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

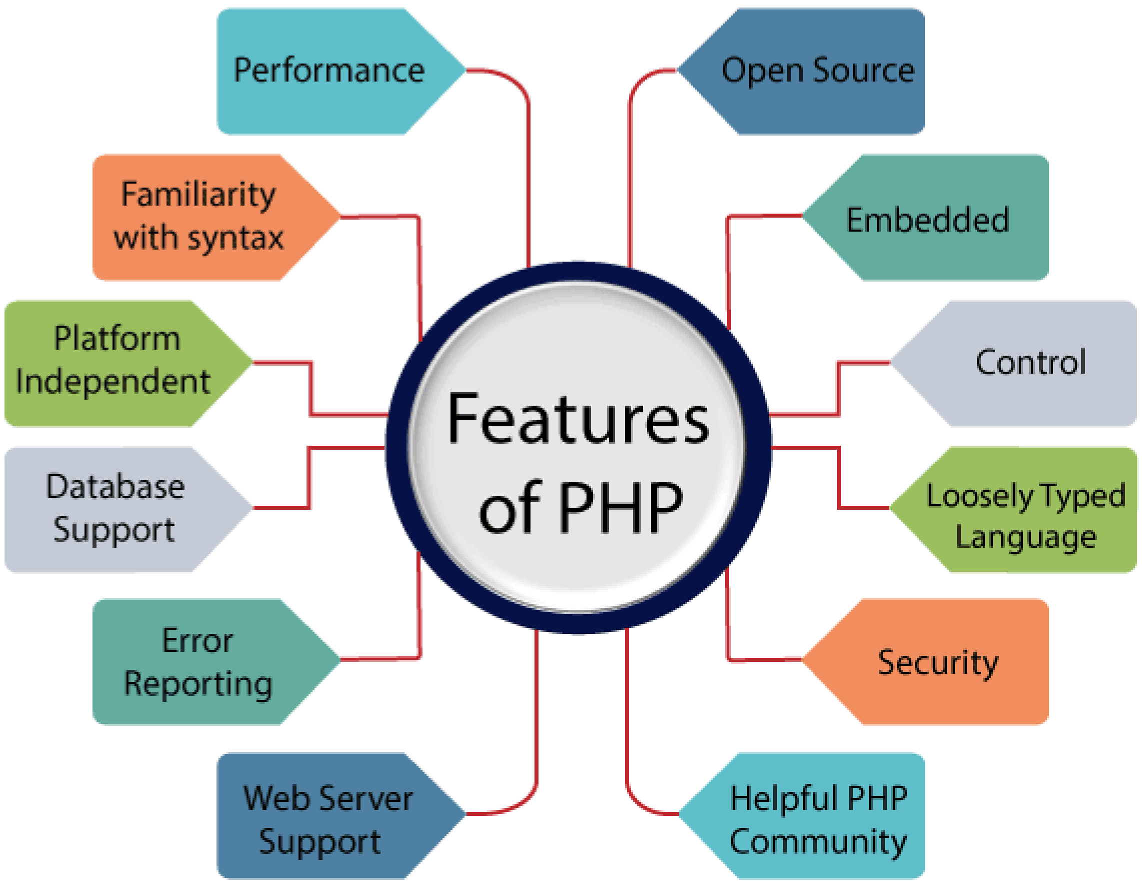
PHP is a server-side scripting language, which is used to design the dynamic web applications with MySQL database.

- 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.

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.

Note: PHP statements ends with semicolon (;).

All PHP code goes between the php tag. It starts with <?php and ends with ?>. The syntax of PHP tag is given below:

```
<?php
//your code here
?>
```

Let's see a simple PHP example where we are writing some text using PHP echo command.

```
<!DOCTYPE>
<html>
<body>
<?php
echo "<h2>Hello First PHP</h2>";
?>
</body>
</html>
```

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.

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:

```
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: printing multi line string

```
<?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: printing escaping characters

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

```
?>
```
Output:

Hello escape "sequence" characters

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

Message is: Hello JavaTpoint PHP

PHP Variables

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:

```
$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 g
enerate 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
```

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

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:
```

Using $GLOBALS instead of global

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

Example:

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

Output:

Sum of global variables is: 18

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:

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

Output:

Value of x: 7

Note: local variable has higher priority than the global variable.

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 $ and $$ Variables

The $var (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

The $$var (double dollar) is a reference variable that stores the value of the $variable inside it.

To understand the difference better, let's see some examples.

Example 1
```
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
?>
```
In the above example, we have assigned a value to the variable x as abc. Value of reference variable $$x is assigned as 200.

Now we have printed the values $x, $$x and a string.

Example
```
<?php
$name="Cat";
${$name}="Dog";
${${$name}}="Monkey";
echo $name. "<br>";
echo ${$name}. "<br>";
echo $Cat. "<br>";
echo ${${$name}}. "<br>";
echo $Dog. "<br>";
?>
```
In the above example, we have assigned a value to the variable name Cat. Value of reference variable *${$name}* is assigned as Dog and *${${$name}}* as Monkey. Now we have printed the values as *$name, ${$name}, $Cat, ${${$name}}* and *$Dog*.

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.

Note: Unlike variables, constants are automatically global throughout the script.

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.

```
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().

```
<?php
define("MESSAGE","Hello JavaTpoint PHP");
echo MESSAGE;
?>
```

Output:

Hello JavaTpoint PHP

Create a constant with case-insensitive name:

```
<?php
```

```
define("MESSAGE","Hello   JavaTpoint   PHP",true);//not   case
sensitive
echo MESSAGE, "</br>";
echo message;
?>
```

Output:

```
Hello JavaTpoint PHP

Hello JavaTpoint PHP
```

```
<?php
define("MESSAGE","Hello JavaTpoint PHP",false);//case sensitive
echo MESSAGE;
echo message;
?>
```

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.

```
<?php
const MESSAGE="Hello const by JavaTpoint PHP";
echo MESSAGE;
?>
```

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:

```
constant (name)
<?php
    define("MSG", "JavaTpoint");
    echo MSG, "</br>";
    echo constant("MSG");
    //both are similar
?>
```

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:

```
<?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:

1. An integer can be either positive or negative.
2. An integer must not contain decimal point.
3. Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
4. 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
    $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
    $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
    $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.

Example:

```
<?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
     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.

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:

```
$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 operand.
+=	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, otherwise 0.
|	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 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
==	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 data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data 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 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.

```
<?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 the result. 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.

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 and types	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)

```
<?php
// this is C++ style single line comment
# this is Unix Shell style single line comment
echo "Welcome to PHP single line comments";
?>
```

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 of PHP multiple line comment.

```
<?php
/*
Anything placed
within comment
will not be displayed
on the browser;
*/
echo "Welcome to PHP multi line comment";
?>
```

Output:

Welcome to PHP multi line comment

Control Statement

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

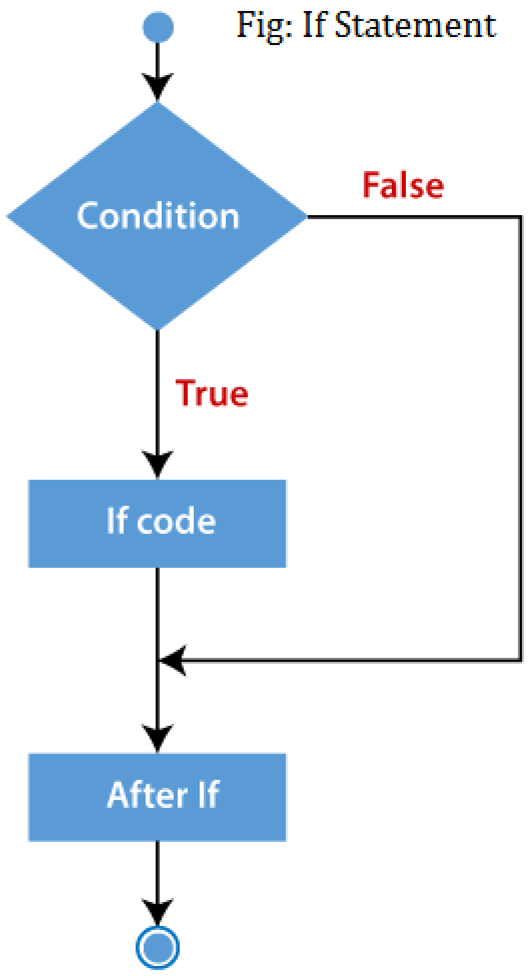
PHP if statement allows conditional execution of code. It is executed if condition is true.

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){
//code to be executed
}
```

Flowchart



Example

```
<?php
$num=12;
if($num<100){
echo "$num is less than 100";
}
?>
```

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

```
if(condition){
//code to be executed if true
}else{
//code to be executed if false
}
```

Flowchart

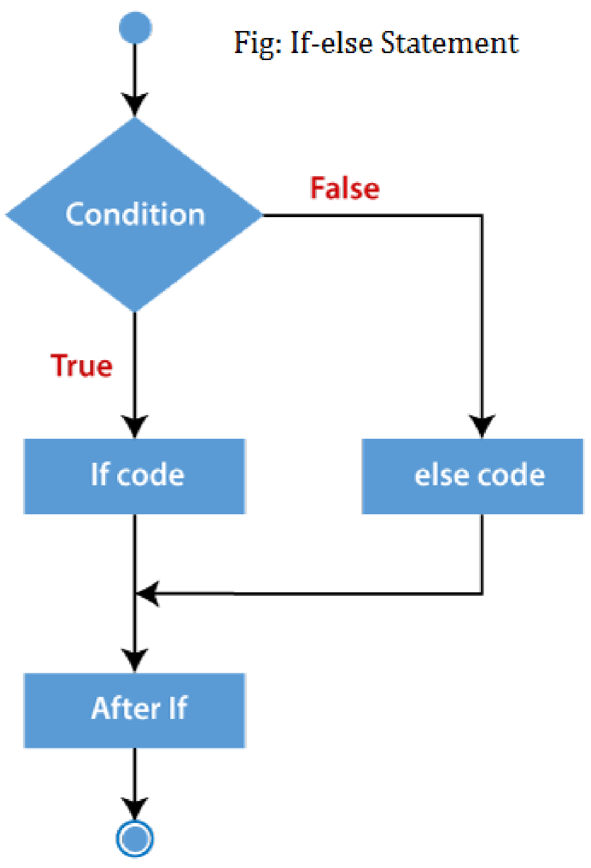


Fig: If-else Statement

Example

```
<?php
$num=12;
if($num%2==0){
echo "$num is even number";
}else{
echo "$num is odd number";
}
?>
```

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

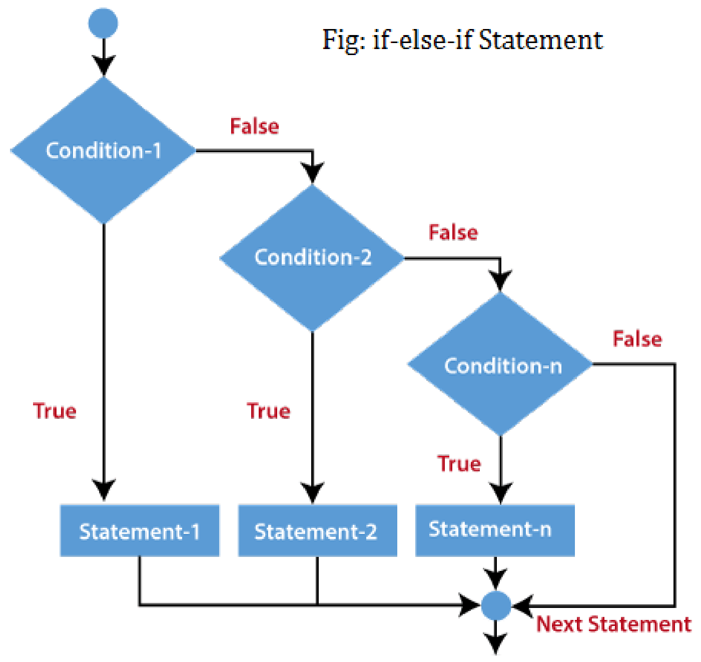


Fig: if-else-if Statement

Example

```
<?php
    $marks=69;
    if ($marks<33){
        echo "fail";
    }
    else if ($marks>=34 && $marks<50) {
        echo "D grade";
    }
    else if ($marks>=50 && $marks<65) {
       echo "C grade";
    }
    else if ($marks>=65 && $marks<80) {
        echo "B grade";
    }
    else if ($marks>=80 && $marks<90) {
        echo "A grade";
    }
    else if ($marks>=90 && $marks<100) {
        echo "A+ grade";
    }
   else {
        echo "Invalid input";
    }
?>
```

Output:

B Grade

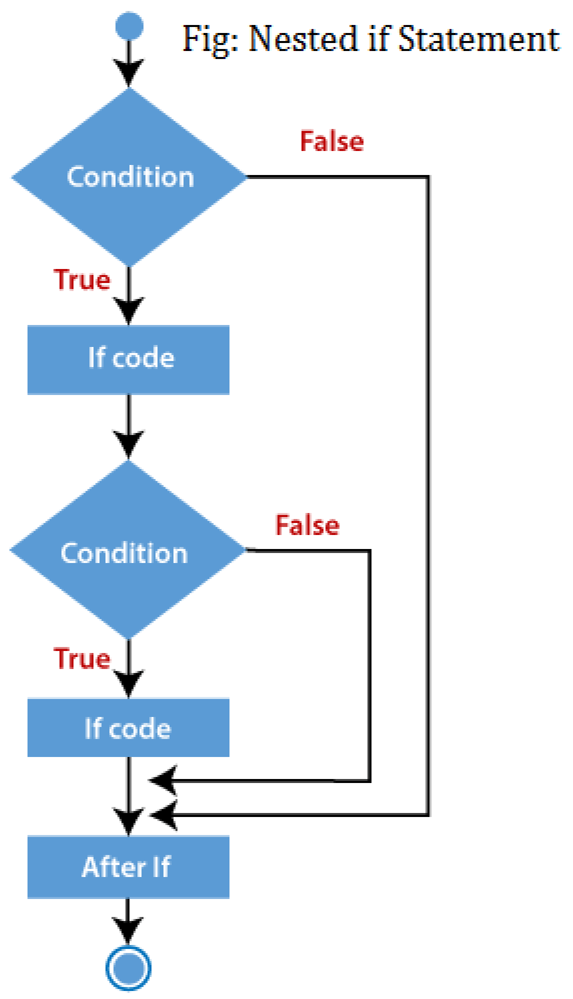
PHP nested if Statement

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

```
if (condition) {
//code to be executed if condition is true
if (condition) {
//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 {
            echo "Not eligible to give vote";
        }
    }
?>
```

Output:

Eligible to give vote

PHP Switch Example

```
<?php
              $a = 34; $b = 56; $c = 45;
    if ($a < $b) {
        if ($a < $c) {
            echo "$a is smaller than $b and $c";
        }
    }
?>
```

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.

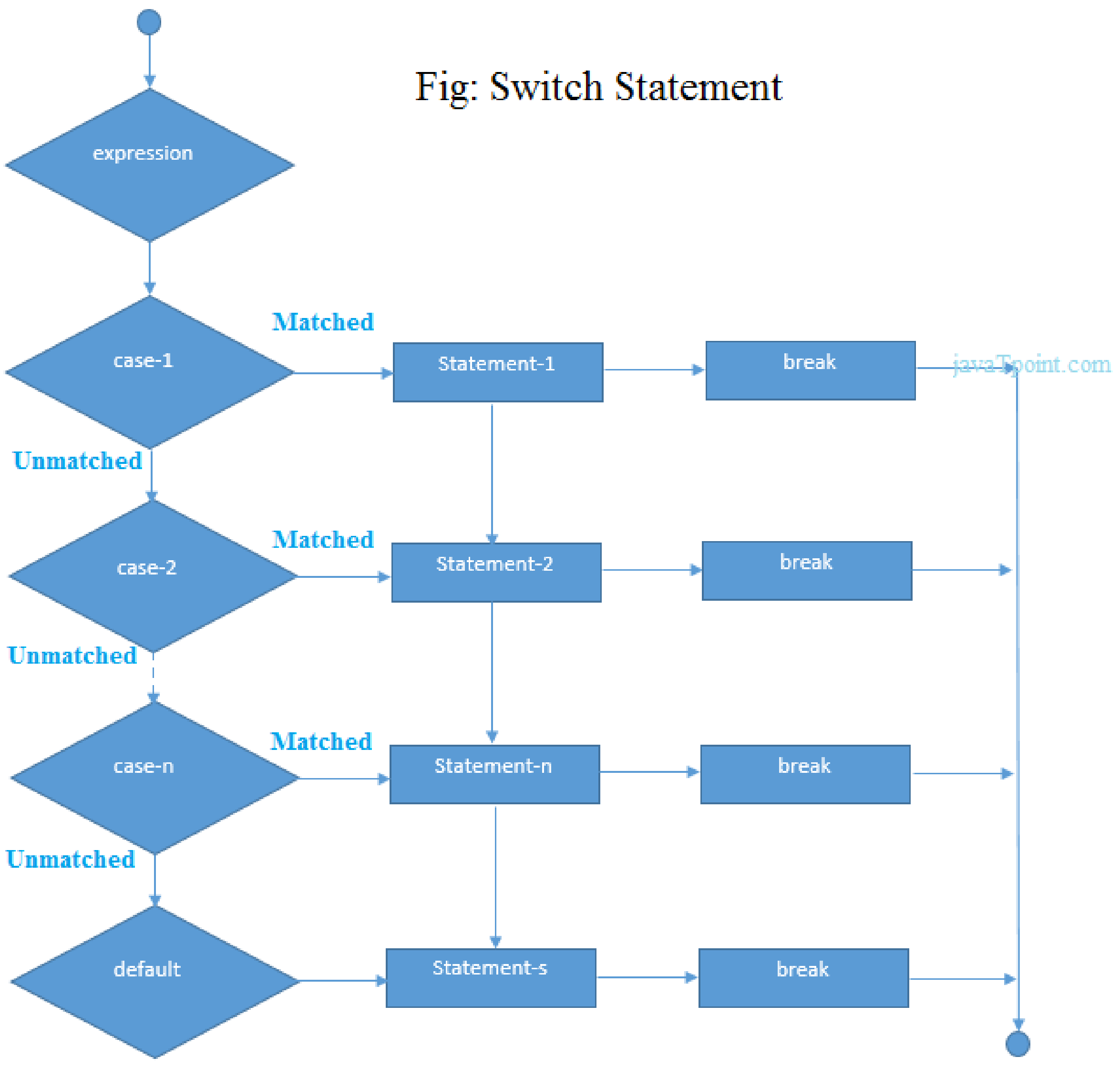
Syntax

```
switch(expression){
case value1:
 //code to be executed
 break;
case value2:
 //code to be executed
 break;
......
default:
 code to be executed if all cases are not matched;
}
```

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.

```
<?php
    $ch = 'U';
    switch ($ch)
    {
        case 'a':
            echo "Given character is vowel";
            break;
        case 'e':
            echo "Given character is vowel";
            break;
        case 'i':
            echo "Given character is vowel";
            break;
        case 'o':
            echo "Given character is vowel";
            break;
        case 'u':
            echo "Given character is vowel";
            break;
        case 'A':
            echo "Given character is vowel";
            break;
        case 'E':
            echo "Given character is vowel";
            break;
        case 'I':
            echo "Given character is vowel";
            break;
        case 'O':
            echo "Given character is vowel";
            break;
        case 'U':
            echo "Given character is vowel";
            break;
        default:
            echo "Given character is consonant";
            break;
    }
?>
```

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 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.

```
<?php
    $ch = 'c';
    switch ($ch)
    {
        case 'a':
            echo "Choice a";
            break;
        case 'b':
            echo "Choice b";
            break;
        case 'c':
            echo "Choice c";
            echo "</br>";
        case 'd':
            echo "Choice d";
            echo "</br>";
        default:
            echo "case a, b, c, and d is not found";
    }
?>
```

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.

```
<?php
    $car = "Hyundai";
        $model = "Tucson";
        switch( $car )
        {
            case "Honda":
                switch( $model )
                {
                    case "Amaze":
                           echo "Honda Amaze price is 5.93 -
9.79 Lakh.";
                        break;
                    case "City":
                           echo "Honda City price is 9.91 -
14.31 Lakh.";
                        break;
                }
                break;
            case "Renault":
                switch( $model )
                {
                    case "Duster":
                        echo "Renault Duster price is 9.15 -
14.83 L.";
                        break;
                    case "Kwid":
                           echo "Renault Kwid price is 3.15 -
5.44 L.";
                        break;
                }
                break;
            case "Hyundai":
                switch( $model )
                {
                    case "Creta":
                        echo "Hyundai Creta price is 11.42 -
18.73 L.";
                        break;
        case "Tucson":
```

```
                                    echo "Hyundai Tucson price is 22.39
- 32.07 L.";
                                break;
                            case "Xcent":
                                    echo "Hyundai Xcent price is 6.5 -
10.05 L.";
                                break;
                        }
                        break;
                }
?>
```

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

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

Parameters

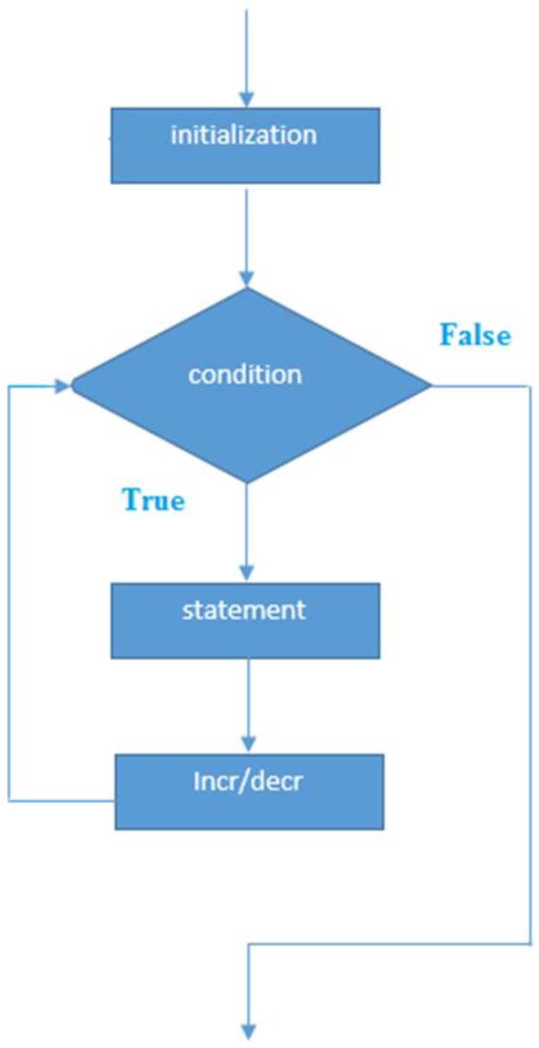
The php for loop is similar to the java/C/C++ for loop. The parameters of for loop have the following meanings:

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

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

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
    $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
    /* example 1 */

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

    /* example 2 */

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

    /* example 3 */

    $i = 1;
    for (; ; ) {
        if ($i > 9) {
            break;
        }
        echo $i;
        $i++;
    }
    echo "</br>";
    /* example 4 */

    for ($i = 1, $j = 0; $i <= 9; $j += $i, print $i, $i++);
?>
```

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
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

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

Example

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

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

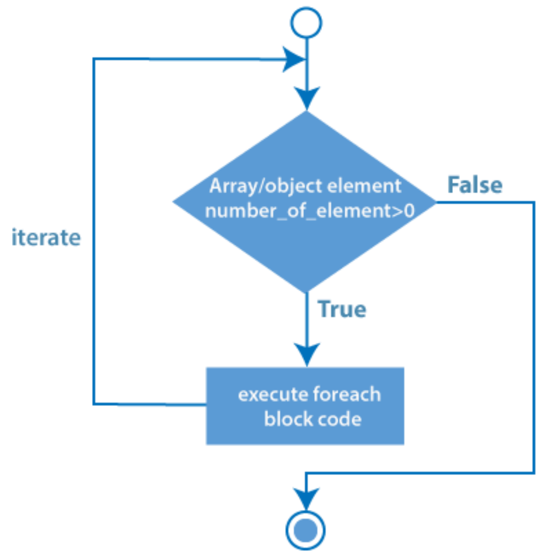
```
foreach ($array as $value) {
    //code to be executed
}
```

There is one more syntax of foreach loop.

Syntax

```
foreach ($array as $key => $element) {
    //code to be executed
}
```

Flowchart


Array/object element number_of_element>0
False
iterate
True
execute foreach block code

Example 1:

PHP program to print array elements using foreach loop.

```
<?php
    //declare array
    $season = array ("Summer", "Winter", "Autumn", "Rainy");

    //access array elements using foreach loop
    foreach ($season as $element) {
        echo "$element";
        echo "</br>";
    }
?>
```

Output:

```
Summer

Winter

Autumn

Rainy
```

Example 2:

PHP program to print associative array elements using foreach loop.

```
<?php
    //declare array
    $employee = array (
        "Name" => "Alex",
        "Email" => "alex_jtp@gmail.com",
        "Age" => 21,
        "Gender" => "Male"
    );

    //display associative array element through foreach loop
    foreach ($employee as $key => $element) {
        echo $key . " : " . $element;
        echo "</br>";
    }
?>
```

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.

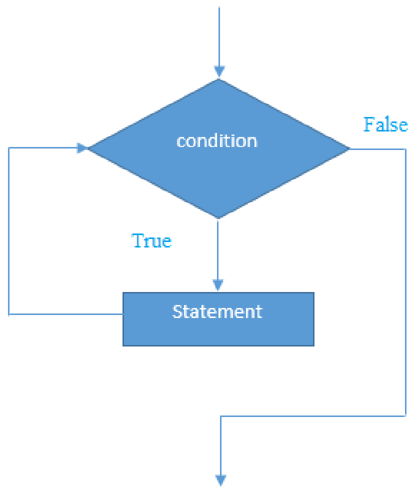
Syntax

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

Alternative Syntax

```
while(condition):
//code to be executed

endwhile;
```

PHP While Loop Flowchart



PHP While Loop Example

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

Output:

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

Alternative Example

```
<?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

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

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

```
while(true) {
//code to be executed
}
Example

<?php
    while (true) {
        echo "Hello, Good Day!";
        echo "</br>";
    }
?>
```

Output:

```
Hello, Good Day!
Hello, Good Day!
Hello, Good Day!
.
.
.
.
.
Hello, Good Day!
Hello, Good Day!
```

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.

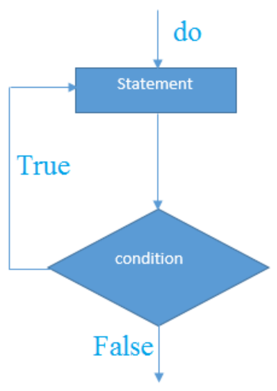
It executes the code at least one time always because the condition is checked after executing the code.

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

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

Flowchart



Example

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

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.

```
 <?php
    $x = 5;
    do {
        echo "Welcome to javatpoint! </br>";
        $x++;
    } while ($x < 10);
?>
```

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.

```
 <?php
    $x = 1;
    do {
        echo "1 is not greater than 10.";
        echo "</br>";
        $x++;
    } while ($x > 10);
    echo $x;
?>
```

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

```
jump statement;
break;
```

Flowchart

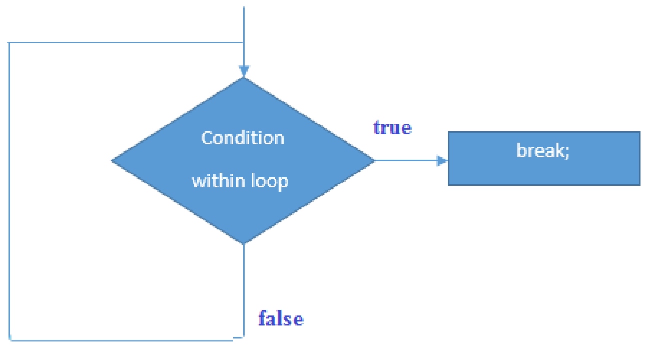


Figure: Flowchart of break statement

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.

```
<?php
for($i=1;$i<=10;$i++){
```

```
echo "$i <br/>";
if($i==5){
break;
}
}
?>
```

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");
}
?>
```

Output:

```
number is equal to 200
```

PHP Break: with array of string

```
<?php
//declare an array of string
$number = array ("One", "Two", "Three", "Stop", "Four");
foreach ($number as $element) {
if ($element == "Stop") {
break;
}
echo "$element </br>";
}
?>
```
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.

```
<?php
$car = 'Mercedes Benz';
switch ($car) {
default:
echo '$car is not Mercedes Benz<br>';
case 'Orange':
echo '$car is Mercedes Benz';
}
?>
```
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.

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

Output:

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

Note: The break keyword immediately ends the execution of the current structure.

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.

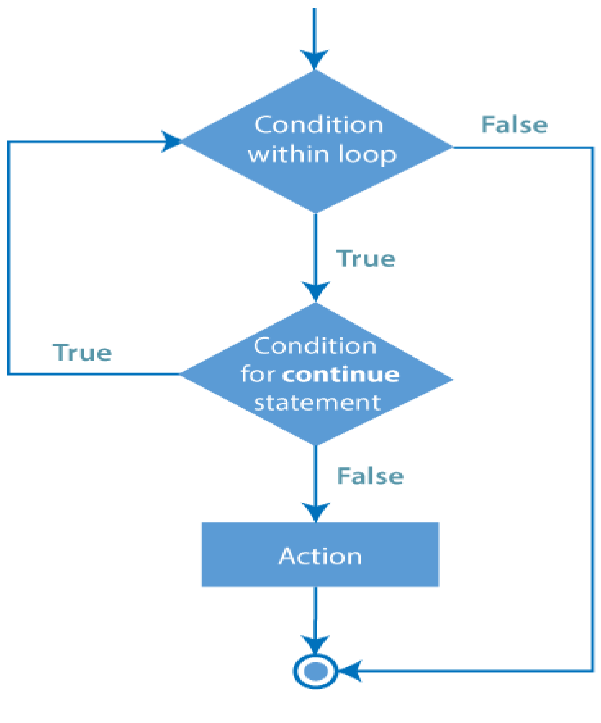
The continue statement can be used with all types of loops such as - for, while, do-while, and foreach loop. The continue statement allows the user to skip the execution of the code for the specified condition.

Syntax

The syntax for the continue statement is given below:

```
jump-statement;
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
    //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 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++;
    }
?>
```
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.

```
<?php
    $number = array ("One", "Two", "Three", "Stop", "Four");
    foreach ($number as $element) {
        if ($element == "Stop") {
            continue;
        }
        echo "$element </br>";
    }
?>
```

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:

```
<?php
    //outer loop
    for ($i =1; $i<=3; $i++) {
        //inner loop
        for ($j=1; $j<=3; $j++) {
            if (($i == $j) ) {       //skip when i and j have
same values
                continue 1;     //exit only from inner for loop
            }
            echo $i.$j;
            echo "</br>";
        }
    }
?>
```

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

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

Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.

PHP Functions Example

```
<?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.

```
<?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.

```
<?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.

```
<?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.

```
<?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.

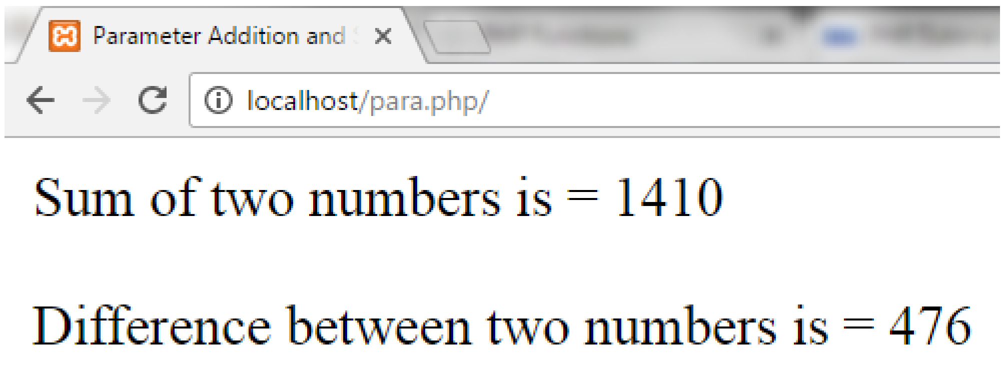
PHP Parameterized Example 1

Addition and Subtraction

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

```
<!DOCTYPE html>
<html>
<head>
    <title>Parameter Addition and Subtraction Example</title>
</head>
<body>
<?php
        //Adding two numbers
         function add($x, $y) {
            $sum = $x + $y;
            echo "Sum of two numbers is = $sum <br><br>";
         }
         add(467, 943);

         //Subtracting two numbers
         function sub($x, $y) {
            $diff = $x - $y;
            echo "Difference between two numbers is = $diff";
         }
         sub(943, 467);
      ?>
</body>
</html>
```

Output:



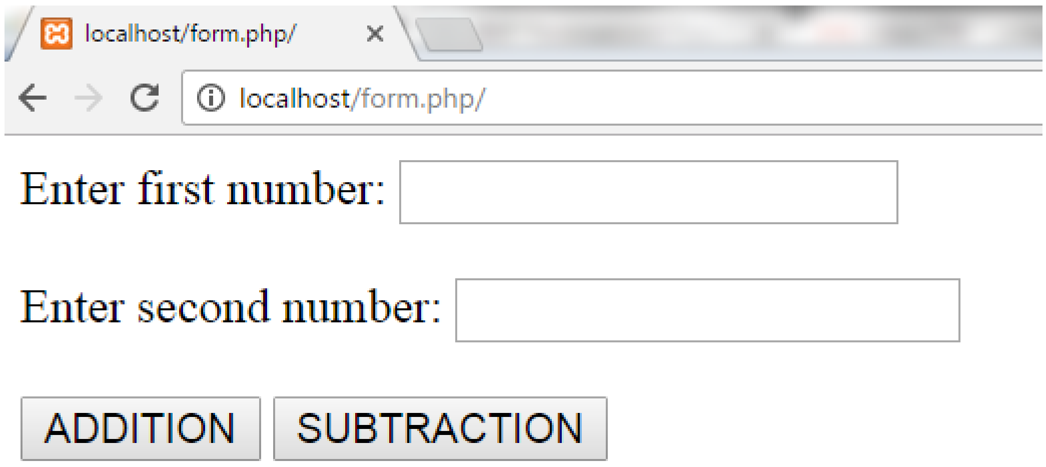
```
Sum of two numbers is = 1410

Difference between two numbers is = 476
```

PHP Parameterized Example 2
Addition and Subtraction with Dynamic number

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

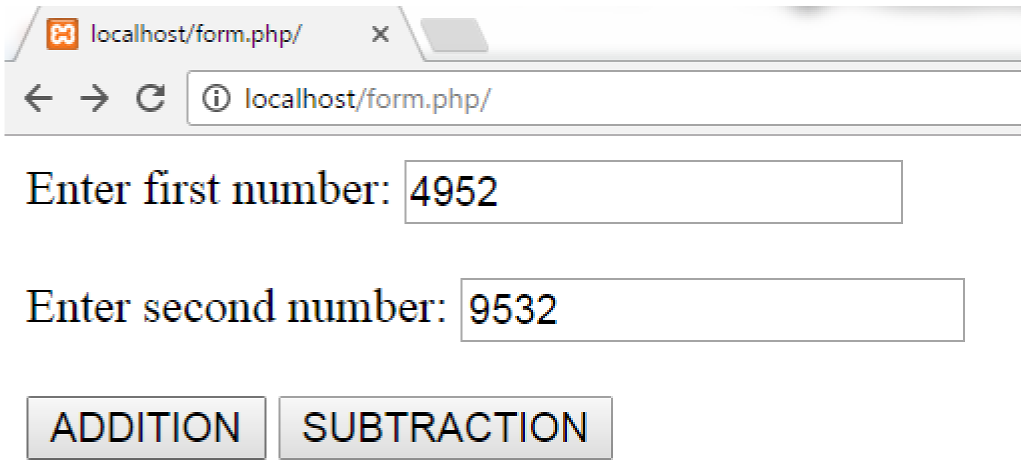
```
<?php
//add() function with two parameter
function add($x,$y)
{
$sum=$x+$y;
echo "Sum = $sum <br><br>";
}
//sub() function with two parameter
function sub($x,$y)
{
$sub=$x-$y;
echo "Diff = $sub <br><br>";
}
//call function, get  two argument through input box and click
on add or sub button
if(isset($_POST['add']))
{
//call add() function
 add($_POST['first'],$_POST['second']);
}
if(isset($_POST['sub']))
{
//call add() function
sub($_POST['first'],$_POST['second']);
}
?>
<form method="post">
Enter first number: <input type="number" name="first"/><br><br>
Enter        second       number:       <input       type="number"
name="second"/><br><br>
<input type="submit" name="add" value="ADDITION"/>
<input type="submit" name="sub" value="SUBTRACTION"/>
</form>
```

Output:



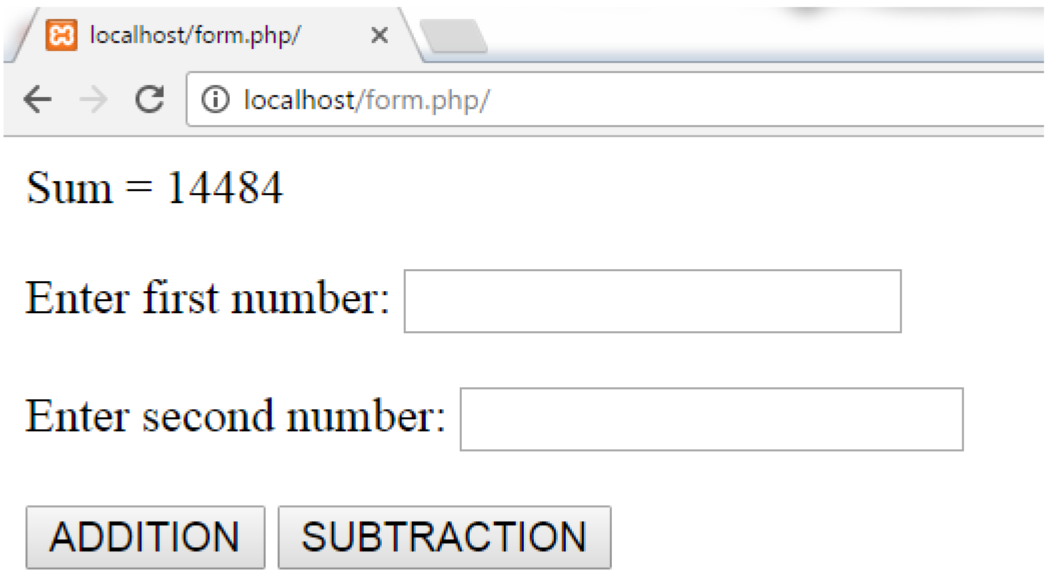
```
localhost/form.php/
Enter first number:
Enter second number:
ADDITION SUBTRACTION
```

We passed the following number,



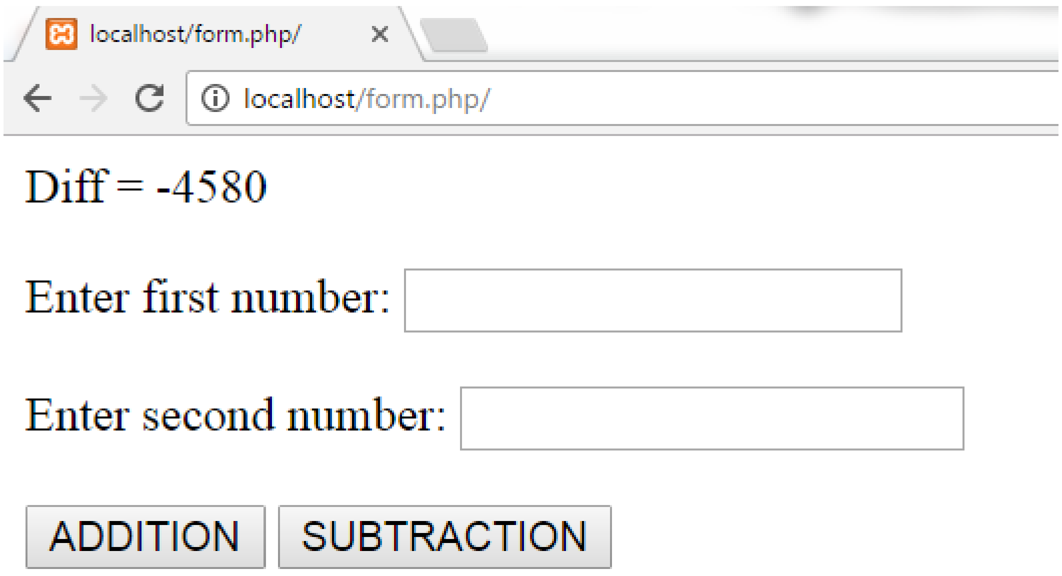
```
localhost/form.php/
Enter first number: 4952
Enter second number: 9532
ADDITION SUBTRACTION
```

Now clicking on ADDITION button, we get the following output.



```
Sum = 14484
```

Now clicking on SUBTRACTION button, we get the following output.



```
Diff = -4580
```

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.

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

Output:

```
Hello
```

Example 2

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

```php
<?php
function increment($i)
{
    $i++;
}
$i = 10;
```

```
increment($i);
echo $i;
?>
```

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.

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

Output:

This is Call By Reference

Example 2

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

```
<?php
function increment(&$i)
{
    $i++;
}
$i = 10;
increment($i);
echo $i;
?>
```

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

Example 1

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

Output:

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

Since PHP 5, you can use the concept of default argument value with call by reference also.

Example 2

```
<?php
function greeting($first="Sonoo",$last="Jaiswal"){
echo "Greeting: $first $last<br/>";
}
greeting();
greeting("Rahul");
greeting("Michael","Clark");
?>
```

Output:

```
Greeting: Sonoo Jaiswal
```

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

Example 3

```php
<?php
function add($n1=10,$n2=10){
$n3=$n1+$n2;
echo "Addition is: $n3<br/>";
}
add();
add(20);
add(40,40);
?>
```

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.

```
<?php
function add(...$numbers) {
    $sum = 0;
    foreach ($numbers as $n) {
        $sum += $n;
    }
    return $sum;
}

echo add(1, 2, 3, 4);
?>
```

Output:

```
10
```

PHP Recursive Function

PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.

It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

Example 1: Printing number

```
<?php
function display($number) {
    if($number<=5){
     echo "$number <br/>";
     display($number+1);
    }
}

display(1);
?>
```

Output:

```
1
2
3
4
5
```

Example 2 : Factorial Number

```
<?php
function factorial($n)
{
    if ($n < 0)
        return -1; /*Wrong value*/
    if ($n == 0)
        return 1; /*Terminating condition*/
    return ($n * factorial ($n -1));
}

echo factorial(5);
?>
```

Output:

```
120
```

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 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

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

Output:

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

```
<?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.

```
<?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"=>"20
0000");
```

2nd way:

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

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"20
0000");
echo "Sonoo salary: ".$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
```

```
<?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
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"20
0000");
foreach($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

```
<?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
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"20
0000");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

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

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"20
0000");
print_r(array_change_key_case($salary,CASE_LOWER));
?>
```

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

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

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"20
0000");
print_r(array_chunk($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);
?>
```

Output:

```
4
```

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 $n )
{
  echo "$n<br />";
}
?>
```

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

```
<?php
       $str='Hello text within single quote';
       echo $str;
?>
```

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

```
<?php
$str1='Hello text
multiple line
text within single quoted string';
$str2='Using  double  "quote"  directly  inside  single  quoted
string';
$str3='Using escape sequences \n in single quoted string';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

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
```

Note: In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \' and backslash through \\ inside single quoted PHP strings.

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

```
<?php
$str="Hello text within double quote";
echo $str;
?>
```

Output:

Hello text within double quote

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

Example 2

```
<?php
$str1="Using  double  "quote"  directly  inside  double  quoted
string";
echo $str1;
?>
```

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.

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

```
<?php
    $str = <<<Demo
It is a valid example
Demo;    //Valid code as whitespace or tab is not valid before
closing identifier
echo $str;
?>
```

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.

```
<?php
    $str = <<<Demo
It is Invalid example
       Demo;    //Invalid code as whitespace or tab is not valid
before closing identifier
echo $str;
?>
```

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

```
<?php
    $city = 'Delhi';
    $str = <<<DEMO
Hello! My name is Misthi, and I live in $city.
DEMO;
    echo $str;
 ?>
```

Output:

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

Example

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

```
<?php
    $str = <<<DEMO
It is the example
of multiple
lines of text.
DEMO;
    echo $str;
echo '</br>';

echo <<<DEMO     // Here we are not storing string content in
variable str.
It is the example
of multiple
lines of text.
DEMO;
 ?>
```

Output:

It is the example of multiple lines of text.

It is the example of multiple lines of text.

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.

Note: Newdoc works as single quotes.

Example-1:

```
<?php
    $str = <<<'DEMO'
    Welcome to javaTpoint.
           Learn with newdoc example.
DEMO;
echo $str;
echo '</br>';
echo <<< 'Demo'    // Here we are not storing string content in
variable str.
    Welcome to javaTpoint.
           Learn with newdoc example.
Demo;
?>
```

Output:

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

PHP Include and Require

PHP allows us to create various elements and functions, which are used several times in many pages. It takes much time to script these functions in multiple pages. Therefore, use the concept of file inclusion that helps to include files in various programs and saves the effort of writing code multiple times.

"PHP allows you to include file so that a page content can be reused many times. It is very helpful to include files when you want to apply the same HTML or PHP code to multiple pages of a website." There are two ways to include file in PHP.

1. include
2. require

Both include and require are identical to each other, except failure.

- include only generates a warning, i.e., E_WARNING, and continue the execution of the script.
- require generates a fatal error, i.e., E_COMPILE_ERROR, and stop the execution of the script.

Advantage

Code Reusability: By the help of include and require construct, we can reuse HTML code or PHP script in many PHP scripts.

Easy editable: If we want to change anything in webpages, edit the source file included in all webpage rather than editing in all the files separately.

PHP include

PHP include is used to include a file on the basis of given path. You may use a relative or absolute path of the file.

Syntax

There are two syntaxes available for include:

include 'filename ';

Or

include ('filename');

Examples

Let's see a simple PHP include example.

```
<a href="http://www.javatpoint.com">Home</a> |
<a href="http://www.javatpoint.com/php-tutorial">PHP</a> |
<a href="http://www.javatpoint.com/java-tutorial">Java</a> |
<a href="http://www.javatpoint.com/html-tutorial">HTML</a>

<?php include("menu.html"); ?>
<h1>This is Main Page</h1>
```

Output:

```
Home |
PHP |
Java |
HTML
This is Main Page
```

PHP require

PHP require is similar to include, which is also used to include files. The only difference is that it stops the execution of script if the file is not found whereas include doesn't.

Syntax

There are two syntaxes available for require:

```
require 'filename';
Or
require ('filename');
```

Examples

Let's see a simple PHP require example.

```
<a href="http://www.javatpoint.com">Home</a> |
<a href="http://www.javatpoint.com/php-tutorial">PHP</a> |
<a href="http://www.javatpoint.com/java-tutorial">Java</a> |
<a href="http://www.javatpoint.com/html-tutorial">HTML</a>

<?php require("menu.html"); ?>
<h1>This is Main Page</h1>
```

Output:

```
Home |

PHP |

Java |

HTML

This is Main Page
```

PHP include vs PHP require

Both include and require are same. But if the file is missing or inclusion fails, include allows the script to continue but require halts the script producing a fatal E_COMPILE_ERROR level error.

Let's understand the difference with the help of example:

Example

```
<?php
    //include welcome.php file
    include("welcome.php");
    echo "The welcome file is included.";
?>
```

Output:

The welcome.php file is not available in the same directory, which we have included. So, it will produce a warning about that missing file but also display the output.

Warning: include(welcome.php): failed to open stream: No such file or directory in C:\xampp\htdocs\program\include.php on line 3

Warning: include(): Failed opening 'welcome.php' for inclusion (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\program\include.php on line 3
The welcome file is included.

```
<?php
    echo "HELLO";
    //require welcome.php file
    require("welcome.php");
    echo "The welcome file is required.";
?>
```

Output:

In case of require() if the file (welcome.php) is not found in the same directory. The require() will generate a fatal error and stop the execution of the script, as you can see in the below output.

```
HELLO
Warning: require(Welcome.php): failed to open stream: No such file or directory
in C:\xampp\htdocs\program\include.php on line 3

Fatal error: require(): Failed opening required 'Welcome.php'
(include_path='C:\xampp\php\PEAR')
```

PHP File Handling

PHP File System allows us to create file, read file line by line, read file character by character, write file, append file, delete file and close file.

1. PHP Open File

PHP fopen() function is used to open file or URL and returns resource. The fopen() function accepts two arguments: $filename and $mode. The $filename represents the file to be opended and $mode represents the file mode for example read-only, read-write, write-only etc.

Syntax

```
resource fopen ( string $filename , string $mode [, bool
$use_include_path = false [, resource $context ]] )
```

PHP Open File Mode

Mode	Description
r	Opens file in read-only mode. It places the file pointer at the beginning of the file.
r+	Opens file in read-write mode. It places the file pointer at the beginning of the file.
w	Opens file in write-only mode. It places the file pointer to the beginning of the file and truncates the file to zero length. If file is not found, it creates a new file.
w+	Opens file in read-write mode. It places the file pointer to the beginning of the file and truncates the file to zero length. If file is not found, it creates a new file.
a	Opens file in write-only mode. It places the file pointer to the end of the file. If file is not found, it creates a new file.

a+	Opens file in read-write mode. It places the file pointer to the end of the file. If file is not found, it creates a new file.
x	Creates and opens file in write-only mode. It places the file pointer at the beginning of the file. If file is found, fopen() function returns FALSE.
x+	It is same as x but it creates and opens file in read-write mode.
c	Opens file in write-only mode. If the file does not exist, it is created. If it exists, it is neither truncated (as opposed to 'w'), nor the call to this function fails (as is the case with 'x'). The file pointer is positioned on the beginning of the file
c+	It is same as c but it opens file in read-write mode.

Example

```
<?php
$handle = fopen("c:\\folder\\file.txt", "r");
?>
```

PHP Close File - fclose()
The PHP fclose() function is used to close an open file pointer.

Syntax

```
ool fclose ( resource $handle )
```

Example

```
<?php
fclose($handle);
?>
```

PHP Read File - fread()

PHP provides various functions to read data from file. There are different functions that allow you to read all file data, read data line by line and read data character by character.

The available PHP file read functions are given below.

- fread()
- fgets()
- fgetc()

PHP Read File - fread()

The PHP fread() function is used to read data of the file. It requires two arguments: file resource and file size.

Syntax

```
string fread (resource $handle , int $length )
```

$handle represents file pointer that is created by fopen() function.

$length represents length of byte to be read.

Example

```
<?php
$filename = "c:\\file1.txt";
$fp = fopen($filename, "r");//open file in read mode

$contents = fread($fp, filesize($filename));//read file

echo "<pre>$contents</pre>";//printing data of file
fclose($fp);//close file
?>
```

Output

```
this is first line
this is another line
this is third line
```

PHP Read File - fgets()

The PHP fgets() function is used to read single line from the file.

Syntax

```
string fgets ( resource $handle [, int $length ] )
```

Example

```
<?php
$fp = fopen("c:\\file1.txt", "r");//open file in read mode
echo fgets($fp);
fclose($fp);
?>
```

Output

this is first line

PHP Read File - fgetc()

The PHP fgetc() function is used to read single character from the file. To get all data using fgetc() function, use !feof() function inside the while loop.

Syntax

```
string fgetc ( resource $handle )
```

Example

```
<?php
$fp = fopen("c:\\file1.txt", "r");//open file in read mode
while(!feof($fp)) {
  echo fgetc($fp);
}
fclose($fp);
?>
```

Output

this is first line this is another line this is third line

PHP Write File

PHP fwrite() and fputs() functions are used to write data into file. To write data into file, you need to use w, r+, w+, x, x+, c or c+ mode.

PHP Write File - fwrite()

The PHP fwrite() function is used to write content of the string into file.

Syntax

```
int fwrite ( resource $handle , string $string [, int $length ]
)
```

Example

```
<?php
$fp = fopen('data.txt', 'w');//opens file in write-only mode
fwrite($fp, 'welcome ');
fwrite($fp, 'to php file write');
fclose($fp);

echo "File written successfully";
?>
```

Output: data.txt

welcome to php file write

PHP Overwriting File

If you run the above code again, it will erase the previous data of the file and writes the new data. Let's see the code that writes only new data into data.txt file.

```
<?php
$fp = fopen('data.txt', 'w');//opens file in write-only mode
fwrite($fp, 'hello');
fclose($fp);

echo "File written successfully";
?>
```

Output: data.txt

Hello

PHP Append to File
You can append data into file by using a or a+ mode in fopen() function. Let's see a simple example that appends data into data.txt file.

Let's see the data of file first.

data.txt

welcome to php file write

PHP Append to File - fwrite()
The PHP fwrite() function is used to write and append data into file.

Example

```
<?php
$fp = fopen('data.txt', 'a');//opens file in append mode
fwrite($fp, ' this is additional text ');
fwrite($fp, 'appending data');
fclose($fp);

echo "File appended successfully";
?>
```

Output: data.txt

welcome to php file write this is additional text appending data

PHP Delete File
In PHP, we can delete any file using unlink() function. The unlink() function accepts one argument only: file name. It is similar to UNIX C unlink() function.

PHP unlink() generates E_WARNING level error if file is not deleted. It returns TRUE if file is deleted successfully otherwise FALSE.

Syntax

```
bool unlink ( string $filename [, resource $context ] )
```

$filename represents the name of the file to be deleted.

PHP Delete File Example

```
<?php
$status=unlink('data.txt');
if($status){
echo "File deleted successfully";
}else{
echo "Sorry!";
}
?>
```

Output

```
File deleted successfully
```

PHP Directory Introduction

The directory functions allow you to retrieve information about directories and their contents.

Installation

The PHP directory functions are part of the PHP core. No installation is required to use these functions.

PHP Directory Functions

Function	
chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle
readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

PHP Error Handling

PHP is used for web development. Error handling in PHP is almost similar to error handling in all programming languages. The default error handling in PHP will give file name line number and error type.

Ways to handle PHP Errors:

- Using die() method
- Custom Error Handling

Basic error handling: Using die() function The die() function print a message and exit from current script.

Syntax:

```
die( $message )
```

Example:

```
<?php

// Php code showing default error handling

$file = fopen("geeks.txt", "w");
?>
```

Note: Run the above code and geeks.txt file is not present then it will display an run-time error message.

Runtime Error:

```
PHP Warning: fopen(geeks.txt): failed to open stream: Permission denied
in /home/dac923dff0a2558b37ba742613273073.php on line 2
```

To prevent this error use die() function. Below is the implementation of die() function:

Example:

```
<?php

// PHP code to check errors

// If file is not present
// then exit from script
if( !file_exists("geeks.txt") ) {
     die("File is not present");
}

// If file is present
// then continue
else {
     $file = fopen("geeks.txt", "w");
}
?>
```

Note: If geeks.txt file not present then it will display output.

Output

File is not present

Custom Error handling: Creating a custom error handler in PHP is quite simple. Create a function that can be called when a error has been occurred in PHP.

Syntax:

```
error_function( $error_level, $error_message, $error_file, $error_line, $error_context)
```

Parameters: This function accepts five parameters as mentioned above and described below:

- **$error_level:** It is required parameter and it must be an integer. There are predefined error levels.
- **$error_message:** It is required parameter and it is the message which user want to print.
- **$error_file:** It is optional parameter and used to specify the file in which error has been occurred.
- **$error_line:** It is optional parameter and used to specify the line number in which error has been occurred.
- **$error_context:** It is optional parameter and used to specify an array containing every variable and their value when error has been occurred.

error_level: These are the possible error level which are listed below:

- 1 : .E_ERROR :fatal runtime error execution of script has been halted
- 2 : E_WARNING :non fatal runtime error execution of script has been halted
- 4 : E_PARSE :compile time error it is generated by the parser
- 8 :E_NOTICE :The script found something that might be an error
- 16 :E_CORE_ERROR :Fatal errors that occurred during initial startup of script
- 32 :E_CORE_WARNING :Non fatal errors that occurred during initial startup of script
- 8191 :E_ALL :All errors and warning

set_error_handler() Function: After creating myerror() function need to set custom error handler because in normal way PHP handles it but if user doing custom error handling then user have to set it in place of argument and pass out myerror function as a string.

Example:

```
<?php

// Creates my error function which prints message
//to user
function myerror($error_no, $error_msg) {
     echo "Error: [$error_no] $error_msg ";
     echo "\n Now Script will end";

     // When error occurred script has to be stopped
     die();
}

// Setting set_error_handler
set_error_handler("myerror");

$a = 10;
$b = 0;

// This will generate error
echo($a / $b);;
?>
```

Output:

```
Error: [2] Division by zero
Now Script will end
```

Conclusion: It is always try to error handling using Custom error handling because it will show more specified message according to the user that can be helpful to the user. If error is not handle using Custom error handling then a error occurred then out script will be halted by default but if it handle error using Custom error handling then it can continue script after displaying error message.