## What is PHP?

PHP is a server side scripting language. that is used to develop Static websites or Dynamic websites or Web applications. PHP stands for Hypertext Pre-processor, that earlier stood for Personal Home Pages.

PHP scripts can only be interpreted on a server that has PHP installed.

The client computers accessing the PHP scripts require a web browser only.

A PHP file contains PHP tags and ends with the extension “.php”.

## What is a Scripting Language?

A script is a set of programming instructions that is interpreted at runtime.

A scripting language is a language that interprets scripts at runtime. Scripts are usually embedded into other software environments

* 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.

**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.

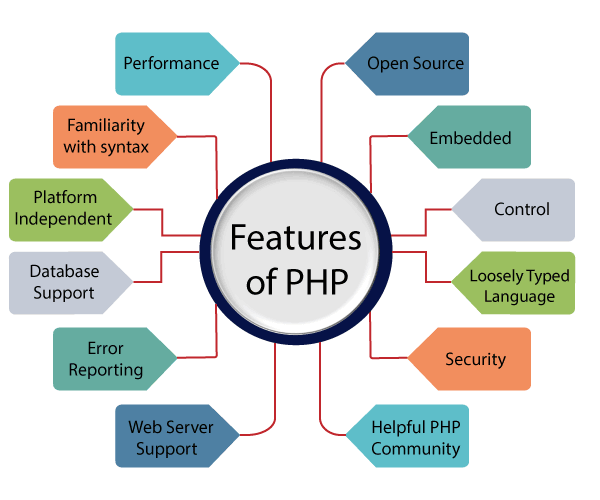
**Embedded:**

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.

|  |  |
| --- | --- |
| * 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. | * 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 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.

$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 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 | 1. <?php 2. **function** mytest() 3. { 4. $lang = "PHP"; 5. echo "Web development language: " .$lang; 6. } 7. mytest(); 8. //using $lang (local variable) outside the function will generate an error 9. echo $lang; 10. ?>   **Output:**  Web development language: PHP  Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 28 |

## Global variable

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

### **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

### **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

<?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 in the function.**

# **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.

|  |  |
| --- | --- |
| **Example 1** <?php  $x = "abc";  $$x = 200;  echo $x."<br/>";  echo $$x."<br/>";  echo $abc;  ?>  **Output:**  PHP $ and $$ variables | **Example3**  1. <?php 2. $name="Cat"; 3. ${$name}="Dog"; 4. {${$name}}="Monkey"; 5. echo $name. "<br>"; 6. echo ${$name}. "<br>"; 7. echo $Cat. "<br>"; 8. echo ${${$name}}. "<br>"; 9. echo $Dog. "<br>"; 10. ?>   **Output:**  PHP $ and $$ variables |

## 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

# **PHP Constants**

PHP constants are name or identifier that can't be changed during the execution of the script except for [magic constants](https://www.javatpoint.com/php-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 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) 2. **name:** It specifies the constant name. 3. **value:** It specifies the constant value. 4. **case-insensitive:** Specifies whether a constant is case-insensitive. Default value is false. It means it is case sensitive by default.      1. **<?php** 2. define("MESSAGE","Hello JavaTpoint PHP"); 3. echo MESSAGE; 4. **?>**   **Output:**  Hello JavaTpoint PHP | 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**.   1. **<?php** 2. const MESSAGE="Hello const by JavaTpoint PHP"; 3. echo MESSAGE; 4. **?>**   **Output:**  Hello const by JavaTpoint PHP  Constant() function  There is another way to print the value of constants using constant() function instead of using the echo statement.  **Syntax**  The syntax for the following constant function:   1. constant (name)      1. **<?php** 2. define("MSG", "JavaTpoint"); 3. echo MSG, "**</br>**"; 4. echo constant("MSG"); 5. //both are similar 6. **?>**   **Output:**  JavaTpoint  JavaTpoint |

# **Magic Constants**

Magic constants are the **predefined constants** in PHP which get **changed** on the **basis of their use**. They start with double underscore (\_\_) and ends with double underscore.

They are similar to other **predefined constants** but as they **change** their **values** with the **context**, they are called **magic** constants.

There are **nine** magic constants in PHP. In which eight magic constants start and end with double underscores (\_\_).

1. [\_\_LINE\_\_](https://www.javatpoint.com/php-magic-constants#LINE)
2. [\_\_FILE\_\_](https://www.javatpoint.com/php-magic-constants#FILE)
3. [\_\_DIR\_\_](https://www.javatpoint.com/php-magic-constants#DIR)
4. [\_\_FUNCTION\_\_](https://www.javatpoint.com/php-magic-constants#FUNCTION)
5. [\_\_CLASS\_\_](https://www.javatpoint.com/php-magic-constants#CLASS)
6. [\_\_TRAIT\_\_](https://www.javatpoint.com/php-magic-constants#TRAIT)
7. [\_\_METHOD\_\_](https://www.javatpoint.com/php-magic-constants#METHOD)
8. [\_\_NAMESPACE\_\_](https://www.javatpoint.com/php-magic-constants#NAMESPACE)
9. [ClassName::class](https://www.javatpoint.com/php-magic-constants#ClassName)

\_\_LINE\_\_: Represents the current line number in the source code.

* \_\_FILE\_\_: Represents the full path of the current file.
* \_\_DIR\_\_: Represents the directory of the current file.
* \_\_FUNCTION\_\_: Represents the name of the current function.
* \_\_CLASS\_\_: Represents the name of the current class.
* \_\_TRAIT\_\_: Represents the name of the current trait.
* \_\_METHOD\_\_: Represents the name of the current method.
* \_\_NAMESPACE\_\_: Represents the current namespace.
* ClassName::class: Returns the fully qualified class name of the ClassName, including the namespace.

# **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](https://www.javatpoint.com/php-data-types#boolean)
2. [integer](https://www.javatpoint.com/php-data-types#integer) 21 4748 3648 – 21 4748 3647
3. [float](https://www.javatpoint.com/php-data-types#float)
4. [string](https://www.javatpoint.com/php-data-types#string)

## PHP Data Types: Compound Types

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

1. [array](https://www.javatpoint.com/php-data-types#array)
2. [object](https://www.javatpoint.com/php-data-types#object)

## PHP Data Types: Special Types

There are 2 special data types in PHP.

1. [resource](https://www.javatpoint.com/php-data-types#resource)
2. [NULL](https://www.javatpoint.com/php-data-types#NULL)

# **PHP Operators**

operators are used to perform operations on variables or values.

* [Arithmetic Operators](https://www.javatpoint.com/php-operators#Arithmetic)
* [Assignment Operators](https://www.javatpoint.com/php-operators#Assignment)
* [Bitwise Operators](https://www.javatpoint.com/php-operators#Bitwise)
* [Comparison Operators](https://www.javatpoint.com/php-operators#Comparison)
* [Incrementing/Decrementing Operators](https://www.javatpoint.com/php-operators#Incrementing)
* [Logical Operators](https://www.javatpoint.com/php-operators#Logical)
* [String Operators](https://www.javatpoint.com/php-operators#String)
* [Array Operators](https://www.javatpoint.com/php-operators#Array)
* [Type Operators](https://www.javatpoint.com/php-operators#Type)
* [Execution Operators](https://www.javatpoint.com/php-operators#Execution)
* [Error Control Operators](https://www.javatpoint.com/php-operators#Error)

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** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| \*\* | Exponentiation |

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** |
| = | Assign |
| += | Add then Assign |
| -= | Subtract then Assign |
| \*= | Multiply then Assign |
| /= | Divide then Assign (quotient) |
| %= | Divide then Assign (remainder) |

## 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();
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** |
| @ |  |  |  |

# **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 |

Here are PHP programs for the tasks you ve described:

Sum of Digits:

$input = 624;

$sum = 0;

while ($input > 0) {

    $sum += $input % 10;

    $input = (int)($input / 10);

}

echo $sum;

Even or Odd Number:

$input = 23;

if ($input % 2 == 0) {

    echo "even number";

} else {

    echo "odd number";

}

Prime Number:

$input = 17;

$isPrime = true;

for ($i = 2; $i <= sqrt($input); $i++) {

    if ($input % $i == 0) {

       $isPrime = false;

        break;

    }

}

if ($isPrime) {

    echo "prime number";

} else {

    echo "not prime number";

}

Table of a Number:

$input = 5;

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

    echo $input \* $i . " ";

}

Factorial:

$input = 5;

$factorial = 1;

for ($i = 1; $i <= $input; $i++) {

    $factorial \*= $i;

}

echo $factorial;

Armstrong Number:

$input = 371;

$original = $input;

$armstrong = 0;

while ($input > 0) {

    $digit = $input % 10;

    $armstrong += $digit \*\* 3;

    $input = (int)($input / 10);

}

if ($armstrong == $original) {

    echo "armstrong";

} else {

    echo "not armstrong";

}

Palindrome Number:

$input = 121;

$original = $input;

$reverse = 0;

while ($input > 0) {

    $digit = $input % 10;

    $reverse = $reverse \* 10 + $digit;

    $input = (int)($input / 10);

}

if ($reverse == $original) {

    echo "palindrome number";

} else {

    echo "not palindrome number";

}

Fibonacci Series (Without Recursion):

$input = 10;

$a = 0;

$b = 1;

echo $a . " " . $b . " ";

for ($i = 2; $i < $input; $i++) {

    $c = $a + $b;

    echo $c . " ";

    $a = $b;

    $b = $c;

}

Reverse Number:

$input = 234;

$reverse = 0;

while ($input > 0) {

    $reverse = $reverse \* 10 + $input % 10;

    $input = (int)($input / 10);

}

echo $reverse;

Reverse String:

$input = "amit";

$reverse = strrev($input);

echo $reverse;

Swap Two Numbers (With Third Variable):

$a = 5;

$b = 10;

$temp = $a;

$a = $b;

$b = $temp;

echo "a=$a b=$b";

Swap Two Numbers (Without Third Variable):

$a = 5;

$b = 10;

$a = $a + $b;

$b = $a - $b;

$a = $a - $b;

echo "a=$a b=$b";

Alphabet Triangle using PHP:

$n = 5;

for ($i = 1; $i <= $n; $i++) {

    // Print leading spaces

    for ($j = $n; $j > $i; $j--) {

        echo " ";

    }

    // Print increasing alphabets

    for ($j = 1; $j <= $i; $j++) {

        echo chr(64 + $j);

    }

    // Print decreasing alphabets

    for ($j = $i - 1; $j > 0; $j--) {

        echo chr(64 + $j);

    }

    echo "\n";

}

A

ABA

ABCBA

ABCDCBA

ABCDEDCBA

Number Triangle:

$n = 6;

for ($i = 1; $i <= $n; $i++) {

    // Print leading spaces

    for ($j = $n; $j > $i; $j--) {

        echo " ";

    }

    // Print increasing numbers

    for ($j = 1; $j <= $i; $j++) {

        echo $j;

    }

    // Print decreasing numbers

    for ($j = $i - 1; $j > 0; $j--) {

        echo $j;

    }

    echo "\n";

}

Star Triangle:

You can create different patterns of star triangles. Heres one example:

$n = 5;

// Print the star triangle

for ($i = 1; $i <= $n; $i++) {

    for ($j = $n; $j > $i; $j--) {

        echo " ";

    }

    for ($j = 1; $j <= $i; $j++) {

        echo "\* ";

    }

    echo "\n";

}

enter the range= 6

1

121

12321

1234321

123454321

12345654321

# **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**:
* **Less Code**:
* **Easy to understand**:

[Parameterized Function](https://www.javatpoint.com/php-parameterized-function)

[PHP Call By Value](https://www.javatpoint.com/php-call-by-value)

[PHP Call By Reference](https://www.javatpoint.com/php-call-by-reference)

[PHP Default Arguments](https://www.javatpoint.com/php-default-argument-values-function)

[PHP Variable Arguments](https://www.javatpoint.com/php-variable-length-argument-function)

[PHP Recursive Function](https://www.javatpoint.com/php-recursive-function)

|  |  |  |  |
| --- | --- | --- | --- |
| PHP **call by value,** actual value is not modified if it is modified inside the function**Example 2** Let's understand PHP call by value concept through another example.   1. <?php 2. **function** increment($i) 3. { 4. $i++; 5. } 6. $i = 10; 7. increment($i); 8. echo $i; 9. ?>   Output:  10 | PHP **call by reference**, actual value is modified if it is modified inside the function  **Example 2**  Let's understand PHP call by reference concept through another example.   1. <?php 2. **function** increment(&$i) 3. { 4. $i++; 5. } 6. $i = 10; 7. increment($i); 8. echo $i; 9. ?>   Output:  11 | **Default Argument Values Function**   1. <?php 2. **function** sayHello($name="Ram"){ 3. echo "Hello $name<br/>"; 4. } 5. sayHello("Sonoo"); 6. sayHello();//passing no value 7. sayHello("Vimal"); 8. ?>   Output:  Hello Sonoo  Hello Ram  Hello Vimal | **PHP Variable Length Argument Function** It means you can pass 0, 1 or n number of arguments in function. To do so, you need to use 3 ellipses (dots) before the argument name.  The 3 dot concept is implemented for variable length argument since PHP 5.6.  .   1. <?php 2. **function** add(...$numbers) { 3. $sum = 0; 4. **foreach** ($numbers **as** $n) { 5. $sum += $n; 6. } 7. **return** $sum; 8. } 10. echo add(1, 2, 3, 4); 11. ?>   Output:  10 |

### **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 index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.  There are two ways to define indexed array:  1st way:   1. $season=**array**("summer","winter","spring","autumn");   2nd way:   1. $season[0]="summer"; 2. $season[1]="winter"; 3. $season[2]="spring"; 4. $season[3]="autumn";  Example *File: array1.php*   1. <?php 2. $season=**array**("summer","winter","spring","autumn"); 3. echo "Season are: $season[0], $season[1], $season[2] and $season[3]"; 4. ?>   Output:  Season are: summer, winter, spring and autumn | PHP Associative Array We can associate name with each array elements in PHP using => symbol.  There are two ways to define associative array:  1st way:   1. $salary=**array**("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");   2nd way:   1. $salary["Sonoo"]="350000"; 2. $salary["John"]="450000"; 3. $salary["Kartik"]="200000"; 4. <?php 5. $salary=**array**("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000"); 6. echo "Sonoo salary: ".$salary["Sonoo"]."<br/>"; 7. echo "John salary: ".$salary["John"]."<br/>"; 8. echo "Kartik salary: ".$salary["Kartik"]."<br/>"; 9. ?>   Output:  Sonoo salary: 350000  John salary: 450000  Kartik salary: 200000 |  |

# **Multidimensional Array**

PHP multidimensional array is also known as array of arrays

<?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

array\_push(): Adds one or more elements to the end of an array.

$fruits = ["apple", "banana"];

array\_push($fruits, "cherry", "date");

array\_pop(): Removes and returns the last element of an array.

$fruits = ["apple", "banana", "cherry"];

$lastFruit = array\_pop($fruits);

array\_unshift(): Adds one or more elements to the beginning of an array.

$fruits = ["banana", "cherry"];

array\_unshift($fruits, "apple");

array\_shift(): Removes and returns the first element of an array.

$fruits = ["apple", "banana", "cherry"];

$firstFruit = array\_shift($fruits);

array\_merge(): Combines two or more arrays into a new array.

$fruits1 = ["apple", "banana"];

$fruits2 = ["cherry", "date"];

$combined = array\_merge($fruits1, $fruits2);

implode() / join(): Joins array elements with a string.

$fruits = ["apple", "banana", "cherry"];

$result = implode(", ", $fruits);

array\_slice(): Extracts a portion of an array and returns a new array.

$fruits = ["apple", "banana", "cherry", "date"];

$portion = array\_slice($fruits, 1, 2);

array\_splice(): Removes a portion of an array and replaces it with something else.

$fruits = ["apple", "banana", "cherry", "date"];

$removed = array\_splice($fruits, 1, 2);

array\_reverse(): Reverses the order of elements in an array.

$fruits = ["apple", "banana", "cherry"];

$reversed = array\_reverse($fruits);

sort(): Sorts an array in ascending order.

$fruits = ["banana", "cherry", "apple"];

sort($fruits);

array\_change\_key\_case(): Changes all keys in an array to lowercase.

$colors = ["Red" => 1, "Green" => 2, "Blue" => 3];

$lowercaseKeys = array\_change\_key\_case($colors, CASE\_LOWER);

print\_r($lowercaseKeys);

Output:

Array

(

    [red] => 1

    [green] => 2

    [blue] => 3

)

array\_chunk(): Splits an array into chunks of a specified size.

$numbers = [1, 2, 3, 4, 5, 6, 7, 8];

$chunks = array\_chunk($numbers, 3);

print\_r($chunks);

Output:

Array

(

    [0] => Array

        (

            [0] => 1

            [1] => 2

            [2] => 3

        )

    [1] => Array

        (

            [0] => 4

            [1] => 5

            [2] => 6

        )

    [2] => Array

        (

            [0] => 7

            [1] => 8

        )

)

count(): Returns the number of elements in an array.

fruits = ["apple", "banana", "cherry"];

$count = count($fruits);

echo $count;

Output:

3

array\_search(): Searches for a value in an array and returns the corresponding key if found.

$fruits = ["apple", "banana", "cherry"];

$key = array\_search("banana", $fruits);

echo $key;

Output:

1

array\_intersect(): Computes the intersection of arrays.

$fruits1 = ["apple", "banana", "cherry"];

$fruits2 = ["banana", "cherry", "date"];

$intersection = array\_intersect($fruits1, $fruits2);

print\_r($intersection);

Output:

Array

(

    [1] => banana

    [2] => cherry

)

# **PHP String**

PHP string is a sequence of characters i.e., used to store and manipulate text.

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

strlen(): Returns the length of a string.

$text = "Hello, World!";

$length = strlen($text);

echo $length;

Output: 13

str\_replace(): Replaces all occurrences of a search string with a replacement string.

$text = "The quick brown fox jumps over the lazy dog.";

$newText = str\_replace("fox", "cat", $text);

echo $newText;

Output: The quick brown cat jumps over the lazy dog.

strpos(): Finds the position of the first occurrence of a substring in a string.

$text = "Hello, World!";

$position = strpos($text, "World");

echo $position;

Output: 7

substr(): Returns a part of a string.

$text = "This is a sample text.";

$substring = substr($text, 5, 7);

echo $substring;

Output: is a sa

strtolower(): Converts a string to lowercase.

$text = "Hello, World!";

$lowercaseText = strtolower($text);

echo $lowercaseText;

Output: hello, world!

strtoupper(): Converts a string to uppercase.

$text = "Hello, World!";

$uppercaseText = strtoupper($text);

echo $uppercaseText;

Output: HELLO, WORLD!

trim(): Removes whitespace or other characters from the beginning and end of a string.

$text = "   Hello, World!   ";

$trimmedText = trim($text);

echo $trimmedText;

Output: Hello, World!

explode(): Splits a string into an array based on a specified delimiter.

$text = "apple,banana,cherry";

$fruits = explode(",", $text);

print\_r($fruits);

Output: Array ( [0] => apple [1] => banana [2] => cherry )

implode(): Joins array elements into a string using a specified separator.

$fruits = ["apple", "banana", "cherry"];

$text = implode(", ", $fruits);

echo $text;

Output: apple, banana, cherry

Math Functions

    $number = -3.7;

    $power = 3;

    $numbers = [5, 2, 9, 1, 7, 4];

    $absolute = abs($number); // Returns the absolute value of the number.

    $squareRoot = sqrt($absolute); // Calculates the square root of the absolute value.

    $rounded = round($squareRoot); // Rounds the square root to the nearest integer.

    $maximum = max($numbers); // Finds the highest value in the array.

    $minimum = min($numbers); // Finds the lowest value in the array.

    $random = rand(1, 100); // Generates a random integer between 1 and 100.

    $pi = pi(); // Returns the value of π (pi).

    $powerResult = pow($number, $power); // Raises the number to the specified power.

    echo "Original Number: $number\n";

    echo "Absolute Value: $absolute\n";

    echo "Square Root: $squareRoot\n";

    echo "Rounded: $rounded\n";

    echo "Maximum Value: $maximum\n";

    echo "Minimum Value: $minimum\n";

    echo "Random Number: $random\n";

    echo "Value of π: $pi\n";

    echo "Number Raised to Power $power: $powerResult\n";

Original Number: -3.7

Absolute Value: 3.7

Square Root: 1.9235384061671

Rounded: 2

Maximum Value: 9

Minimum Value: 1

Random Number: [Random integer between 1 and 100]

Value of π: 3.1415926535898

Number Raised to Power 3: -50.653

1. **GLOBALS**: This superglobal allows you to access global variables from anywhere in your script. It's an associative array where each key is a variable name, and the corresponding value is the variable's value.
2. **$\_SERVER**: Contains information about the server environment and the current request. You can use it to access details like the current page, server software, user agent, and more.
3. **$\_GET**: Contains data sent to the script via **URL query** parameters. It's used for retrieving data from the query string.
4. **$\_POST**: Contains data sent to the script via HTTP POST requests. It's used for submitting data through forms.
5. **$\_REQUEST**: Combines data from $\_GET, $\_POST, and $\_COOKIE into a single array. It's used to access user input regardless of the request method.
6. **$\_SESSION**: Provides access to session variables, allowing you to store and retrieve data between page requests.
7. **$\_COOKIE**: Contains data sent to the script via HTTP cookies. It's used for storing small amounts of data on the client's computer.
8. **$\_FILES**: Contains information about file uploads via HTTP POST. It's used when handling file uploads in web forms.
9. **$\_ENV**: Contains information about the environment variables on the server.
10. **$\_GLOBALS**: A deprecated superglobal array that is no longer available in PHP.

# **PHP Form Handling**

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $\_GET and $\_POST.

The form request may be get or post. To retrieve data from get request, we need to use $\_GET, for post request $\_POST.

|  |  |
| --- | --- |
| PHP Get Form Get request is the **default** form request. The data passed through get request is **visible on the URL browser** so it is **not secured**. You can **send limited amount** of data through get request.  *File: form1.html*  <form action="welcome.php" method="get">  Name: <input type="text" name="name"/>  <input type="submit" value="visit"/>  </form>  *File: welcome.php*  <?php  $name=$\_GET["name"];//receiving name field value in $name variable  echo "Welcome, $name";  ?> | PHP Post Form Post request is **widely used** to submit form that have **large** **amount** of data such as file **upload, image upload, login form**, **registration** form etc.  The data passed through post request is **not visibl**e on the URL browser so it is **secured.** You can **send large amount of dat**a through post request.  *File: form1.html*  <form action="login.php" method="post">  <table>  <tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>  <tr><td>Password:</td><td> <input type="password" name="password"/></td></tr>  <tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>  </table>  </form>  *File: login.php*  <?php  $name=$\_POST["name"];//receiving name field value in $name variable  $password=$\_POST["password"];//receiving password field value in $password variable    echo "Welcome: $name, your password is: $password";  ?> |

PHP Include and Require

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.

|  |  |
| --- | --- |
| 1. **include** 'filename '; 2. Or 3. **include** ('filename'); | 1. **require** 'filename'; 2. Or 3. **require** ('filename'); |

# **PHP Cookie**

PHP cookie is a small piece of information which is stored at client browser. It is used to recognize the user.

Cookie is created at server side and saved to client browser. Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.

In short, cookie can be created, sent and received at server end.

#### **Note: PHP Cookie must be used before <html> tag.**

## PHP setcookie() function

PHP setcookie() function is used to set cookie with HTTP response. Once cookie is set, you can access it by $\_COOKIE superglobal variable.

setcookie("CookieName", "CookieValue");/\* defining name and value only\*/

setcookie("CookieName", "CookieValue", time()+1\*60\*60);//using expiry in 1 hour(1\*60\*60 seconds or 3600 seconds)

setcookie("CookieName", "CookieValue", time()+1\*60\*60, "/mypath/", "mydomain.com", 1);

<?php

setcookie("user", "Sonoo");

?>

<html>

<body>

<?php

**if**(!isset($\_COOKIE["user"])) {

    echo "Sorry, cookie is not found!";

} **else** {

    echo "<br/>Cookie Value: " . $\_COOKIE["user"];  // for access cookies we need to use $\_COOKIE

}

?>

</body>

</html>

## PHP Delete Cookie

If you set the expiration date in past, cookie will be deleted.

<?php

setcookie ("CookieName", "", time() - 3600);// set the expiration date to one hour ago

?>

# **PHP Session**

PHP session is used to store and pass information from one page to another temporarily (until user close the website).

PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.

## PHP $\_SESSION

PHP $\_SESSION is an associative array that contains all session variables. It is used to set and get session variable values.

**Example: Store information**

1. $\_SESSION["user"] = "Sachin";

**Example: Get information**

1. echo $\_SESSION["user"];

## PHP Session Example

*File: session1.php*

<?php

session\_start();

?>

<html>

<body>

<?php

$\_SESSION["user"] = "Sachin";

echo "Session information are set successfully.<br/>";

?>

<a href="session2.php">Visit next page</a>

</body>

</html>

*File: session2.php*

<?php

session\_start();

?>

<html>

<body>

<?php

echo "User is: ".$\_SESSION["user"];

?>

</body>

</html>

## Destroying Session

PHP session\_destroy() function is used to destroy all session variables completely.

*File: session3.php*

<?php

session\_start();

session\_destroy();

?>

# **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.

https://www.javatpoint.com/php-file

<?php

// File path

$filePath = 'example.txt';

// Open a file for writing (creates a new file or overwrites an existing one)

$file = fopen($filePath, 'w') or die("Unable to open file for writing!");

// Write content to the file

$content = "This is some text that will be written to the file.\n";

fwrite($file, $content);

// Close the file

fclose($file);

// Open the file for reading

$file = fopen($filePath, 'r') or die("Unable to open file for reading!");

// Read and display the content from the file

echo "File Content (Read):\n";

while (!feof($file)) {

    echo fgets($file);

}

// Close the file

fclose($file);

// Open the file for appending

$file = fopen($filePath, 'a') or die("Unable to open file for appending!");

// Append more content to the file

$additionalContent = "This is additional content appended to the file.\n";

fwrite($file, $additionalContent);

// Close the file

fclose($file);

// Open the file for deletion

unlink($filePath);

// Check if the file still exists

if (!file\_exists($filePath)) {

    echo "The file has been successfully deleted.";

} else {

    echo "Failed to delete the file.";

}

?>

Read files

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

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

* fgets()
* fgetc()

# **PHP File Upload**

PHP allows you to upload single and multiple files through few lines of code only.

PHP file upload features allows you to upload binary and text files both.

## PHP $\_FILES

The PHP global $\_FILES contains all the information of file. By the help of $\_FILES global, we can get file name, file type, file size, temp file name and errors associated with file.

Here, we are assuming that file name is filename.

### **$\_FILES['filename']['name']**

returns file name.

### **$\_FILES['filename']['type']**

returns MIME type of the file.

### **$\_FILES['filename']['size']**

returns size of the file (in bytes).

### **$\_FILES['filename']['tmp\_name']**

returns temporary file name of the file which was stored on the server.

### **$\_FILES['filename']['error']**

returns error code associated with this file.

## move\_uploaded\_file() function

The move\_uploaded\_file() function moves the uploaded file to a new location.

## PHP File Upload Example

*File: uploadform.html*

<form action="uploader.php" method="post" enctype="multipart/form-data">

    Select File:

    <input type="file" name="fileToUpload"/>

    <input type="submit" value="Upload Image" name="submit"/>

</form>

*File: uploader.php*

<?php

$target\_path = "e:/";

$target\_path = $target\_path.basename( $\_FILES['fileToUpload']['name']);

**if**(move\_uploaded\_file($\_FILES['fileToUpload']['tmp\_name'], $target\_path)) {

    echo "File uploaded successfully!";

} **else**{

    echo "Sorry, file not uploaded, please try again!";

}

?>

## PHP Download File Example: Text File

*File: download1.php*

1. **<?php**
2. $file\_url = 'http://www.javatpoint.com/f.txt';
3. header('Content-Type: application/octet-stream');
4. header("Content-Transfer-Encoding: utf-8");
5. header("Content-disposition: attachment; filename=\"" . basename($file\_url) . "\"");
6. readfile($file\_url);
7. **?>**

## PHP Download File Example: Binary File

*File: download2.php*

1. **<?php**
2. $file\_url = 'http://www.myremoteserver.com/file.exe';
3. header('Content-Type: application/octet-stream');
4. header("Content-Transfer-Encoding: Binary");
5. header("Content-disposition: attachment; filename=\"" . basename($file\_url) . "\"");
6. readfile($file\_url);
7. **?>**

## PHP MySQLi

1.Connect to a MySQL Database:

$servername = "localhost";

$username = "username";

$password = "password";

$database = "dbname";

// Create a connection

$conn = mysqli\_connect($servername, $username, $password, $database);

// Check connection

if (!$conn) {

    die("Connection failed: " . mysqli\_connect\_error());

}

2. Insert Data into a Table:

$sql = "INSERT INTO users (first\_name, last\_name, email) VALUES ('John', 'Doe', 'johndoe@example.com')";

if (mysqli\_query($conn, $sql)) {

    echo "Record inserted successfully";

} else {

    echo "Error: " . $sql . "<br>" . mysqli\_error($conn);

}

3. Select Data from a Table:

$sql = "SELECT \* FROM users";

$result = mysqli\_query($conn, $sql);

if (mysqli\_num\_rows($result) > 0) {

    while ($row = mysqli\_fetch\_assoc($result)) {

        echo "ID: " . $row["id"] . " - Name: " . $row["first\_name"] . " " . $row["last\_name"] . "<br>";

    }

} else {

    echo "No records found";

}

4. Update Data in a Table:

$sql = "UPDATE users SET email='newemail@example.com' WHERE id=1";

if (mysqli\_query($conn, $sql)) {

    echo "Record updated successfully";

} else {

    echo "Error updating record: " . mysqli\_error($conn);

}

5. Delete Data from a Table:

$sql = "DELETE FROM users WHERE id=2";

if (mysqli\_query($conn, $sql)) {

    echo "Record deleted successfully";

} else {

    echo "Error deleting record: " . mysqli\_error($conn);

}

6. Close the Database Connection:

mysqli\_close($conn);

# **Access Specifiers in PHP**

There are 3 types of Access Specifiers available in PHP, Public, Private and Protected.

**Public** - class members with this access modifier will be publicly accessible from anywhere, even from outside the scope of the class.

**Private** - class members with this keyword will be accessed within the class itself. It protects members from outside class access with the reference of the class instance.

**Protected** - same as private, except by allowing subclasses to access protected superclass members.

# **Const keyword**

* Constants are one type of variable which we can define for any class with keyword const.
* Value of these variables **cannot** be **changed** any how after assigning.
* Class constants are different than normal variables, as we do **not need $** to declare the class constants.
* If we are inside the class then values of the constants can be get using **self** keyword, but accessing the value outside the class you have to use **Scope Resolution Operator**.

|  |
| --- |
| 1. <?php 2. //create class 3. **class** javatpoint 4. { 5. //create constant variable 6. **const** a= "This is const keyword example"; 7. Echo self::a; 8. } 9. //call constant variable. 10. echo javatpoint::a; 11. ?> |

# **CONSTRUCTOR**

* **PHP** 5 allows developers to declare **constructor methods for classes**.
* Constructor is suitable for any **initialization that the object may need before it is used**.
* We can design constructor using **"\_\_construct" or same name as class name**.
* Parent constructors are not called implicitly if the child class defines a constructor. In order to run a parent constructor, a call to **parent::\_\_construct()**.

|  |  |
| --- | --- |
| Example 1 <?php  **class** Example      {  **public** **function** \_\_construct()          {              echo "Hello javatpoint";          }      }      $obj = **new** Example();      $obj = **new** Example();  ?>  **Output:**  CONSTRUCTOR | Example 2 <?php  **class** demo      {  **public** **function** demo()          {              echo "constructor1...";          }      }    **class** demo1 **extends** demo      {  **public** **function** \_\_construct()          {              echo parent::demo();              echo "constructor2...";          }      }      $obj= **new** demo1();  ?>  **Output:**  CONSTRUCTOR |

# **DESTRUCTOR**

* **PHP 5** introduces a destructor concept similar to that of other object-oriented languages, such as C++.
* The destructor method will be called as soon as all references to a particular object are removed or when the object is **explicitly destroyed in any order in shutdown sequence.**
* We create destructor by using **"\_\_destruct" function.**

## Example 1

|  |
| --- |
| <    **class** demo      {  **public** **function** demo()          {              echo "constructor1...";          }      }    **class** demo1 **extends** demo      {  **public** **function** \_\_construct()          {              echo parent::demo();              echo "constructor2...";          }  **public** **function** \_\_destruct()          {              echo "destroy.....";          }      }      $obj= **new** demo1();  ?>  **Output:**  DESTRUCTOR |

# **Difference between Abstract class and Interfaces.**

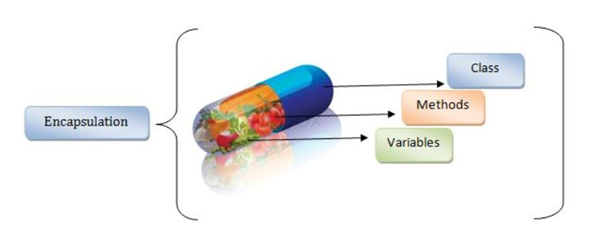
## Abstract class:

* Abstract class comes under partial abstraction.
* Abstract classes can maintain **abstract methods and non abstract methods**.
* In abstract classes, we can **create the variables.**
* In abstract classes, we can use **any access specifier**.
* By using **'extends'** keyword we can access the abstract class features from derived class.
* Multiple **inheritance is not possible**.

Interface:

* Interface comes under fully abstraction.
* Interfaces can maintain **only abstract methods.**
* In interfaces, we **can't create the variables**.
* In interface, we can use only **public access specifier**.
* By using **'implement'** keyword we can get interface from derived class.
* By using interfaces **multiple inheritance is possible**.

# **Encapsulation in PHP**



* Encapsulation is a concept where we encapsulate all the **data and member functions** together to form an object.
* **Wrapping up data member and method together** into a single unit is called Encapsulation.
* Encapsulation also allows a **class to change its internal implementation** without hurting the overall functioning of the system.
* Binding the data with the code that manipulates it.
* It keeps the data and the code **safe from external interference**.

class Person {

    private $name;

    private $age;

    public function setName($name) {

        $this->name = $name;

    }

    public function setAge($age) {

        if ($age >= 0) {

            $this->age = $age;

        }

    }

    public function getName() {

        return $this->name;

    }

    public function getAge() {

        return $this->age;

    }

}

$person = new Person();

$person->setName("John");

$person->setAge(30);

echo "Name: " . $person->getName() . "\n";

echo "Age: " . $person->getAge() . "\n";

# **Final Keyword**

* In PHP, Final keyword is applicable to only class and class methods. We cannot declare as Final in PHP.
* So if we declare class method as a Final then that method cannot be override by the child class.
* Same as method if we declare class as a Final then that class cannot be extended any more.

|  |  |
| --- | --- |
| 1. <?php 2. **class** base 3. { 4. final **public** **function** dis1() 5. { 6. echo "Base class.."; 7. } 8. } 9. **class** derived **extends** base 10. { 11. **public** **function** dis1() 12. { 13. echo "derived class"; 14. } 15. } 16. $obj = **new** derived(); 17. $obj->dis1(); 18. ?> | **Output:**  Final Keyword |

# **Some Helpful Functions in PHP to get the Information About Class and Object**

1. **get\_class()** - Get the class name of an object.
2. **is\_a()** - Check if an object is an instance of a specific class.
3. **get\_class\_methods()** - Get an array of class methods.
4. **get\_object\_vars()** - Get an array of object properties.
5. **property\_exists()** - Check if a property exists in an object.
6. **get\_parent\_class()** - Get the name of an object's parent class.
7. **method\_exists()** - Check if a method exists in a class.

|  |  |
| --- | --- |
| **Inheritance** It is a concept of accessing the features of one class from another class. If we inherit the class features into another class, we can access both class properties. We can extends the features of a class by using 'extends' keyword.   * It supports the concept of **hierarchical classification**. * Inheritance has three types, **single, multiple and multilevel Inheritance**. * **PHP** supports only **single inheritance**, where only one class can be **derived from single parent class**. * We can simulate multiple inheritance by using **interfaces**.  1. <?php 2. **class** a 3. { 4. **function** fun1() 5. { 6. echo "javatpoint"; 7. } 8. } 9. **class** b **extends** a 10. { 11. **function** fun2() 12. { 13. echo "SSSIT"; 14. } 15. } 16. $obj= **new** b(); 17. $obj->fun1(); 18. ?>   **Output:** javatpoint | **Interface** An **interface is similar to a class** except that it cannot contain code.  An interface can define **method names and arguments**, but not the contents of the methods.  Any classes implementing an interface must **implement all methods defined by the interface**.  A class can implement **multiple interfaces**.  An interface is declared using the "**interface**" keyword.  **Interfaces** can't maintain Non-abstract methods.  <?php  **interface** a      {  **public** **function** dis1();      }  **interface** b      {  **public** **function** dis2();      }    **class** demo **implements** a,b  {  **public** **function** dis1()      {          echo "method 1...";      }  **public** **function** dis2()      {          echo "method2...";      }  }  $obj= **new** demo();  $obj->dis1();  $obj->dis2();    ?> |

# **PHP is\_null() function**

By using the is\_null function, we can check whether the variable is NULL or not.

## Return Type:

The PHP is\_null() function returns true if var is null, otherwise false.

|  |  |  |
| --- | --- | --- |
| <?php      $var1 = TRUE;  **if** (is\_null($var1))          {              echo 'Variable is  NULL';          }  **else**          {              echo 'Variable is not NULL';          }  ?>  Output :  Variable is not NULL | 1. <?php 2. $x= 100; 3. unset($x); 4. echo is\_null($x); 5. ?>   Output : 1 | 1. <?php 2. $x = NULL; 3. $y = "\0"; 4. is\_null($x) ? print\_r("True\n") : print\_r("False\n"); 5. echo "<br/>"; 6. is\_null($y) ? print\_r("True\n") : print\_r("False\n"); 7. ?>   Output :True  False |

# **Special Types**

There are 2 special data types in PHP

1. [Resource](https://www.javatpoint.com/php-special-types#resource)
2. [Null](https://www.javatpoint.com/php-special-types#null)

## Resource Data type:

. In simple terms, a resource is a special variable which carrying a reference to an external resource.

## Example 1

<?php

    $conn = ftp\_connect("127.0.0.1") **or** **die**("Could not connect");

    echo get\_resource\_type($conn);

?>

//FTPBuffer

## Null Data Type:

A variable of type Null is a variable without any data. In PHP, null is not a value, and we can consider it as a null variable based on 3 condition.

1. If the variable is not set with any value.
2. If the variable is set with a null value.
3. If the value of the variable is unset.

|  |  |  |
| --- | --- | --- |
| Example 1  <?php      $empty=null;  var\_dump($empty);  ?>  //null | Example 2  1. <?php 2. $a1 = " "; 3. var\_dump($a1);  //string ‘ ’ length 1 4. echo "<br />"; 5. $a2 = null; 6. var\_dump($a2);  // null 7. ?> | Example 3  1. <?php 2. $x = NULL; 3. var\_dump($x); 4. echo "<br>"; 5. $y = "Hello javatpoint!"; 6. $y = NULL; 7. var\_dump($y); 8. ?>  // null |

# **Heredoc Syntax**

By using the syntax, we can display the HTML elements through PHP Script.

## Syntax

1. <<<name of string
2. //content
3. name of string

|  |  |
| --- | --- |
| Example 1 <?php      echo <<<javatpoint      welcome to the best tutorial site  javatpoint;  ?>  **Output:**  Heredoc Syntax | Example 2 Save as string.php  Heredoc Syntax  **Output:**  Heredoc Syntax |

There are usually different types of error. In [PHP](https://www.javatpoint.com/php-tutorial), mainly four types of errors are considered:

1. Syntax Error or Parse Error
2. Fatal Error
3. Warning Error
4. Notice Error

**Syntax Error or Parse Error**: These errors occur when PHP encounters code that violates its grammar rules, **preventing the script from running**. They must be fixed before execution.

**Fatal Error**: Fatal errors are severe issues that cause the **script to terminate** abruptly. They result from **critical issues** like calling an undefined function and need debugging.

**Warning Error**: These are non-fatal errors that allow the **script to continue running**. They indicate potential issues but don't halt execution, like using deprecated functions.

**Notice Error**: Notices are the least severe, informing about non-critical issues, such as accessing undefined variables. They **won't interrupt script execution**.