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.

PHP Echo

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

The syntax of PHP echo is given below:

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

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

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

PHP echo: printing string

File: echo1.php

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

Output:

```
Hello by PHP echo
```

PHP echo: printing multi line string

File: echo2.php

```
<?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

File: echo3.php

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

Output:

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

PHP echo: printing variable value

File: echo4.php

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

Output:

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

PHP Print

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

The syntax of PHP print is given below:

1. int print(string $arg)

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

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

PHP print: printing string

File: print1.php

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

Output:

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

PHP print: printing multi line string

File: print2.php

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

Output:

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

PHP print: printing escaping characters

File: print3.php

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

Output:

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

PHP print: printing variable value

File: print4.php

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

Output:

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

PHP echo and print Statements

We frequently use the echo statement to display the output. There are two basic ways to get the output in PHP:

- echo
- print

echo and print are language constructs, and they never behave like a function. Therefore, there is no requirement for parentheses. However, both the statements can be used with or without parentheses. We can use these statements to output variables or strings.

Difference between echo and print

echo

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

print

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

You can see the difference between echo and print statements with the help of the following programs.

For Example (Check multiple arguments)

You can pass multiple arguments separated by a comma (,) in echo. It will not generate any syntax error.

```
<?php
    $fname = "Gunjan";
    $lname = "Garg";
    echo "My name is: ".$fname,$lname;
?>
```

It will generate a **syntax error** because of multiple arguments in a print statement.

```
<?php
    $fname = "Gunjan";
    $lname = "Garg";
    print "My name is: ".$fname,$lname;
?>
```

For Example (Check Return Value)

echo statement does not return any value. It will generate an error if you try to display its return value.

```
<?php
$lang = "PHP";
$ret = echo $lang." is a web development language.";
echo "</br>";
echo "Value return by print statement: ".$ret;
?>
```

Output:

As we already discussed that print returns a value, which is always 1.

```
<?php
$lang = "PHP";
$ret = print $lang." is a web development language.";
print "</br>";
print "Value return by print statement: ".$ret;
?>
```

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.

File: first.php

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

Output:

Hello First PHP

How to run PHP programs in XAMPP

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

For example - p1.php.

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

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

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

```
<?php
echo "Hello World!";
?>
```

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

Note: PHP program must be saved in the htdocs folder, which resides inside the xampp folder, where you installed the XAMPP. Otherwise it will generate an error - Object not found.

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

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

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

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

PHP Case Sensitivity

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

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

```
<!DOCTYPE>
<html>
```

```
<body>
<?php
echo "Hello world using echo </br>";
ECHO "Hello world using ECHO </br>";
EcHo "Hello world using EcHo </br>";
?>
</body>
</html>
```

Output:

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

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

```
<html>
<body>
<?php
$color = "black";
echo "My car is ". $ColoR ."</br>";
echo "My dog is ". $color ."</br>";
echo "My Phone is ". $COLOR ."</br>";
?>
</body>
</html>
```

Output:

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

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

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

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:

1. $variablename=value;

Rules for declaring PHP variable:

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

PHP Variable: Declaring string, integer, and float

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

File: variable1.php

```
1. <?php
2. $str="hello string";
3. $x=200;
4. $y=44.6;
5. echo "string is: $str <br/>";
6. echo "integer is: $x <br/>";
7. echo "float is: $y <br/>";
8. ?>
```

Output:

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

PHP Variable: Sum of two variables

File: variable2.php

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

Output:

```
11
```

PHP Variable: case sensitive

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

File: variable3.php

```
1. <?php
2. $color="red";
```

```
3. echo "My car is " . $color . "<br>";
4. echo "My house is " . $COLOR . "<br>";
5. echo "My boat is " . $coLOR . "<br>";
6. ?>
```

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

PHP variables must start with letter or underscore only.
PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

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

Output:

```
hello
hello
```

File: variableinvalid.php

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

Output:

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

PHP: Loosely typed language

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

PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

Local variable

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.

A variable declaration outside the function with the same name is completely different from the variable declared inside the function. Let's understand the local variables with the help of an example:

File: local_variable1.php

```
<?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
```

File: local_variable2.php

```
<?php
   function mytest()
   {
      $lang = "PHP";
      echo "Web development language: " .$lang;
   }
   mytest();
   //using $lang (local variable) outside the function will generate an error
   echo $lang;
?>
```

Output:

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

Global variable

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

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

Example:

File: global_variable1.php

```
<?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:

File: global_variable2.php

```
<?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:

File: global_variable3.php

```
<?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:

File: global_variable2.php

```
<?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:

File: static_variable.php

```
<?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.

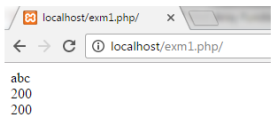
To understand the difference better, let's see some examples.

Example 1

```
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
```

7. ?>

Output:

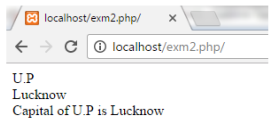


In the above example, we have assigned a value to the variable **x** as **abc**. Value of reference variable **$$x** is assigned as **200**.
Now we have printed the values **$x, $$x** and **$abc**.

Example2

```
1. <?php
2.  $x="U.P";
3. $$x="Lucknow";
4. echo $x. "<br>";
5. echo $$x. "<br>";
6. echo "Capital of $x is " . $$x;
7. ?>
```

Output:

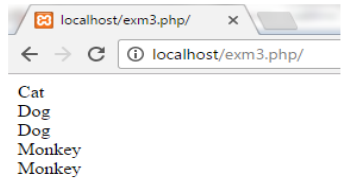


In the above example, we have assigned a value to the variable **x** as **U.P**. Value of reference variable **$$x** is assigned as **Lucknow.**
Now we have printed the values **$x, $$x** and a string.

Example3

```
1. <?php
2. $name="Cat";
3. ${$name}="Dog";
4. ${${$name}}="Monkey";
5. echo $name. "<br>";
6. echo ${$name}. "<br>";
7. echo $Cat. "<br>";
8. echo ${${$name}}. "<br>";
9. echo $Dog. "<br>";
10. ?>
```

Output:



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 Data Types

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

1. Scalar Types (predefined)

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

PHP Data Types: Scalar Types

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

1. boolean
2. integer
3. float
4. string

PHP Data Types: Compound Types

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

1. array
2. object

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. resource
2. NULL

PHP Boolean

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

Example:

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

Output:

```
This condition is TRUE.
```

PHP Integer

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

Rules for integer:

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

Example:

```
<?php
   $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:

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

Output:

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

PHP String

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

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

Example:

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

Output:

```
Hello Javatpoint
Hello $company
```

PHP Array

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

Example:

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

Output:

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

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

PHP object

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

Example:

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

Output:

```
Bike Model: Royal Enfield
```

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

PHP Resource

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

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

PHP Null

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

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

Example:

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

Output:

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.

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

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

File: constant1.php

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

Output:

```
Hello JavaTpoint PHP
```

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

File: constant2.php

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

Output:

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

File: constant3.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**.

File: constant4.php

```
<?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)
```

File: constant5.php

```
<?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 Operators

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

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

In the above example, + is the binary + operator, 10 and 20 are operands and $num is variable.
PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Comparison Operators
- Incrementing/Decrementing Operators
- Logical Operators
- String Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators

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

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

Arithmetic Operators

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

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

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

Assignment Operators

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

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left 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.

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

Output:

```
Charu is a developer.
```

```
bool(true) bool(false)
```

Execution Operators

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

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

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 If Else

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

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

PHP If Statement

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

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

Syntax

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

Flowchart

Example

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

Output:

```
12 is less than 100
```

PHP If-else Statement

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

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

Syntax

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

Flowchart

Example

```
1. <?php
2. $num=12;
3. if($num%2==0){
4. echo "$num is even number";
5. }else{
6. echo "$num is odd number";
7. }
8. ?>
```

Output:

```
12 is even number
```

PHP If-else-if Statement

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

Syntax

```
1. if (condition1){
2. //code to be executed if condition1 is true
```

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

Flowchart

Example

```
<?php
    $marks=69;
    if ($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 While Loop

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

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

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

Syntax

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

Alternative Syntax

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

PHP While Loop Flowchart

PHP While Loop Example

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

Output:

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

Alternative Example

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

Output:

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

Example

Below is the example of printing alphabets using while loop.

```
1. <?php
2.     $i = 'A';
3.     while ($i < 'H') {
4.         echo $i;
```

```
        $i++;
        echo "</br>";
    }
?>
```

Output:

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

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
```

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

Output:

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

Example

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

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

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

    /* example 2 */

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

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 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

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
```

Example 3

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

Output:

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

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

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

Output:

```
Hello text within double quote
```

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

Example 2

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

Output:

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

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

Example 3

```
1. <?php
2. $str1="Hello text
3. multiple line
4. text within double quoted string";
```

```
5. $str2="Using double \"quote\" with backslash inside double quoted string";
6. $str3="Using escape sequences \n in double quoted string";
7. echo "$str1 <br/> $str2 <br/> $str3";
8. ?>
```

Output:

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

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

Example 4

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

Output:

```
Number is: 10
```

Heredoc

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

Naming Rules

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

For Example

Valid Example

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

Output:

```
It is a valid example
```

Invalid Example

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

```
1. <?php
```

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

This code will generate an error.

Output:

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

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

Example

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

Output:

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

Example

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

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

Output:

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

Below are the example with class and their variable

Example

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

Output:

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

PHP String Functions

PHP provides various string functions to access and manipulate strings.

A list of PHP string functions is given below.

addcslashes()	It is used to return a string with backslashes.
addslashes()	It is used to return a string with backslashes.
bin2hex()	It is used to converts a string of ASCII characters to hexadecimal values.
chop()	It removes whitespace or other characters from the right end of a string
chr()	It is used to return a character from a specified ASCII value.

chunk_split()	It is used to split a string into a series of smaller parts.
convert_cyr_string()	It is used to convert a string from one Cyrillic character-set to another.
convert_uudecode()	It is used to decode a uuencoded string.
convert_uuencode()	It is used to encode a string using the uuencode algorithm.
count_chars()	It is used to return information about characters used in a string.
crc32()	It is used to calculate a 32-bit CRC for a string.
crypt()	It is used to create hashing string One-way.
echo()	It is used for output one or more strings.
explode()	It is used to break a string into an array.
fprint()	It is used to write a formatted string to a stream.
get_html_translation_table()	Returns translation table which is used by htmlspecialchars() and htmlentities().
hebrev()	It is used to convert Hebrew text to visual text.
hebrevc()	It is used to convert Hebrew text to visual text and new lines (\n) into .
hex2bin()	It is used to convert string of hexadecimal values to ASCII characters.
htmlentities()	It is used to convert character to HTML entities.
html_entity_decode()	It is used to convert HTML entities to characters.
htmlspecialchars()	Converts the special characters to html entities.
htmlspecialchars_decode()	Converts the html entities back to special characters.
Implode()	It is used to return a string from the elements of an array.
Join()	It is the Alias of implode() function.
Levenshtein()	It is used to return the Levenshtein distance between two strings.
Lcfirst()	It is used to convert the first character of a string to lowercase.
localeconv()	Get numeric formatting information
ltrim()	It is used to remove whitespace from the left side of a string.
md5()	It is used to calculate the MD5 hash of a string.

md5_files()	It is used to calculate MD5 hash of a file.
metaphone()	It is used to calculate the metaphone key of a string.
money_format()	It is used to return a string formatted as a currency string.
nl2br()	It is used to insert HTML line breaks in front of each newline in a string.
nl_langinfo()	Query language and locale information
number_format()	It is used to format a number with grouped thousands.
ord()	It is used to return ASCII value of the first character of a string.
parse_str()	It is used to parse a query string into variables.
print()	It is used for output one or more strings.
printf()	It is used to show output as a formatted string.
quoted_printable_decode()	Converts quoted-printable string to an 8-bit string
quoted_printable_encode()	Converts the 8-bit string back to quoted-printable string
quotemeta()	Quote meta characters
rtrim()	It is used to remove whitespace from the right side of a string.
setlocale()	It is used to set locale information.
sha1()	It is used to return the SHA-1 hash of a string.
sha1_file()	It is used to return the SHA-1 hash of a file.
similar_text()	It is used to compare the similarity between two strings.
Soundex()	It is is used to calculate the soundex key of a string.
sprintf()	Return a formatted string
sscanf()	It is used to parse input from a string according to a format.
strcasecmp()	It is used to compare two strings.
strchr()	It is used to find the first occurrence of a string inside another string.
strcmp()	Binary safe string comparison (case-sensitive)
strcoll()	Locale based binary comparison(case-sensitive)
strcspn()	It is used to reverses a string.

stripcslashes()	It is used to unquote a string quoted with addcslashes().
stripos()	It is used to return the position of the first occurrence of a string inside another string.
stristr()	Case-insensitive strstr
strlen()	It is used to return the length of a string.
strncasecmp()	Binary safe case-insensitive string comparison
strnatcasecmp()	It is used for case-insensitive comparison of two strings using a "natural order" algorithm
strnatcmp()	It is used for case-sensitive comparison of two strings using a "natural order" algorithm
strncmp()	It is used to compare of the first n characters.
strpbrk()	It is used to search a string for any of a set of characters.
strripos()	It finds the position of the last occurrence of a case-insensitive substring in a string.
strrpos()	It finds the length of the last occurrence of a substring in a string.
strpos()	It is used to return the position of the first occurrence of a string inside another string.
strrchr()	It is used to find the last occurrence of a string inside another string.
strrev()	It is used to reverse a string.
strspn()	Find the initial length of the initial segment of the string
strstr()	Find the occurrence of a string.
strtok()	Splits the string into smaller strings
strtolower()	Convert the string in lowercase
strtoupper()	Convert the strings in uppercase
strtr()	Translate certain characters in a string or replace the substring
str_getcsv()	It is used to parse a CSV string into an array.
str_ireplace()	It is used to replace some characters in a string (case-insensitive).
str_pad()	It is used to pad a string to a new length.

str_repeat()	It is used to repeat a string a specified number of times.
str_replace()	It replaces all occurrences of the search string with the replacement string.
str_rot13()	It is used to perform the ROT13 encoding on a string.
str_shuffle()	It is used to randomly shuffle all characters in a string.
str_split()	It is used to split a string into an array.
strcoll()	It is locale based string comparison.
strip_tags()	It is used to strip HTML and PHP tags from a string.
str_word_count()	It is used to count the number of words in a string.
substr()	Return the part of a string
substr_compare()	Compares two strings from an offset up to the length of characters. (Binary safe comparison)
substr_count()	Count the number of times occurrence of a substring
substr_replace()	Replace some part of a string with another substring
trim()	Remove whitespace or other characters from the beginning and end of the string.
ucfirst()	Make the first character of the string to uppercase
ucwords()	Make the first character of each word in a string to uppercase
vfprintf()	Write a formatted string to a stream
vprintf()	Display the output as a formatted string according to format
vsprintf()	It returns a formatted string
wordwrap()	Wraps a string to a given number of characters

PHP String Formatting Function Examples

1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax

```
1. string strtolower ( string $string )
```

Example

```
1. <?php
```

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

Output:

```
my name is khan
```

2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

Syntax

```
1. string strtoupper ( string $string )
```

Example

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

Output:

```
MY NAME IS KHAN
```

3) PHP ucfirst() function

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

Syntax

```
1. string ucfirst ( string $str )
```

Example

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

Output:

```
My name is KHAN
```

4) PHP lcfirst() function

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

Syntax

```
1. string lcfirst ( string $str )
```

Example

```
1. <?php
```

```
2. $str="MY name IS KHAN";
3. $str=lcfirst($str);
4. echo $str;
5. ?>
```

Output:

```
mY name IS KHAN
```

5) PHP ucwords() function

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

Syntax

```
1. string ucwords ( string $str )
```

Example

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

Output:

```
My Name Is Sonoo Jaiswal
```

6) PHP strrev() function

The strrev() function returns reversed string.

Syntax

```
1. string strrev ( string $string )
```

Example

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

Output:

```
lawsiaj oonoS si eman ym
```

7) PHP strlen() function

The strlen() function returns length of the string.

Syntax

```
1. int strlen ( string $string )
```

Example

```
1. <?php
2. $str="my name is Sonoo jaiswal";
```

3. $str=strlen($str);
4. echo $str;
5. ?>

Output:

```
24
```

SEARCHING STRING IN PHP

When writing PHP scripts, you often need to search a string for a particular chunk of text. For example, you might be writing a search engine to search through pages of content, or you might want to know if a URL or email address contains a certain domain name.
PHP gives you many useful functions for searching strings. In this article you look at:

- strstr() for finding out whether some text is in a string
- strpos() and strrpos() for finding the position of some text in a string
- substr_count() for finding out how many times some text appears in a string

Simple text searching with strstr()

PHP's strstr() function simply takes a string to search, and a chunk of text to search for. If the text was found, it returns the portion of the string from the first character of the match up to the end of the string:

```
$myString = 'Hello, there!';
echo strstr( $myString, 'llo' );  // Displays "llo, there!"
```

If the text wasn't found then strstr() returns false. You can use this fact to determine if the text chunk was in the string or not:

```
$myString = 'Hello, there!';
if ( strstr( $myString, 'Goodbye' ) ) {
  echo "Text found";
} else {
  echo "Text not found";
}
```

The above code displays:

```
Text not found
```

strstr() is *case sensitive* — for example, "hello" won't match "Hello". If you don't care about matching case, use the case-insensitive version, stristr(), instead.

Finding the position of a match: strpos() and strrpos()

strpos() takes the same 2 arguments as strstr(). However, rather than returning a portion of the string, it returns the index of the first character of the matched text in the string:

```
$myString = 'Hello, there!';
echo strpos( $myString, 'llo' );  // Displays "2"
```

In the above code, strpos() finds the text 'llo' in the target string starting at the 3rd character, so it returns 2. (Remember that character index positions in strings start from 0, not 1.)
Be careful when using strpos() to check for the existence of a match. The following code incorrectly displays "Not found", because strpos() returns 0, which is equivalent to false in PHP:

```
$myString = 'Hello, there!';
if ( strpos( $myString, 'Hello' ) ) {
  echo "Found";
```

```
} else {
  echo "Not found";
}
```
To fix this, make sure you test explicitly for false by using the === or !== operator. The following code correctly displays “Found”:

```
$myString = 'Hello, there!';
if ( strpos( $myString, 'Hello' ) !== false ) {
  echo "Found";
} else {
  echo "Not found";
}
```
You can pass a third argument to strpos() to specify the index position from which to begin the search:

```
$myString = 'Hello, there!';
echo strpos( $myString, 'e' ) . '<br />';     // Displays "1"
echo strpos( $myString, 'e', 2 ) . '<br />';  // Displays "9"
```
The strrpos() function is very similar to strpos(). The only difference is that it finds the *last* match in the string instead of the first:

```
$myString = 'Hello, there!';
echo strpos( $myString, 'e' ) . "<br />";   // Displays "1"
echo strrpos( $myString, 'e' ) . "<br />";  // Displays "11"
```
As with strstr(), the strpos() and strrpos() functions are case sensitive. Their case-insensitive equivalents are stripos() and strripos() respectively.

String Replacing in PHP

You can replace portions of a string with different text.

Three useful PHP functions for replacing text:

Function	Description
str_replace()	replaces all occurrences of the search text within the target string
substr_replace()	replaces a specified portion of the target string with another string
strtr()	replaces certain characters in the target string with other characters

str_replace():

A String is a collection of characters. The **str_replace()** is a built-in function in PHP and is used to replace all the occurrences of the search string or array of search strings by a replacement string or array of replacement strings in the given string or array respectively. We can replace the string with another by using PHP *str_replace()*. This method is case-

sensitive, so the programmer has to take care of the case while programming and interpreting the output.
Syntax:
str_replace(substring,new_replaced_string,original_string);

We need to provide three parameters.

- su*bstring* is the string present in the original string that should be replaced
- *replaced_string* is the new string that replaces the substring.
- *original_string* is the input string.

Example 1: The following code replaces the string with another string.
PHP

```
<?php
//input string
$string="Geeks for Geeks";
//replace geeks in the string with computer
echo str_replace("Geeks","computer",$string);
?>
```

Output:
computer for computer

Example 2: The following code replaces the string with empty.
PHP

```
<?php
//input string
$string="Geeks for Geeks";
//replace geeks in the string with empty
echo str_replace("Geeks","",$string);
?>
```

Output:
for

Example 3: The following code replaces the string with integers.
PHP

```
<?php
//input string
$string="Geeks for Geeks";
//replace geeks in the string with value - 1
echo str_replace("Geeks",1,$string);
?>
```

Output:
1 for 1

substr_replace()

substr_replace() replaces a specific portion of the target string.

To use it, pass three arguments:

- the string to work on,
- the replacement text, and
- the index within the string at which to start the replacement.

substr_replace() replaces all the characters from the start point to the end of the string with the replacement text, returning the modified string as a copy.

The original string remains untouched.

Demo

```
<?php
$myString = "this is a test test test test test ttttttttt";


echo substr_replace( $myString, "T", 11 ) . " \n ";
?>//from   www  .j  av  a2s . co m
```

Result

```
this is a tT
```


You can specify an optional fourth argument containing the number of characters to replace:

Demo

```
<?php
$myString = "this is a test test test test test ttttttttt,";


echo substr_replace( $myString, "T", 19, 5 ) . " \n ";
?>/*from  ww  w  .ja  va  2s.c o  m*/
```

Result

```
this is a test testT test test ttttttttt,
```


Pass a negative fourth argument to replace up to that many characters from the end of the string:

Demo

```
<?php
$myString = "this is a test test test test test ttttttttt,";


echo substr_replace( $myString, "T", 19, -20 ) . " \n ";
?>/*from ww  w. j av a 2  s. c om*/
```

Result

```
this is a test testTest test ttttttttttt,
```


You can pass a zero value to insert the replacement text into the string rather than replacing characters:

Demo

```
<?php
$myString = "this is a test test test test test ttttttttt,";


echo substr_replace( $myString, "really ", 3, 0 ) . " \n ";
?>/*from   ww  w.  ja v  a 2 s  .c  om*/
```

Result

```
thireally s is a test test test test test ttttttttt,
```

PHP Arrays

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

Advantage of PHP Array

Less Code: We don't need to define multiple variables.
Easy to traverse: By the help of single loop, we can traverse all the elements of an array.
Sorting: We can sort the elements of array.

PHP Array Types

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.
There are two ways to define indexed array:
1st way:

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

2nd way:

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

Example

File: array1.php

```
<?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
```

File: array2.php

```
<?php
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
```

```
$season[3]="autumn";
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

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

PHP Associative Array

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

There are two ways to define associative array:

1st way:

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

2nd way:

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
```

Example

File: arrayassociative1.php

```
<?php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

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

File: arrayassociative2.php

```
<?php
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

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

PHP Multidimensional Array

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

Definition

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

PHP Multidimensional Array Example

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

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

File: multiarray.php

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );
for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]."  ";
  }
  echo "<br/>";
}
?>
```

Output:

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