PHP

History?

PHP is developed by the **Rasmus Lerdorf** in 1994 with the very first version of PHP that simply designed to set the Common Gateway Interface (CGI) binaries, which are written in C programming language. The latest version of PHP is PHP versions 8 which is released on November 24, 2022. It can be easily embedded with HTML files. HTML codes can also be written in a PHP file. The PHP codes are executed on the server-side whereas HTML codes are directly executed on the browser.

What is php?

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

- 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.
- 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.
- ---
- It handles dynamic content, database as well as session tracking for the website.
- You can create sessions in PHP.
- It can access cookies variable and also set cookies.
- It helps to encrypt the data and apply validation.
- PHP supports several protocols such as HTTP, POP3, SNMP, LDAP, IMAP, and many more.
- Using PHP language, you can control the user to access some pages of your website.
- As PHP is easy to install and set up, this is the main reason why PHP is the best language to learn.
- PHP can handle the forms, such as - collect the data from users using forms, save it into the database, and return useful information to the user. **For example** - Registration form.
- PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.
- PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

Disadvantages of PHP:

1. *PHP is not secure as it is open source.*
2. *Not good to create desktop applications.*
3. *Not suitable for large Web Applications- Php code is hard to maintain since it is not very modular.*
4. *Modification Problem – PHP does not allow the change in the core behavior of the web applications.*

Web Development:

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

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

Prerequisite

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

HTML - HTML is used to design static webpage.

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

JavaScript - JavaScript is used to design an interactive website.

How to run PHP code in XAMPP

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

Note: PHP statements ends with semicolon (;).

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

```
<?php
```

```
    //your code here
?>
```

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

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

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!";
?>
```

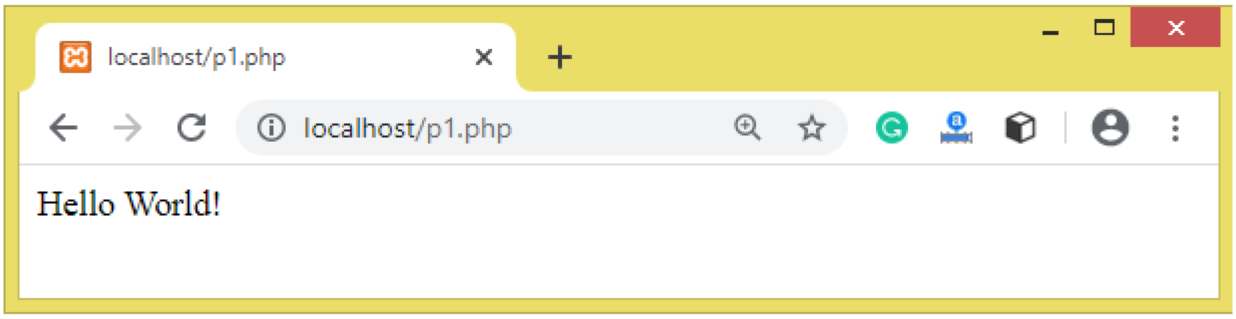
Step 2: Save the file with **hello.php** name in the htdocs folder, which resides inside the xampp folder.

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:



```
Hello World!
```

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

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!";
?>
```

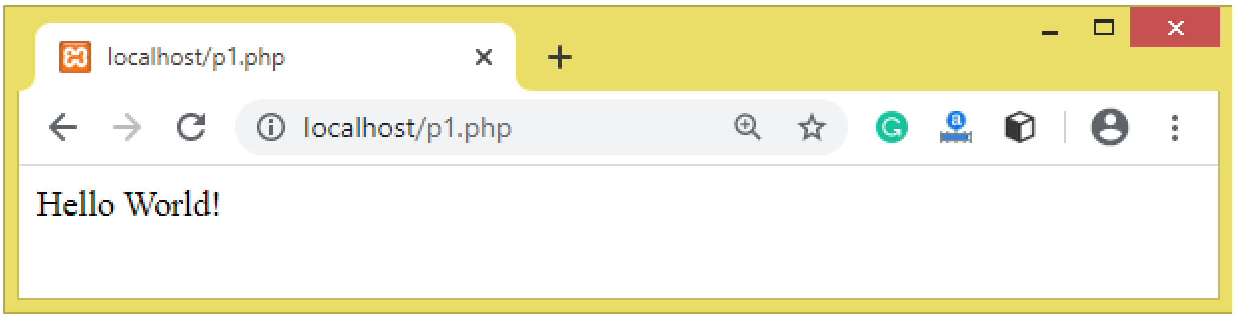
Step 2: Save the file with **hello.php** name in the htdocs folder, which resides inside the xampp folder.

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:



```
Hello World!
```

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

- *PHP tags: One must use the PHP standard tags, rather than the shorthand tags to delimit the PHP code.*

Full tags :

```
<?PHP
   PHP  code
?>
```

Short tags:

```
<?
   PHP  code
?>
```

Basic Syntax

The structure which defines PHP computer language is called **PHP syntax**.

The PHP script is executed on the server and the HTML result is sent to the browser. It can normally have HTML and PHP tags. PHP or Hypertext Preprocessor is a widely used open-source general-purpose scripting language and can be embedded with HTML. PHP files are saved with the ".php" extension. PHP scripts can be written anywhere in the document within PHP tags along with normal HTML.

Escaping To PHP:

Writing the PHP code inside *<?php ....?>* is called **Escaping to PHP**.

The mechanism of separating a normal HTML from PHP code is called the mechanism of Escaping To PHP. There are various ways in which this can be done. Few methods are already set by default but in order to use few others like Short-open or ASP-style tags, we need to change the configuration of the php.ini file. These tags are also used for embedding PHP within HTML. There are 4 such tags available for this purpose.

Canonical PHP Tags: The script starts with **<?php** and ends with **?>**. These tags are also called 'Canonical PHP tags'. Everything outside of a pair of opening and closing tags is ignored by the PHP parser. The open and closing tags are called delimiters. Every PHP command ends with a semi-colon (;). Let's look at the *hello world* program in PHP.

A PHP parser is a program that takes PHP code as input and produces an abstract syntax tree (AST) as output. The AST is a data structure that represents the structure of the PHP code. It can be used for static code analysis, code manipulation, and code generation.

- PHP

```
<?php
# Here echo command is used to
print
echo "Hello, world!";
?>
```

SGML or Short HTML Tags: These are the shortest option to initialize a PHP code. The script starts with **<?** and ends with **?>**. This will only work by setting the *short_open_tag* setting in the *php.ini* file to 'on'.

Example:

- PHP

```
<?
# Here echo command will only
work if
# setting is done as said
before
echo "Hello, world!";
?>
```

ASP Style Tags: To use this we need to set the configuration of the *php.ini* file. These are used by Active Server Pages to describe code blocks. These tags start with **<%** and end with **%>**.

Example:

- PHP

```
<%
# Can only be written if setting is
turned on
# to allow %
echo "hello world";
%>
```

PHP Echo & print:

Both are [language constructs](#) in PHP programs that are more or less the same as both are used to output data on the browser screen. The print *statement is an alternative to* echo.

PHP Echo:

It is a language construct and never behaves like a function, hence no parenthesis is required. But the developer can use parenthesis if they want. The end of the *echo* statement is identified by the semi-colon (';'). It output one or more strings. We can use '*echo*' to output strings, escaping characters, numbers, variables, values, and results of expressions.

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.

Displaying Strings: *We can simply use the keyword echo followed by the string to be displayed within quotes. The below example shows how to display strings with PHP.*

- *PHP*

```
<?php
    echo "Hello,This is a display string
example!";
?>
```

Output:

```
Hello,This is a display string example!
```

Displaying Strings as multiple arguments: *We can pass multiple string arguments to the echo statement instead of a single string argument, separating them by comma (‘,’) operator. For example, if we have two strings i.e "Hello" and "World" then we can pass them as ("Hello", "World").*

- *PHP*

```
<?php
    echo "Multiple ","argument
","string!";
?>
```

Output:

```
Multiple argument string!
```

Displaying Variables: *The below example shows different ways to display variables with the help of a PHP echo statement.*

- *PHP*

```
<?php
  // Defining the
variables
  $text = "Hello,
World!";
  $num1 = 10;
  $num2 = 20;

  // Echoing the
variables
  echo $text."\n";
  echo
$num1."+".$num2."=";
  echo $num1 + $num2;
?>
```

Output:

```
Hello, World!
10+20=30
```

The (.) operator in the above code can be used to concatenate two strings in PHP and the "\n" is used for a new line and is also known as line-break.

PHP echo: printing escaping characters

File: echo3.php

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

Output:

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

PHP echo: printing variable value

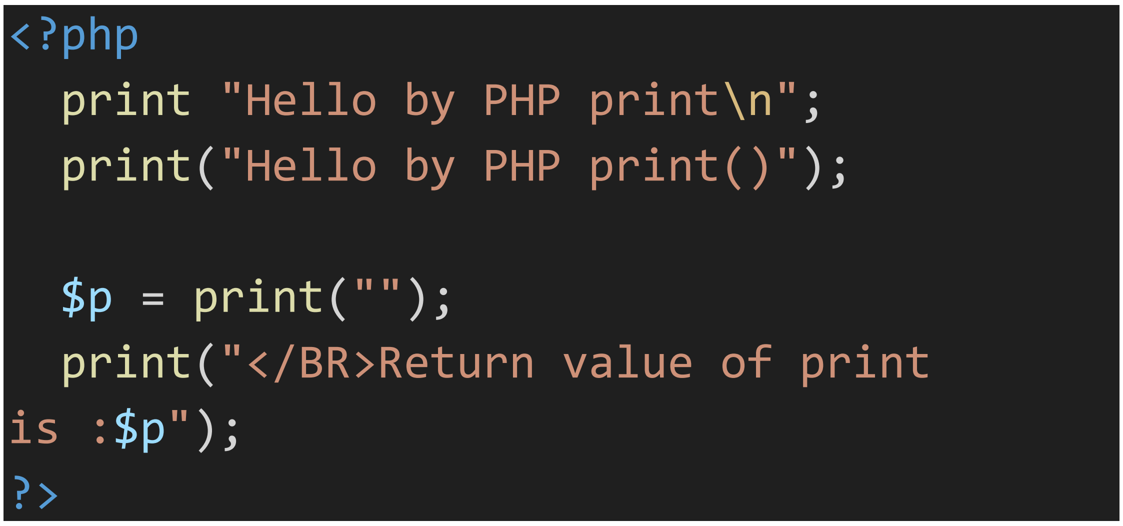
PHP print:

The PHP print statement is similar to the echo statement and can be used alternative to echo many times. 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. *Like an echo, the print statement can also be used to print strings and variables.*

- Using print, we cannot pass multiple arguments. *we cannot display multiple strings separated by comma(,) with a single print statement.*
- print is slower than the echo statement.

PHP print: printing string

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



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

  $p = print("");
  print("</BR>Return value of print
is :$p");
?>
```

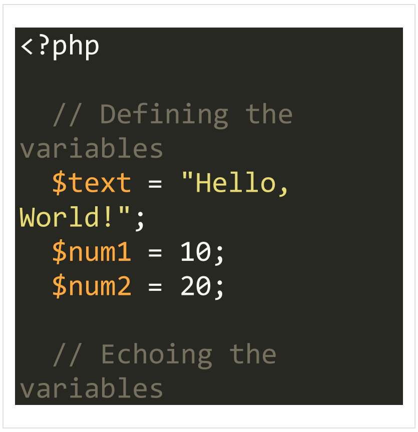
PHP print: printing escaping characters

File: print3.php

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

PHP print: printing variable value

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



```
<?php

  // Defining the
variables
  $text = "Hello,
World!";
  $num1 = 10;
  $num2 = 20;

  // Echoing the
variables
```

```
  print $text."\n";
  print
$num1."+".$num2."=";
  print $num1 + $num2;
?>
```

Output:

```
Hello, World!
10+20=30
```

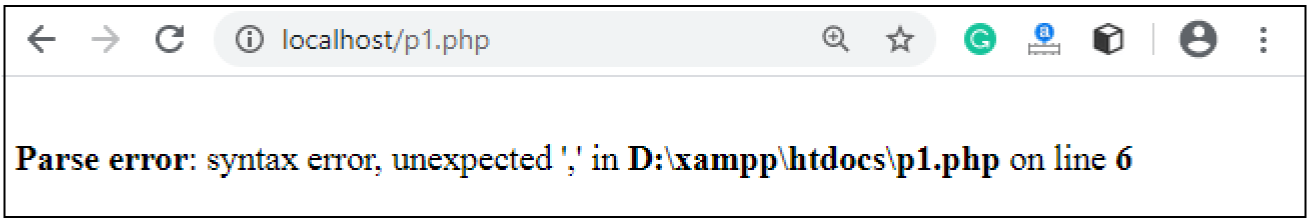
5.

- Using print, we cannot pass multiple arguments.*we cannot display multiple strings separated by comma(,) with a single print statement.*

6.

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

Output:



```
Parse error: syntax error, unexpected ',' in D:\xampp\htdocs\p1.php on line 6
```

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

Difference between echo and print statements in PHP:

S. No.	echo statement	print statement
1.	echo accepts a list of arguments (multiple arguments can be passed), separated by commas.	print accepts only one argument at a time.
2.	It returns no value or returns void.	It returns the value 1.
3.	It displays the outputs of one or more strings separated by commas.	The print outputs only the strings.

S.No.	echo statement	print statement
4.	It is comparatively faster than the print statement.	It is slower than an echo statement.

Variable in PHP:

Variables in a program are used to store some values or data that can be used later in a program. The variables are also like containers that store character values, numeric values, memory addresses, and strings. PHP has its own way of declaring and storing variables.

In PHP, a variable is declared using $ sign, followed by the variable name.

```
  $n = 10;
```

There are a few rules, that need to be followed and facts that need to be kept in mind while dealing with variables in PHP:

- Any variables declared in PHP must begin with a dollar sign ($), followed by the variable name.
- A variable can have long descriptive names (like $factorial, $even_nos) or short names (like $n or $f or $x)
- A variable name can only contain alphanumeric characters and underscores (i.e., 'a-z', 'A-Z', '0-9, and '_') in their name. Even it cannot start with a number.
- A constant is used as a variable for a simple value that cannot be changed. It is also case-sensitive.
    - A PHP variable name cannot contain spaces.

- Assignment of variables is done with the assignment operator, "equal to (=)". The variable names are on the left of equal and the expression or values are to the right of the assignment operator '='.
- One must keep in mind that variable names in PHP names must start with a letter or underscore and no numbers.
-
- PHP is a loosely typed language, and we do not require to declare the data types of variables, rather PHP assumes it automatically by analyzing the values. The same happens while conversion. No variables are declared before they are used. It automatically converts types from one type to another whenever required.
- PHP variables are case-sensitive, i.e., $sum and $SUM are treated differently.

Data types used by PHP to declare or construct variables:

- Integers
- Doubles

- NULL
- Strings
- Booleans
- Arrays
- Objects
- Resources

Valid & invalid naming convention of variable:

```
<?php

// These are all valid declarations
$val = 5;
$val2 = 2;
$x_Y = "gfg";
$_X = "GeeksforGeeks";

// This is an invalid declaration as it
// begins with a number
$10_ val = 56;

// This is also invalid as it contains
// special character other than _
$f.d = "num";

?>
```

PHP Variable: case sensitive:

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc. Although   keywords in PHP are not case sensitive.

```
<?php
  echo "<H1><U>Variable Case Sensivity in 
PHP</U></H1>";

  $num = 10;
  $color = "Red";
  echo "Car is $color</BR>";
  echo "Boat is $COLOR</BR>";
  echo "House is $CoLor</BR>";
 /*
  Warning: Undefined variable $COLOR in
E:\XAMPP\htdocs\My_PHP\3.variable_case_se
nsivity.php on line 8
  Boat is
  Warning: Undefined variable $CoLor in
E:\XAMPP\htdocs\My_PHP\3.variable_case_se
nsivity.php on line 9
  House is
 */
?>
```

Variable Variables:-

- *PHP allows us to use dynamic variable names, called variable variables.*
- *Variable variables are simply variables whose names are dynamically created by another variable's value.*

PHP $ and $$ Variables

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

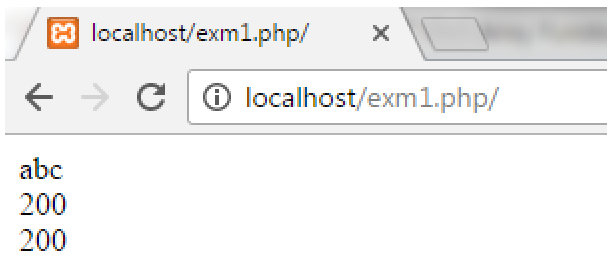
The **$$var** (double dollar) is a reference variable that stores the value of the $variable inside it.

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;
?>
```



```
abc
200
200
```

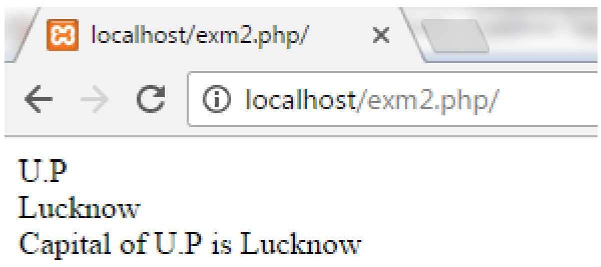
In the above example, we have assigned a value to the variable **x** as **abc**. Value of reference variable **$$x** is assigned as **200**.

Now we have printed the values **$x, $$x** and **$abc**.

Example 2

```
<?php
 $x="U.P";
$$x="Lucknow";
echo $x. "<br>";
echo $$x. "<br>";
echo "Capital of $x is " . $$x;
?>
```

Output:



```
U.P
Lucknow
Capital of U.P is Lucknow
```

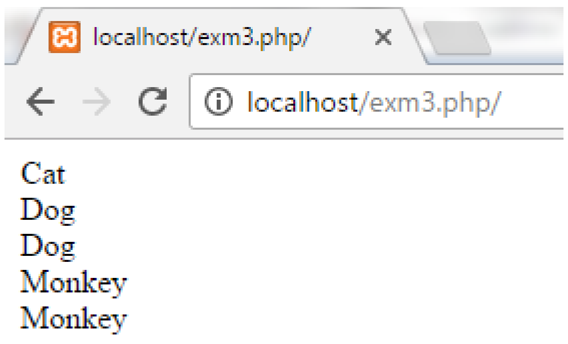
In the above example, we have assigned a value to the variable **x** as **U.P**. Value of reference variable **$$x** is assigned as **Lucknow.**

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

Example 3

```
<?php
$name="Cat";
${$name}="Dog";
${${$name}}="Monkey";
echo $name. "<br>";
echo ${$name}. "<br>";
echo $Cat. "<br>";
echo ${${$name}}. "<br>";
echo $Dog. "<br>";
?>
```

Output:



```
Cat
Dog
Dog
Monkey
Monkey
```

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.**

```
$$a = 'World'; //$($a) is equals to $(hello)
```

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. *It should not contain any special characters except underscore, as mentioned.*

Conventionally, PHP constants should be defined in uppercase letters.

By default, a constant is always case-sensitive, unless mentioned.

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

PHP constant: define()

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

define(name, value, case-insensitive)

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

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

```
<?php
  echo "<H1><U>Constant in PHP</U></H1>";
  define("OMG","Oh! My God");
  echo "OMG: ",OMG;
?>
```

Output:

```
OMG: Oh! My God
```

case-insensitive: Specifies whether a constant is **case-insensitive.** Default value is false. It means it is case sensitive by default.

```
<?php
   // This creates a case-sensitive
constant
   define("WELCOME", "GeeksforGeeks");
//Default false
   echo WELCOME, "\n";
   // This creates a case-insensitive
constant
   define("HELLO", "GeeksforGeeks", true);
   echo hello;
?>
```

PHP constant: const keyword

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

```
<?php
  echo "<H1><U>Constant using 'const' keyword</H1></U></BR>";
  const OMG = "Oh! My God";
  echo "OMG: ",OMG;
?>
```

OUTPUT:

```
Constant using 'const' keyword
OMG: Oh! My God
```

Constants are Global: *By default, constants are automatically* ***global***, *and can be used throughout the script, accessible inside and outside of any function.*
Example:

```php
<?php

define("WELCOME",
"GeeksforGeeks");

function testGlobal() {
    echo WELCOME;
}

testGlobal();

?>
```

constant() function

Instead of using the echo statement ,there is an another way to print constants using the constant() function.

Syntax

Example:

```php
<?php

   define("WELCOME",
"GeeksforGeeks!!!");

   echo WELCOME, "\n";

   echo constant("WELCOME");
   // same as previous

?>
```

Output:

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.

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 constant in the PHP and all of the constant resolved at the compile-time, not like the regular constant which is resolved at run time.

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

Magic constants are case-insensitive.

All the constants are listed below with the example code:

1. __LINE__
2. __FILE__
3. __DIR__
4. __FUNCTION__
5. __CLASS__
6. __TRAIT__
7. __METHOD__
8. __NAMESPACE__

9. ClassName::class

1. __LINE__

It returns the current line number of the file, where this constant is used.

If you use this magic constant in your program file somewhere then this constant will display the line number during compile time.

Example:

```
<?php
  echo "<H1><U>Magic Constants in PHP</H1></U></BR>";
  echo "You are at line Number: ",__LINE__;
?>
```

```
You are at line number: 3
```

2. __FILE__:

This magic constant returns the full path of the executed file, where the file is stored. If it is used inside the include, the name of the included file is returned.

Example:

```
<?php
  echo "</BR>Executed File Name is : ",__FILE__;
?>
```

Output:

```
Executed File Name is : E:\XAMPP\htdocs\My_PHP\Constant\12.magic_constant.php
```

3. __DIR__:

It returns the full directory path of the executed file.

The path returned by this magic constant is equivalent to dirname(__FILE__). This magic constant does not have a trailing slash unless it is a root directory.

Example:

```
<?php

  echo "</BR>Full directory path of executed file: ",__DIR__;

  /*The path returned by this magic constant is equivalent

   to dirname(__FILE__).

   */

  echo "</BR>dirname(__FILE__): ",dirname(__FILE__);

?>
```

Output:

```
Full directory path of executed file: E:\XAMPP\htdocs\My_PHP\Constant
dirname(__FILE__): E:\XAMPP\htdocs\My_PHP\Constant
```

4. __FUNCTION__:

This magic constant returns the function name, where this constant is used. It will return blank if it is used outside of any function.

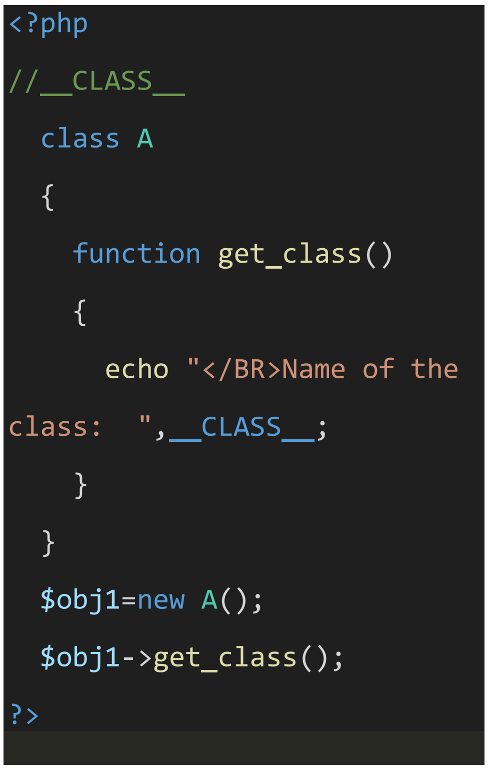
Example:

```
<?php


  function display()

  {

    echo "Function name is: ",__FUNCTION__;

  }

  display();

  echo "</BR>Function name is: ",__FUNCTION__;

?>
```

5. __CLASS__:

It returns the class name, where this magic constant is included. __CLASS__ constant also works in traits.

Example:



```
<?php
//__CLASS__
  class A
  {
    function get_class()
    {
      echo "</BR>Name of the
class:  ",__CLASS__;
    }
  }
  $obj1=new A();
  $obj1->get_class();
?>
```

***Output*:**

Name of the class: A

Example:

```
<?php
  class base
  {
  function test_first(){
      //will always print parent class which is base here.
      echo __CLASS__;
    }
  }
  class child extends base
  {
    public function __construct()
```

```
                {
                    ;
                }
            }
            $t = new child;
            $t->test_first();
        ?>
```

Output:

```
JTP

base
```

6. __TRAIT__:

This magic constant returns the trait name, where it is included.

Example:

```php
<?php
  trait created_trait
  {
    function jtp()
    {
      echo "Name of trait: ",__TRAIT__;
    }
  }
  class Company
  {
    use created_trait;
  }
  $a=new Company;
  $a->jtp();
```

```
?>
```

Output:

```
created_trait
```

7. __METHOD__:

It returns the name of the class method where this magic constant is included. The method name is returned the same as it was declared.

Example:

```php
<?php
  class A
  {
    public function __construct()
    {
      echo __METHOD__,"</BR>";
    }
    public function my_method()
    {
      echo __METHOD__;
    }
  }
  $obj1 = new A();
  $obj1->my_method();
?>
```

Output:

```
A::__construct
A::my_method
```

8. __NAMESPACE__:

It returns the current namespace where it is included.

Example:

```php
<?php
  namespace GeeksforGeeks;
  class Company
  {
    public function gfg()
    {
      return __namespace__;
    }
  }
  $obj = new Company();
  echo $obj->gfg();
?>
```

Output:

```
GeeksforGeeks
```

- Another Example:

```php
<?php
  echo "<h3>Example for __NAMESPACE__</h3>";
  class name {
    public function __construct() {
      echo 'This line will print on calling namespace.';
    }
  }
  $class_name = __NAMESPACE__ . '\name';
  $a = new class_name;
?>
```

Output:

```
Example for __NAMESPACE__

This line will print on calling namespace.
```

9. ClassName::class:

It returns the fully qualified name of the ClassName. ClassName::class is added in **PHP 5.5.0**. It is useful with namespaced classes.

This magic constant does not start and end with the double underscore (__).

Example:

```php
<?php
  namespace Computer_Sciecnec_Portal;
  class Geeks
  {
  }
  echo Geeks::class; //Classname::class
?>
```

Output:

```
Computer_Sciecnec_Portal\Geeks
```

Note: Remember namespace must be the very first statement or after any declare call in the script, otherwise it will generate Fatal error.

PHP Variable Scope

The scope of a variable means the area in the code where the variable is accessible or visible . Our PHP code consists of class, function, structure etc. So it is importance to understand the scope of global and local variable

PHP has three types of variable scopes:

Local variable:

Local variabale is declare inside a function. These local variable can only accessed within that function. Local variables can not be accessed outside the function.

Any declaration of a variable outside the function with the same name as that of the one within the function is a completely different variable.

Example:

```
<?php
  $n=10;
  function my_fun()
  {
    $n=20;
    echo "Inside my_fun(): \$n = $n</BR>";
  }
  my_fun();
  echo "Outside the function: \$n = $n";
?>
```

Output:

```
Inside my_fun(): $n = 20
Outside the function: $n = 10
```

Global variable:

Global variable are declared outside the function. Global variables can access anywhere in a program.
PHP provides "global" keyword to access global variable inside the function.

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.

As we can access the global variable inside the function , so if we update the value of this global variable via "global" keyword, the change also reflected outside the function.

USE OF $GLOBALS[INDEX]   :

All the global variable in PHP are stored in arrray named **$GLOBALS[INDEX]** . Where the index is the name of the variable. (Do nort put $ sign).

Example:

```php
<?php
  $n = 10; //global variable
  echo "\$n = ",$n,"</BR>";
  function display()
  {
    echo "Inside display() function.</BR>";
    $n1=20;
    echo "\$n1 =",$n1,"</BR>";
    global $n;
    echo "\$n(Global variable) = ",$n,"</BR>";
    $n=400;
  }
  display();
  echo "Outside the display() function</BR>";
  echo "\$n = ",$n;
  echo "</BR>var_dump(\$GLOBALS) : ",var_dump($GLOBALS);
  echo "</BR>var_dump(\$GLOBALS['n']): ",var_dump($GLOBALS['n']);
?>
```

OUTPUT:

```
$n = 10
Inside display() function.
$n1 =20
$n(Global variable) = 10
Outside the display() function
$n = 400
var_dump($GLOBALS) : array(5) { ["_GET"]=> array(0) { } ["_POST"]=>
array(0) { } ["_COOKIE"]=> array(0) { } ["_FILES"]=> array(0) { }
["n"]=> int(400) }
var_dump($GLOBALS['n']): int(400)
```

Static variable:

It is the characteristic of PHP to delete the variable, once it completes its execution and the memory is freed. But sometimes we need to store the variables even after the completion of function execution. To do this we use the static keywords and the variables are then called static variables. PHP associates a data type depending on the value for the variable.

Example:

```
<?php
  function my_fun()
  {
    static $i = 0;
    $i++;
    echo "\$i = $i</BR>";
  }
  my_fun();
  my_fun();
  my_fun();
?>
```

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

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.

Example:

```
<?php
   $x = 20; // Variable 1
   $y = 4; // Variable 2

   echo($x + $y), "</BR>";
   echo($x - $y), "</BR>";
   echo($x * $y), "</BR>";
   echo($x / $y), "</BR>";
   echo($x % $y), "</BR>";
   echo($x ** $y), "</BR>";
?>
```

OUTPUT:

```
24
16
80
5
0
160000
```

Comparison Operators: *These operators are used to compare two elements and outputs the result in boolean form. Here are the comparison operators along with their syntax and operations in PHP.*

Operator	Name	Syntax	Operation
==	Equal To	$x == $y	Returns True if both the operands are equal
!=	Not Equal To	$x != $y	Returns True if both the operands are not equal
<>	Not Equal To	$x <> $y	Returns True if both the operands are unequal
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$x === $y	Returns True if both the operands are equal and are of the same type
!==	Not Identical	$x == $y	Returns True if both the operands are unequal and are of different types
<	Less Than	$x < $y	Returns True if $x is less than $y
>	Greater Than	$x > $y	Returns True if $x is greater than $y
<=	Less Than or	$x <=	Returns True if $x is less

Operator	Name	Syntax	Operation
	Equal To	$y	than or equal to $y
>=	Greater Than or Equal To	$x >= $y	Returns True if $x is greater than or equal to $y
<=>	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

***Example**: This example describes the comparison operator in PHP.*

- *PHP*

```
<?php
  $a = 80;
  $b = 50;
  $c = "80";

  // Here var_dump function has been used to
  // display structured information. We will
learn
  // about this function in complete details
in further
  // articles.
// the var_dump() function to display the value of
the expression $a1 != $b1.

  var_dump($a == $c) + "\n";
  var_dump($a != $b) + "\n";
  var_dump($a <> $b) + "\n";
  var_dump($a === $c) + "\n";
  var_dump($a !== $c) + "\n";
  var_dump($a < $b) + "\n";
  var_dump($a > $b) + "\n";
  var_dump($a <= $b) + "\n";
  var_dump($a >= $b);
?>
```

***Output**:*

```
bool(true)

bool(true)

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

```
<?php
  $a1 = 30;
  $a2 = 30;

  $b1 = 21;
  $b2 = "21";
  $f1 = 4.6;

  $r = $a1 == $a1;
  echo "\$a1 != \$a1: ",var_dump($r),"<BR>";
// the var_dump() function to display the value of the
expression $a1 != $a1.
```

```
  $r = $b1 == $b2;
  echo "\$b1 != \$b2: ",var_dump($r),"<BR>";
```

```
  $r = $b1===$b2;
  echo "\$b1===\$b2: ",var_dump($r),"<BR";
?>
```

<u>Output:</u>

```
$a1 != $a1: bool(true)
$b1 != $b2: bool(true)
$b1===$b2: bool(false)
```

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.

These are basically used to operate with conditional statements and expressions. Conditional statements are based on conditions. Also, a condition can either be met or cannot be met so the result of a conditional statement can either be true or false. Here are the logical operators along with their syntax and operations in PHP.

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 $a 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

The or operator returns true if either of its operands is true. For example, the following code will return true:

PHP

```
$a = true;$b = false;
echo $a or $b; // true
```

The xor operator, on the other hand, returns true only if **one** of its operands is true. If both operands are true or both operands are false, the xor operator will return false. For example, the following code will return true:

PHP

$a = true;$b = true;

echo $a xor $b; // false

Conditional or Ternary Operators:

These operators are used to compare two values and take either of the results simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as a shorthand notation for *if...else* statement that we will read in the article on decision making.

Syntax:

```
$var = (condition)? value1 : value2;
```

Here, the condition will either evaluate as true or false. If the condition evaluates to True, then value1 will be assigned to the variable $var otherwise value2 will be assigned to it.

Operator	Name	Operation
?:	Ternary	If the condition is true? then $x : or else $y. This means that if the condition is true then the left result of the colon is accepted otherwise the result is on right.

Example: This example describes the Conditional or Ternary operators in PHP.

- PHP

```
<?php
  $x = -12;
  echo ($x > 0) ? 'The number is positive' : 'The number
is negative';
?>
```

Output:

```
The number is negative
```

Assignment Operators: These operators are used to assign values to different variables, with or without mid-operations. Here are the assignment operators along with their syntax and operations, that PHP provides for the operations.

Operator	Name	Syntax	Operation
=	Assign	$x = $y	Operand on the left obtains the value of the operand on the right
+=	Add then Assign	$x += $y	Simple Addition same as $x = $x + $y
-=	Subtract then Assign	$x -= $y	Simple subtraction same as $x = $x – $y
*=	Multiply then Assign	$x *= $y	Simple product same as $x = $x * $y
/=	Divide then Assign (quotient)	$x /= $y	Simple division same as $x = $x / $y
%=	Divide then Assign (remainder)	$x %= $y	Simple division same as $x = $x % $y

Example: *This example describes the assignment operator in PHP.*

- PHP

```
<?php
  // Simple assign operator
  $y = 75;
  echo $y, "\n";

  // Add then assign operator
  $y = 100;
  $y += 200;
  echo $y, "\n";

  // Subtract then assign operator
  $y = 70;
  $y -= 10;
  echo $y, "\n";

  // Multiply then assign operator
  $y = 30;
  $y *= 20;
  echo $y, "\n";

  // Divide then assign(quotient)
operator
```

```
  $y = 100;
  $y /= 5;
  echo $y, "\n";

  // Divide then assign(remainder)
operator
  $y = 50;
  $y %= 5;
  echo $y;
?>
```

Output:

```
75
300
60
600
20
0
```

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

Let’s look at the truth table of the bitwise operators.

X	Y	X & Y	X | Y	X ^ Y
0	**0**	0	0	0
0	**1**	0	1	1

X	Y	X & Y	X | Y	X ^ Y
1	0	0	1	1
1	1	1	1	0

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

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

Example: *This example describes the string operator in PHP.*

- *PHP*

```
<?php
  $x = "Geeks";
  $y = "for";
  $z = "Geeks!!!";
  echo $x . $y . $z,
"\n";
```

```
  $x .= $y . $z;
  echo $x;
?>
```

Output:

```
GeeksforGeeks!!!
GeeksforGeeks!!!
```

Array Operators: *These operators are used in the case of arrays. Here are the array operators along with their syntax and operations, that PHP provides for the array operation.* Basically, these operators are used to compare the values of arrays.

Operator	Name	Syntax	Operation
+	Union	$x + $y	Union of both i.e., $x and $y
==	Equality	$x == $y	Returns true if both has same key-value pair
!=	Inequality	$x != $y	Returns True if both are unequal
===	Identity	$x === $y	Returns True if both have the same key-value pair in the same order and of the same type
!==	Non-Identity	$x !== $y	Returns True if both are not identical to each other
<>	Inequality	$x <> $y	Returns True if both are unequal.Return TRUE if $a is not equal to $b

Example: *This example describes the array operation in PHP.*

- *PHP*

```
<?php
```

```
  $x = array("k" => "Car", "l" =>
"Bike");
  $y = array("a" => "Train", "b" =>
"Plane");

  var_dump($x + $y);
  var_dump($x == $y) + "\n";
  var_dump($x != $y) + "\n";
  var_dump($x <> $y) + "\n";
  var_dump($x === $y) + "\n";
  var_dump($x !== $y) + "\n";
?>
```

***Output*:**

```
array(4) {
  ["k"]=>
  string(3) "Car"
  ["l"]=>
  string(4) "Bike"
  ["a"]=>
  string(5) "Train"
  ["b"]=>
  string(5) "Plane"
}
bool(false)
bool(true)
bool(true)
bool(false)
bool(true)
```

) and return a false value if both of the values are not same.
This should always be kept in mind that the present equality operator == is different from the assignment operator =. The assignment operator changes and assigns the variable on the left to have a new value as the variable on right, while the ***equal operator == tests for equality*** *and returns true or false as per the comparison results.*
Example:

- *php*

```
<?php

// Variable contains integer
value
$x = 999;

// Variable contains string
value
$y = '999';

// Compare $x and $y
if ($x == $y)
    echo 'Same content';
else
    echo 'Different
content';
?>
```

Output:

```
Same content
```

Identical Operator ===

The comparison operator called as the Identical operator is the triple equal sign "===". This operator allows for a ***much stricter comparison between the given variables or values.***
This operator returns true if both variable contains ***same information and same data types*** *otherwise return false.*
Example:

- *php*

```
<?php

// Variable contains integer value
$x = 999;

// Variable contains string value
$y = '999';

// Compare $x and $y
if ($x === $y)
    echo 'Data type and value both
are same';
else
    echo 'Data type or value are
different';
```

```
?>
```

Output:

```
Data type or value are different
```

In the above example, value of $x and $y are equal but data types are different so else part will execute.

Type Operators

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

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

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

Output:

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

Execution Operators

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

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

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

Error Control Operators

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

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

Spaceship Operators:

PHP 7 has introduced a new kind of operator called spaceship operator. The spaceship operator or combined comparison operator is denoted by "<=>". These operators are used to compare values but instead of returning the boolean results, it returns integer values. If both the operands are equal, it returns 0. If the right operand is greater, it returns -1. If the left operand is greater, it returns 1. The following table shows how it works in detail:

Operator	Syntax	Operation
$x < $y	$x <=> $y	Identical to -1 (right is greater)
$x > $y	$x <=> $y	Identical to 1 (left is greater)

Operator	Syntax	Operation
$x <= $y	$x <=> $y	Identical to -1 (right is greater) or identical to 0 (if both are equal)
$x >= $y	$x <=> $y	Identical to 1 (if left is greater) or identical to 0 (if both are equal)
$x == $y	$x <=> $y	Identical to 0 (both are equal)
$x != $y	$x <=> $y	Not Identical to 0

Example: *This example illustrates the use of the spaceship operator in PHP.*

- *PHP*

```
<?php
  $x = 50;
  $y = 50;
  $z = 25;

  echo $x <=> $y;
  echo "\n";

  echo $x <=> $z;
  echo "\n";

  echo $z <=> $y;
  echo "\n";

  // We can do the same for
Strings
  $x = "Ram";
  $y = "Krishna";

  echo $x <=> $y;
  echo "\n";

  echo $x <=> $y;
  echo "\n";

  echo $y <=> $x;
?>
```

Output:

```
0
```

```
1
-1
1
1
-1
```

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

String:

A list of PHP string functions are given below.

4. strpos() function: This function takes two string arguments and if the second string is present in the first one, it will return the starting position of the string otherwise returns FALSE.

Example:

- PHP

```php
<?php

echo strpos("Hello GeeksforGeeks!",
"Geeks"), "\n";

echo strpos("Hello GeeksforGeeks!",
"for"), "\n";

var_dump(strpos("Hello GeeksforGeeks!",
"Peek"));

?>
```

Output:

```
6
11
bool(false)
```

5. trim() function: *This function allows us to remove whitespaces or strings from both sides of a string.*

Example:

- *PHP*

```php
<?php

echo trim("Hello World!",
"Hed!");

?>
```

Output:

```
llo Worl
```

6. explode() function: *This function converts a string into an array.*

Example:

- *PHP*

```php
<?php


$input  = "Welcome to
geeksforgeeks";

print_r(explode(" ",$input));

?>
```

Output:

```
Array ( [0] => Welcome [1] => to [2] => geeksforgeeks )
```

```php
//explode()  String to array converter
  $s1="Raj Joy Sam Mahi";
  $a=explode(" ",$s1);
  var_dump($a);
  echo gettype($a)."<br>";

  //implode()   Array to string converter
  $new=implode($a);
  echo gettype($new),"<br>";
  echo "\$new: $new<br>";
```

```
  $new=implode(" ",$a);
  echo "\$new: $new<br>";
```

7. strtolower() function: This function converts a string into the lowercase string.

Example:

- PHP

```
<?php

$input  = "WELCOME TO
GEEKSFORGEEKS";

echo strtolower($input);

?>
```

Output:

```
welcome to geeksforgeeks
```

8. strtoupper() function: This function converts a string into the uppercase string.

Example:

- PHP

```
<?php

$input  = "Welcome to
geeksforgeeks";
```

```php
echo strtoupper($input);

?>
```

Output:

```
WELCOME TO GEEKSFORGEEKS
```

9. strwordcount() function: *This function counts total words in a string.*

Example:

- *PHP*

```php
<?php

$input  = "Welcome to
GeeksforGeeks";

echo str_word_count($input);

?>
```

Output:

```
3
```

10. substr() function: *This function gives the substring of a given string from a given index.*

Example:

- *PHP*

```php
<?php

$input  = "Welcome to
geeksforgeeks";

echo(substr($input,3));

?>
```

Output:

```
come to geeksforgeeks
```

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 Function Examples

1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax

```
string strtolower ( string $string )
```

Example

```php
<?php
$str="My name is KHAN";
$str=strtolower($str);
echo $str;
?>
```

Output:

```
my name is khan
```

2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

Syntax

```
string strtoupper ( string $string )
```

Example

```
<?php
$str="My name is KHAN";
$str=strtoupper($str);
echo $str;
?>
```

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

```
string ucfirst ( string $str )
```

Example

```
<?php
$str="my name is KHAN";
$str=ucfirst($str);
echo $str;
?>
```

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

```
string lcfirst ( string $str )
```

Example

```
<?php
$str="MY name IS KHAN";
$str=lcfirst($str);
echo $str;
?>
```

Output:

```
mY name IS KHAN
```

5) PHP ucwords() function

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

Syntax

```
string ucwords ( string $str )
```

Example

```
<?php
$str="my name is Sonoo jaiswal";
$str=ucwords($str);
echo $str;
?>
```

Output:

```
My Name Is Sonoo Jaiswal
```

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.

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key. Suppose we want to store five names and print them accordingly. This can be easily done by the use of five different string variables. But if instead of five, the number rises to a hundred, then it would be really difficult for the user or developer to create so many different variables. Here array comes into play and helps us to store every element within a single variable and also allows easy access using an index or a key. An array is created using an **array()** function in PHP.

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.

There are basically three types of arrays in PHP:

- **Indexed or Numeric Arrays:** An array with a numeric index where values are stored linearly.
- **Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.

Multidimensional Arrays: An array which contains single or multiple array within it and can be accessed via multiple indices.

Indexed or Numeric Arrays

These type of arrays can be used to store any type of elements, but an index is always a number. By default, the index starts at zero. These arrays can be created in two different ways as shown in the following example:

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:

- PHP

```php
<?php

// One way to create an indexed array
$name_one = array("Zack", "Anthony", "Ram", "Salim",
"Raghav");

// Accessing the elements directly
echo "Accessing the 1st array elements directly:\n";
echo $name_one[2], "\n";
echo $name_one[0], "\n";
echo $name_one[4], "\n";

// Second way to create an indexed array
$name_two[0] = "ZACK";
$name_two[1] = "ANTHONY";
$name_two[2] = "RAM";
$name_two[3] = "SALIM";
$name_two[4] = "RAGHAV";

// Accessing the elements directly
echo "Accessing the 2nd array elements directly:\n";
echo $name_two[2], "\n";
echo $name_two[0], "\n";
echo $name_two[4], "\n";
```

```
?>
```

Output:

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 Form Handling

We can create and use forms in PHP. $_GET and $_POST super global variable is used to get or collect form data from HTML form after submitting it.

PHP Get Form

Get request is the default form request. $_GET super global variable is used to get or collect form data from HTML form after submitting it. When 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.

Let's see a simple example to receive data from get request in PHP.

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 super global variable is used to get or collect form data from HTML form after submitting it. When the data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request. 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.

Let's see a simple example to receive data from post request in PHP.

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 $passwo

echo "Welcome: $name, your password is: $password";
?>
```

PHP Database Connectivity with MySql

What is MySQL?

MySQL is an open-source relational database management system (RDBMS). It is the most popular database system used with PHP. MySQL is developed, distributed, and supported by Oracle Corporation.

- The data in a MySQL database are stored in tables which consists of columns and rows.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use database system.It uses standard SQL
- MySQL compiles on a number of platforms.

How to connect PHP with MySQL Database?

PHP 5 and later can work with a MySQL database using:

1. MySQLi extension.
2. PDO (PHP Data Objects).

Note: Since PHP 5.5, **mysql_connect()** extension is *deprecated*. Now it is recommended to use one of the 2 alternatives.

Difference Between MySQLi and PDO

- PDO works on 12 different database systems, whereas MySQLi works only with MySQL databases.
- Both PDO and MySQLi are object-oriented, but MySQLi also offers a procedural API.
- If at some point of development phase, the user or the development team wants to change the database then it is easy to that in PDO than MySQLi as PDO supports 12 different database systems.He would have to only change the connection string and a few queries. With MySQLi,he will need to rewrite the entire code including the queries.

There are three ways of working with MySQl and PHP

1. MySQLi (object-oriented)
2. MySQLi (procedural)
3. PDO

Connecting to MySQL database using PHP

- **Using MySQLi object-oriented procedure**:
  We can use the MySQLi object-oriented procedure to establish a connection to MySQL database from a PHP script.

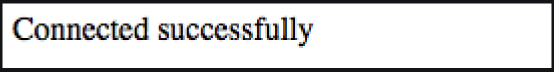
```php
<?php

//Using MYSQLi object oriented procedure
  $conn=new mysqli("localhost","root","");
  if ($conn->connect_error)
  {
    die("connection failed: ".$conn->connect_error);
  }
  echo "Connected Successfully";
  $conn->close();

?>
```

Note on the object-oriented example above:

$connect_error was broken until PHP 5.2.9 and 5.3.0. If you need to ensure compatibility with PHP versions prior to 5.2.9 and 5.3.0, use the following code instead:

```php
// Check connection
if (mysqli_connect_error()) {
  die("Database connection failed: " . mysqli_connect_error());
}
```

Output:



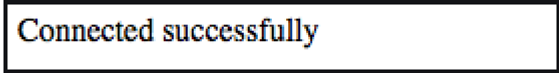
```
Connected successfully
```

Explanation: We can create an instance of the **mysqli** class providing all the necessary details required to establish the connection such as host, username, password etc. If the instance is created successfully then the connection is successful otherwise there is some error in establishing connection.

- **Using MySQLi procedural procedure** : There is also a procedural approach of MySQLi to establish a connection to MySQL database from a PHP script as described below.

```
<?php
 // Using MYSQLi procedural procedure
  $conn=mysqli_connect("localhost","root","","tanmay"); //tanmay ->
name of the database

  if (!$conn)   //mysqli_connect_error()
  {
    die("connection failed: ".$conn->mysqli_connect_error());
  }
  echo "Connected Successfully";
  mysqli_close($conn);
?>
```

Output:



```
Connected successfully
```

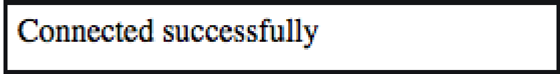
Explanation: In MySQLi procedural approach instead of creating an instance we can use the mysqli_connect() function available in PHP to establish a connection. This function takes the information as arguments such as host, username , password , database name etc. This function returns MySQL link identifier on successful connection or FALSE when failed to establish a connection.

- **Using PDO procedure**: PDO stands for PHP Data Objects. That is, in this method we connect to the database using data objects in PHP as described below:

  Syntax:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

Output:



```
Connected successfully
```

Explanation: The exception class in PDO is used to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Note: In the PDO example above we have also **specified a database (myDB)**. PDO require a valid database to connect to. If no database is specified, an exception is thrown.

Closing A Connection

When we establish a connection to MySQL database from a PHP script , we should also disconnect or close the connection when our work is finished. Here we have described the syntax of closing the connection to a MySQL database in all 3 methods described above. We have assumed that the reference to the connection is stored in $conn variable.

- **Using MySQLi object oriented procedure**
  Syntax

- **Using MySQLi procedural procedure**
  Syntax

- **Using PDO procedure**
  Syntax