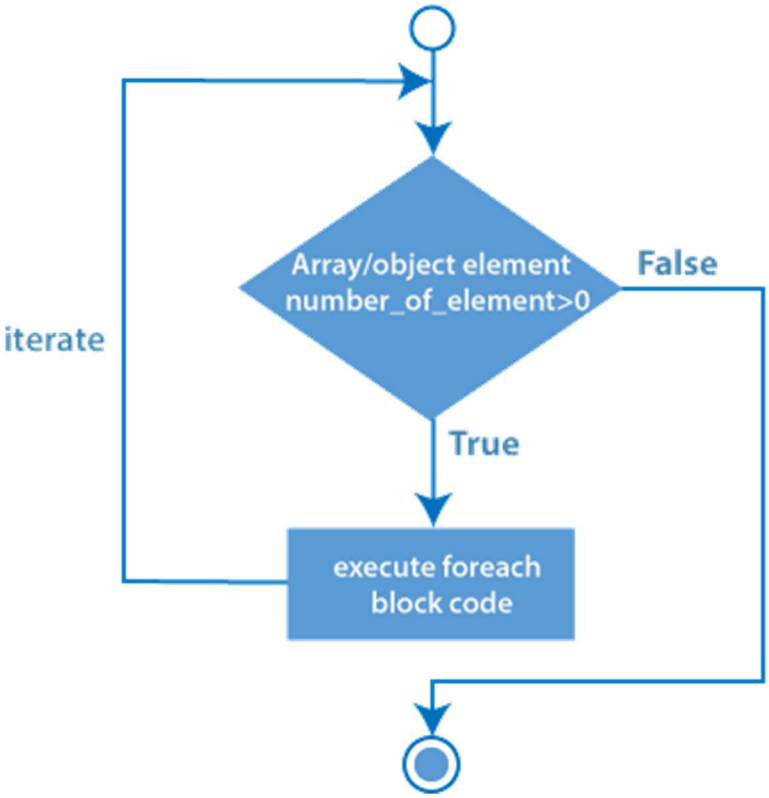
```
1. foreach ($array as $value) {
2.     //code to be executed
3. }
```

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

```
<?php
    //declare multi-dimensional array
    $a = array();
    $a[0][0] = "Alex";
    $a[0][1] = "Bob";
    $a[1][0] = "Camila";
    $a[1][1] = "Denial";

    //display multi-dimensional array elements through foreach loop
    foreach ($a as $e1) {
        foreach ($e1 as $e2) {
            echo "$e2\n";
        }
    }
?>
```

**Output:**

```
Alex Bob Camila Denial
```

## Example 4:

Dynamic array

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

**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.

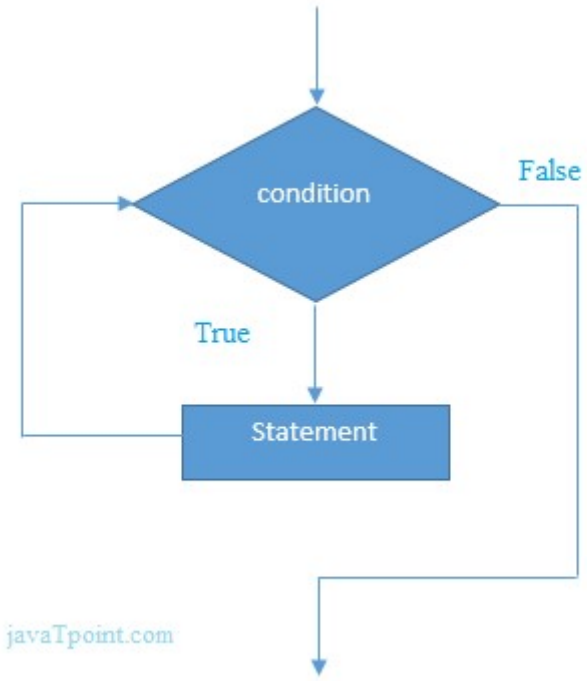
## Syntax

```
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

```
<?php
$n=1;
while($n<=10){
echo "$n<br/>";
$n++;
}
?>
```

```
<?php
$n=1;
while($n<=10):
echo "$n<br/>";
$n++;
endwhile;
?>
```

**Output:**

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

## Example

Below is the example of printing alphabets using while loop.

```
<?php
    $i = 'A';
    while ($i < 'H') {
        echo $i;
        $i++;
        echo "</br>";
    }
?>
```

**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 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.

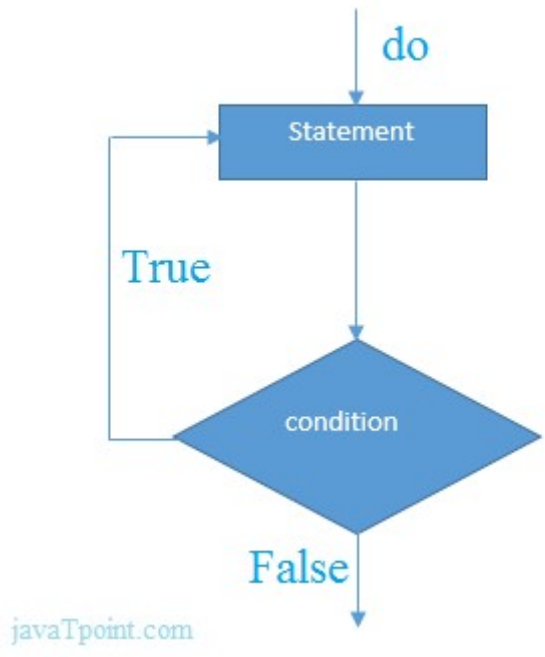
It executes the code at least one time always because the condition is checked after executing the code.

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<br/>";
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 th false. |
| Condition checks first, and then block of statements executes. | Block of statements executes first and then condition c |
| 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

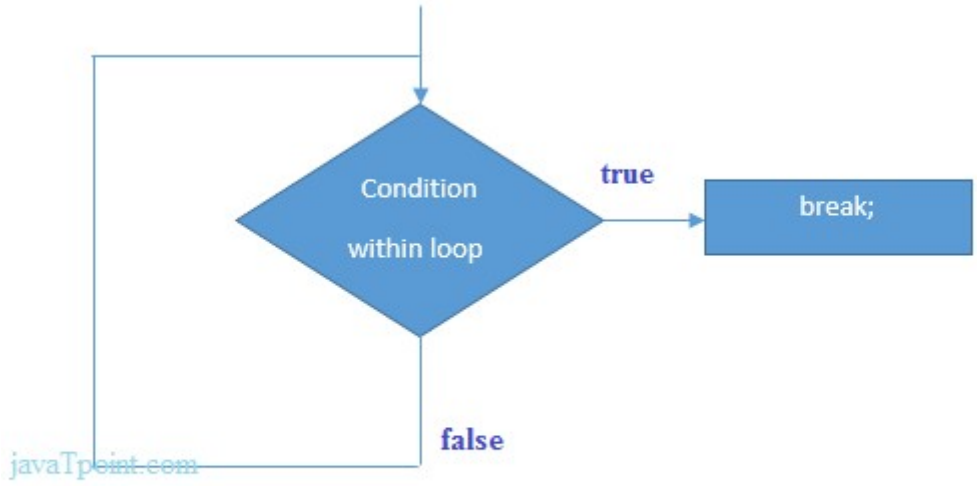



**Figure: Flowchart of break statement**

## 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.

```
<?php
for($i=1;$i<=3;$i++){
 for($j=1;$j<=3;$j++){
  echo "$i   $j<br/>";
  if($i==2 && $j==2){
   break;
  }
 }
}
?>
```

**Output:**

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

## PHP Break: inside switch statement

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

```
<?php
$num=200;
switch($num){
case 100:
echo("number is equals to 100");
break;
case 200:
echo("number is equal to 200");
break;
case 50:
echo("number is equal to 300");
break;
default:
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.

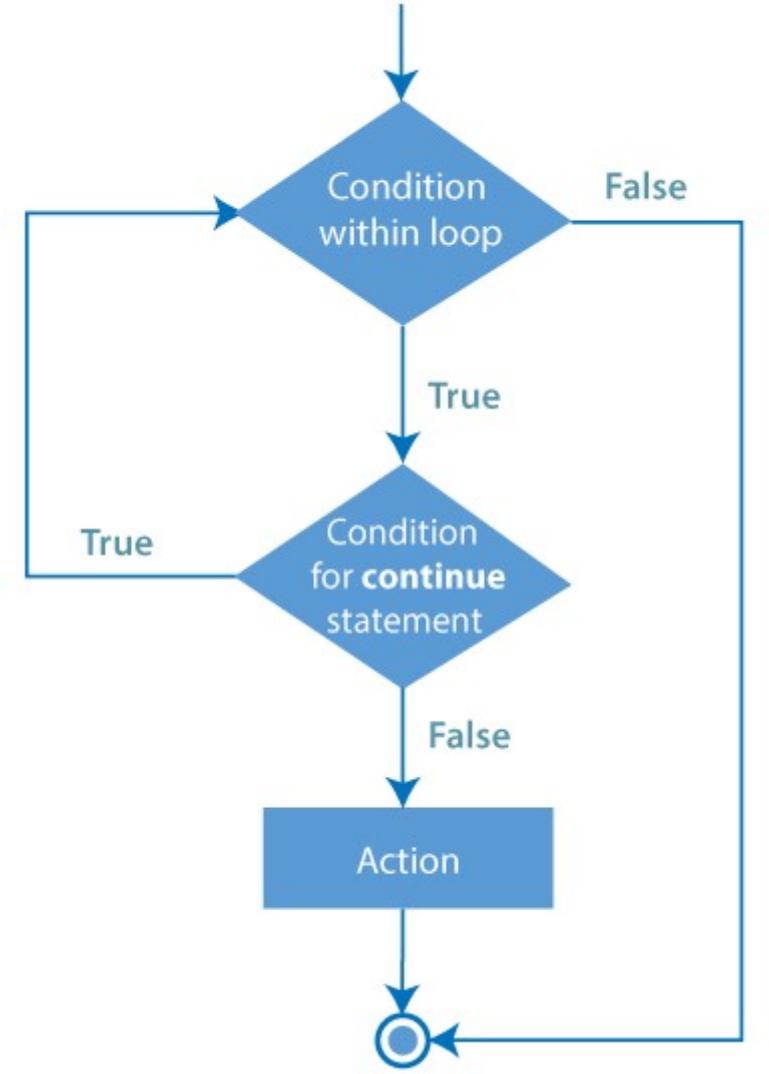
The continue statement can be used with all types of loops such as - for, while, do-while, and foreach loop. The continue statement allows the user to skip the execution of the code for the specified condition.

## 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.

```php
<?php
    //outer loop
    for ($i =1; $i<=3; $i++) {
        //inner loop
        for ($j=1; $j<=3; $j++) {
            if (!($i == $j) ) {
                continue;       //skip when i and j does not have same values
            }
            echo $i.$j;
            echo "</br>";
        }
    }
?>
```

**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.

```php
<?php
    //php program to demonstrate the use of continue statement

    echo "Even numbers between 1 to 20: </br>";
    $i = 1;
    while ($i<=20) {
        if ($i %2 == 1) {
            $i++;
            continue;   //here it will skip rest of statements
        }
        echo $i;
        echo "</br>";
        $i++;
    }
```

15. ?>

**Output:**

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

# 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

```
function functionname(){
//code to be executed
}
```

## PHP Functions Example

*File: function1.php*

```
<?php
function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

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*

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
sayHello("Sonoo");
sayHello("Vimal");
sayHello("John");
?>
```

Output:

```
Hello Sonoo
Hello Vimal
```

```
Hello John
```

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

*File: functionarg2.php*

```
<?php
function sayHello($name,$age){
echo "Hello $name, you are $age years old<br/>";
}
sayHello("Sonoo",27);
sayHello("Vimal",29);
sayHello("John",23);
?>
```

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*

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

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*

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

Output:

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

## PHP Function: Returning Value

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

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

Output:

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

# PHP Parameterized Function

PHP Parameterized functions are the functions with parameters. You can pass any number of parameters inside a function. These passed parameters act as variables inside your function.

They are specified inside the parentheses, after the function name.

The output depends upon the dynamic values passed as the parameters into the function.

---

## PHP Parameterized Example 1

**Addition and Subtraction**

In this example, we have passed two parameters **$x** and **$y** inside two functions **add()** and **sub()**.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>Parameter Addition and Subtraction Example</title>
5. </head>
6. <body>
7. <?php
8.         //Adding two numbers
9.         function add($x, $y) {
10.           $sum = $x + $y;
11.           echo "Sum of two numbers is = $sum <br><br>";
12.        }
13.        add(467, 943);
14.
15.        //Subtracting two numbers
16.        function sub($x, $y) {
17.           $diff = $x - $y;
18.           echo "Difference between two numbers is = $diff";
19.        }
20.        sub(943, 467);
21.     ?>
22. </body>
```

23. </html>

# 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
```

*File: array2.php*

```
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"]."<br/>";
4. echo "John salary: ".$salary["John"]."<br/>";
5. echo "Kartik salary: ".$salary["Kartik"]."<br/>";
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"]."<br/>";
6. echo "John salary: ".$salary["John"]."<br/>";
7. echo "Kartik salary: ".$salary["Kartik"]."<br/>";
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:

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

2nd way:

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

## PHP Indexed Array Example

*File: array1.php*

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

Output:

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

*File: array2.php*

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

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*

```
<?php
$size=array("Big","Medium","Short");
foreach( $size as $s )
{
  echo "Size is: $s<br />";
}
?>
```

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.

```
<?php
$size=array("Big","Medium","Short");
echo count($size);
?>
```

Output:

```
3
```

# 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:

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

2nd way:

```
$salary["Sonoo"]="550000";
$salary["Vimal"]="250000";
$salary["Ratan"]="200000";
```

## Example

*File: arrayassociative1.php*

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "Vimal salary: ".$salary["Vimal"]."<br/>";
echo "Ratan salary: ".$salary["Ratan"]."<br/>";
?>
```

Output:

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

*File: arrayassociative2.php*

```
<?php
$salary["Sonoo"]="550000";
$salary["Vimal"]="250000";
$salary["Ratan"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "Vimal salary: ".$salary["Vimal"]."<br/>";
echo "Ratan salary: ".$salary["Ratan"]."<br/>";
?>
```

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.

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
foreach($salary as $k => $v) {
echo "Key: ".$k." Value: ".$v."<br/>";
}
?>
```

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

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

## 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*

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );

for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]."  ";
  }
  echo "<br/>";
}
?>
```

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**

```
array array ([ mixed $... ] )
```

**Example**

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

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**

```
array array_change_key_case ( array $array [, int $case = CASE_LOWER ] )
```

**Example**

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

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

**Example**

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_LOWER));
?>
```

Output:

## 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**

```
array array_chunk ( array $array , int $size [, bool $preserve_keys = false ] )
```

**Example**

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_chunk($salary,2));
?>
```

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**

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

**Example**

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

## 5) PHP sort() function

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

**Syntax**

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

**Example**

```
<?php
$season=array("summer","winter","spring","autumn");
sort($season);
foreach( $season as $s )
{
  echo "$s<br />";
}
?>
```

Output:

```
autumn
spring
summer
winter
```

## 6) PHP array_reverse() function

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

**Syntax**

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

**Example**

```
<?php
$season=array("summer","winter","spring","autumn");
$reverseseason=array_reverse($season);
foreach( $reverseseason as $s )
{
  echo "$s<br />";
}
?>
```

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**

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

**Example**

```
<?php
$season=array("summer","winter","spring","autumn");
$key=array_search("spring",$season);
echo $key;
?>
```

Output:

```
2
```

## 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**

```
array array_intersect ( array $array1 , array $array2 [, array $... ] )
```

**Example**

```
<?php
$name1=array("sonoo","john","vivek","smith");
$name2=array("umesh","sonoo","kartik","smith");
$name3=array_intersect($name1,$name2);
foreach( $name3 as $n )
{
```

```
7.    echo "$n<br />";
8.  }
9.  ?>
```

Output:

```
sonoo
smith
```

# PHP Program to find the factorial of a number

```
<h1>
<?php
if(isset($_GET['submit']))
{
   $n=$_GET['num'];
   $fact=1;
   for($i=1;$i<=$n;$i++)
   {
      $fact=$fact*$i;

   }
   echo "Factorial of $n is $fact";
}

?>
</h1>
<form action="fact.php" method="get">
   <input type="text" name="num">
   <input type="submit" name="submit" value="submit">
```

```
</form>
```

**PHP Program to check for perfect no.**

```
<h1>
<?php
if(isset($_GET['submit']))
{
   $n=$_GET['num'];
   $sum=0;
   for($i=1;$i<$n;$i++)
   {
      if($n%$i==0)
      {
         $sum=$sum+$i;
      }

   }
   if($sum == $n)
   echo "Perfect No.";
   else
   echo "Not a Perfect No.";
}

?>
</h1>
<form action="first.php" method="get">
```

```
    <input type="text" name="num">
    <input type="submit" name="submit" value="submit">
</form>
```