

INTRODUCTION TO PHP

What is PHP

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

PHP was created by Rasmus Lerdorf in 1994 but appeared in the market in 1995. Some important points need to be noticed about PHP are as followed:

- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.
- PHP is simple and easy to learn language.

History

- PHP was developed by Rasmus Lerdorf in 1994 as a simple set of CGI binaries written in C. He called this suite of scripts "Personal Home Page Tools". It can be regarded as PHP version 1.0.
- In April 1996, Rasmus introduced PHP/FI. Included built-in support for DBM, mSQL, and Postgres95 databases, cookies, user-defined function support. PHP/FI was given the version 2.0 status.
- PHP: Hypertext Preprocessor – PHP 3.0 version came about when Zeev Suraski and Andi Gutmans rewrote the PHP parser and acquired the present-day acronym. It provided a mature interface for multiple databases, protocols and APIs, object-oriented programming support, and consistent language syntax.
- PHP 4.0 was released in May 2000 powered by Zend Engine. It had support for many web servers, HTTP sessions, output buffering, secure ways of handling user input and several new language constructs.
- PHP 5.0 was released in July 2004. It is mainly driven by its core, the Zend Engine 2.0 with a new object model and dozens of other new

features. PHP's development team includes dozens of developers and others working on PHP-related and supporting projects such as PEAR, PECL, and documentation.

- PHP 7.0 was released in Dec 2015. This was originally dubbed PHP next generation (phpng). Developers reworked Zend Engine is called Zend Engine 3. Some of the important features of PHP 7 include its improved performance, reduced memory usage, Return and Scalar Type Declarations and Anonymous Classes.
- PHP 8.0 was released on 26 November 2020. This is a major version having many significant improvements from its previous versions. One standout feature is Just-in-time compilation (JIT) that can provide substantial performance improvements. The latest version of PHP is 8.2.8, released on July 4th, 2023.

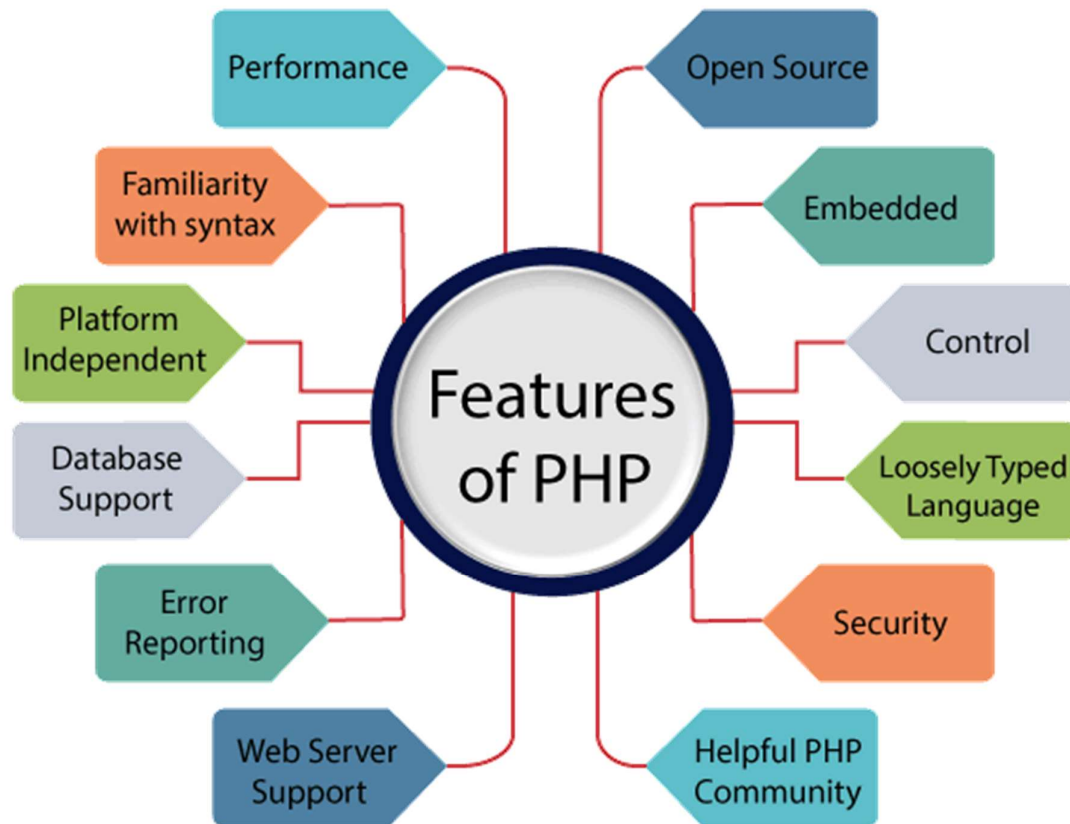
Why use PHP

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

- It handles dynamic content, database as well as session tracking for the website.
- You can create sessions in PHP.
- It can access cookies variable and also set cookies.
- It helps to encrypt the data and apply validation.
- PHP supports several protocols such as HTTP, POP3, SNMP, LDAP, IMAP, and many more.
- Using PHP language, you can control the user to access some pages of your website.
- As PHP is easy to install and set up, this is the main reason why PHP is the best language to learn.
- PHP can handle the forms, such as - collect the data from users using forms, save it into the database, and return useful information to the user. For example - Registration form.

PHP Features

PHP is very popular language because of its simplicity and open source. There are some important features of PHP given below:



Performance:

PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance.

Open Source: PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

Familiarity with syntax:

PHP has easily understandable syntax. Programmers are comfortable coding with it.

Embedded:

PHP code can be easily embedded within HTML tags and script.

Platform Independent:

PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

Database Support:

PHP supports all the leading databases such as MySQL, SQLite, ODBC, etc.

Error Reporting -

PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

Loosely Typed Language:

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

Web servers Support:

PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

Security:

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threats 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.

Install PHP

To install PHP, we will suggest you to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems. There are many AMP options available in the market that are given below:

- WAMP for Windows
- LAMP for Linux
- MAMP for Mac
- SAMP for Solaris
- FAMP for FreeBSD

XAMPP (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail, etc.

If you are on Windows and don't want Perl and other features of XAMPP, you should go for WAMP. In a similar way, you may use LAMP for Linux and MAMP for Macintosh.

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!";
?>
```

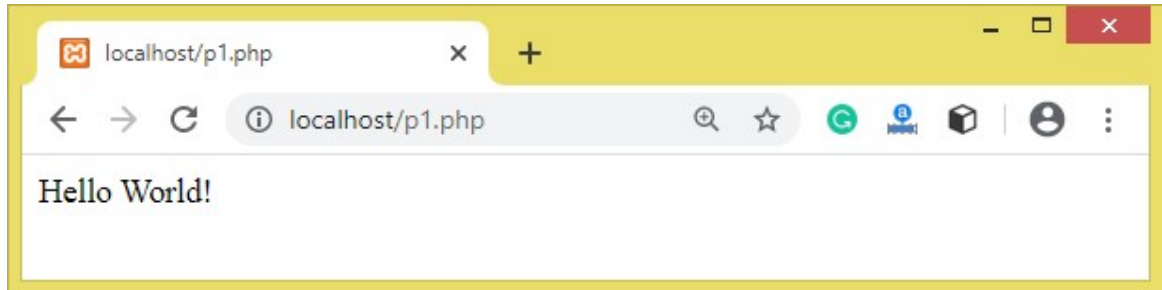
Step 2: Save the file with hello.php name in the htdocs folder, which resides inside the xampp folder.

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:



PHP Case Sensitivity

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

PHP echo

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

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

Syntax:

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

eg.,

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

Output: Hello by PHP echo

PHP echo: printing multi line string

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

Output:

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

PHP echo: printing escaping characters

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

Output:

Hello escape "sequence" characters

PHP echo: printing variable value

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

Output:

Message is: Hello JavaTpoint PHP

PHP print

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

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

The syntax of PHP print is given below:

int print(string \$arg)

eg.,

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

Output:Hello by PHP print this is multi line text printed by PHP print statement

Difference between echo and print

echo

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

print

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

PHP 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

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:

- Scalar Types (predefined)
- Compound Types (user-defined)
- Special Types

PHP Data Types: Scalar Types

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

- boolean
- integer
- float
- string

PHP Data Types: Compound Types

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

- array

- object

PHP Data Types: Special Types

There are 2 special data types in PHP.

- resource
- NULL

PHP Boolean

Booleans are the simplest data type works like switch. It holds only two values: TRUE (1) or FALSE (0).

Example:

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

PHP Integer

It holds only whole numbers, i.e., numbers without fractional part or decimal points.

Rules for integer:

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

Example:

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

```
echo "Octal number: " . $oct1. "</br>";  
echo "HexaDecimal number: " . $hexa1. "</br>";  
?>
```

Output:

Decimal number: 34

Octal number: 163

HexaDecimal number: 69

PHP Float

Unlike integer, it can hold numbers with a fractional or decimal point, including a negative or positive sign.

PHP String

A string is a non-numeric data type. String values must be enclosed either within single quotes or in double quotes. But both are treated differently. To clarify this, see the example below:

Example:

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

Output:

Hello Javatpoint

Hello \$company

PHP Array

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

Example:

```
<?php  
$bikes = array ("Royal Enfield", "Yamaha", "KTM");  
var_dump($bikes); //the var_dump() function returns the datatype and  
values
```

```
echo "</br>";  
echo "Array Element1: $bikes[0] </br>";  
echo "Array Element2: $bikes[1] </br>";  
echo "Array Element3: $bikes[2] </br>";  
?>
```

Output:

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

Array Element1: Royal Enfield

Array Element2: Yamaha

Array Element3: KTM

PHP object

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

Example:

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

Output:

Bike Model: Royal Enfield

PHP Resource

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

PHP Null

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

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

Example:

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

PHP Variables

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

Here, some important points to know about variables:

As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.

After declaring a variable, it can be reused throughout the code.

Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

\$variablename=value;

PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed.

PHP has three types of variable scopes:

Local variable: The variables that are declared within a function are called local variables for that function.

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

?>

Global variable: The global variables are the variables that are declared outside the function.

```
<?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;
```

?>

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.

Static variable: It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. We use the static keyword before the variable to define a variable, and this variable is called as static variable.

```
<?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, which are not really constants. Unlike variables, constants are automatically global throughout the script. PHP constants can be defined by 2 ways:

- Using **define()** function: Use the define() function to create a constant. It defines constant at run time.

Syntax:

```
define(name, value, case-insensitive)
```

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

Eg.,

```
<?php
```

```
define("MESSAGE","Hello JavaTpoint PHP");
```

```
echo MESSAGE;
```

```
?>
```

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

```
<?php
```

```
const MESSAGE="Hello const by JavaTpoint PHP";
```

```
echo MESSAGE;
```

```
?>
```

Constant() function

There is another way to print the value of constants using `constant()` function instead of using the `echo` statement.

Syntax

`constant (name)`

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

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.

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

1. `__LINE__`: It returns the current line number of the file, where this constant is used.
2. `__FILE__`: This magic constant returns the full path of the executed file, where the file is stored.
3. `__DIR__`: It returns the full directory path of the executed file.
4. `__FUNCTION__`: This magic constant returns the function name, where this constant is used.
5. `__CLASS__`: It returns the class name, where this magic constant is used.
6. `__TRAIT__`: This magic constant returns the trait name, where it is used.
7. `__METHOD__`: It returns the name of the class method where this magic constant is included.
8. `__NAMESPACE__`: It returns the current namespace where it is used.
9. `ClassName::class`: This magic constant does not start and end with the double underscore (`__`). It returns the fully qualified name of the `ClassName`. `ClassName::class` is added in PHP 5.5.0. It is useful with namespaced classes.

All of the constants are resolved at compile-time instead of run time, unlike the regular constant. Magic constants are case-insensitive.

PHP Operators

Arithmetic Operators

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 $\$b$

Assignment Operators

Operator	Name	Example	Explanation
=	Assign	$\$a = \b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$\$a += \b	Addition same as $\$a = \$a + \$b$
-=	Subtract then Assign	$\$a -= \b	Subtraction same as $\$a = \$a - \$b$
*=	Multiply then Assign	$\$a *= \b	Multiplication same as $\$a = \$a * \$b$
/=	Divide then Assign (quotient)	$\$a /= \b	Find quotient same as $\$a = \$a / \$b$
%=	Divide then Assign (remainder)	$\$a \% = \b	Find remainder same as $\$a = \$a \% \$b$

Bitwise Operators

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/Relational Operators

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

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

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

String Operators

<u>Operator</u>	<u>Name</u>	<u>Example</u>	<u>Explanation</u>
.	<u>Concatenation</u>	<u>\$a . \$b</u>	<u>Concatenate both \$a and \$b</u>
<u>.=</u>	<u>Concatenation and Assignment</u>	<u>\$a .= \$b</u>	<u>First concatenate \$a and \$b, then assign the concatenated string to \$a, e.g. \$a = \$a . \$b</u>

Array Operators

<u>Operator</u>	<u>Name</u>	<u>Example</u>	<u>Explanation</u>
+	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.

Eg.,

```
$charu = new Developer();
//testing the type of object
if( $charu instanceof Developer)
```

```
{
    echo "Charu is a developer.";
}
else
{
    echo "Charu is a programmer.";
}
```

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

Eg,

```
echo `dir`
```

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.

Eg.,

```
@file('non_existent_file')
```

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.

Eg.,

```
<?php
```

```
$x = "abc";
```

```
$$x = 200;
```

```
echo $x."<br/>";
```

```
echo $$x."<br/>";
```

```
echo $abc;
```

```
?>
```

o/p:



UNIT II

Programming with PHP

PHP If Else

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

- **If**

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

Syntax

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

Eg.,

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

- **if-else**

If-else statement executes one block of code if the specified condition is true and another block of code if the condition is false.

Syntax:

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

Eg.,

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

- if-else-if

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

Syntax:

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

Eg,

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


- nested if

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

Syntax:

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

Eg.,

```
<?php  
    $age = 23;  
    $nationality = "Indian";  
    //applying conditions on nationality and age  
    if ($nationality == "Indian")  
    {  
        if ($age >= 18) {  
            echo "Eligible to give vote";  
        }  
        else {  
            echo "Not eligible to give vote";  
        }  
    }  
?>
```

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

```
switch(expression){  
case value1:  
    //code to be executed  
    break;  
case value2:  
    //code to be executed  
    break;
```

.....

default:

code to be executed if all cases are not matched;

}

Example:

```
<?php
```

```
$num=20;
```

```
switch($num){
```

```
case 10:
```

```
echo("number is equals to 10");
```

```
break;
```

```
case 20:
```

```
echo("number is equal to 20");
```

```
break;
```

```
case 30:
```

```
echo("number is equal to 30");
```

```
break;
```

```
default:
```

```
echo("number is not equal to 10, 20 or 30");
```

```
}
```

```
?>
```

PHP For Loop

PHP for loop can be used to traverse set of code for the specified number of times.

Syntax:

```
for(initialization; condition; increment/decrement){
```

```
//code to be executed
```

```
}
```

Eg.,

```
<?php
```

```
for($n=1;$n<=10;$n++){
```

```
echo "$n<br/>";
```

```
}
```

```
?>
```

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop. The inner for loop executes only when the outer for loop condition is found true.\

Eg.,

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

PHP For Each Loop

PHP for each loop is used to traverse array elements.

Syntax

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

Example:

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

PHP While Loop

The while loop is also called an Entry control loop because the condition is checked before entering the loop body.

Syntax

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

Alternative Syntax

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

eg.,

```
<?php
```

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

PHP do-while loop

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

Syntax

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

Example

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

PHP Break

The break keyword immediately ends the execution of the loop or switch structure and program control resumes at the next statements outside the loop/switch.

Syntax

```
jump statement;
break;
```

example:

```
<?php
for($i=1;$i<=10;$i++){
echo "$i <br/>";
if($i==5){
break;
}
}
}?>
```

PHP continue statement

It continues the current flow of the program or next iteration and skips the remaining code at the specified condition.

example:

```
<?php
for($i=1;$i<=10;$i++){
echo "$i <br/>";
if($i==5){
continue;
}
}
?>
```

PHP Conditional Assignment Operators

Operator	Name	Example	Result
?:	Ternary	\$x = <i>expr1</i> ? <i>expr2</i> : <i>expr3</i>	Returns the value of \$x. The value of \$x is <i>expr2</i> if <i>expr1</i> = TRUE. The value of \$x is <i>expr3</i> if <i>expr1</i> = FALSE
??	Null coalescing	\$x = <i>expr1</i> ?? <i>expr2</i>	Returns the value of \$x. The value of \$x is <i>expr1</i> if <i>expr1</i> exists, and is not NULL. If <i>expr1</i> does not exist, or is NULL, the value of \$x is <i>expr2</i> . Introduced in PHP 7

PHP Arrays**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.

- Indexed Array
- Associative Array
- Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

1st way:

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

2nd way:

```
$season[0]="summer";
```

```
$season[1]="winter";
```

```
$season[2]="spring";
```

```
$season[3]="autumn";
```

Eg.,

```
<?php
```

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

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

```
?>
```

Traversing PHP Indexed Array

We can easily traverse array in PHP using foreach loop.

Eg.,

```
<?php
```

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

```
foreach( $size as $s )
```

```
{
```

```
    echo "Size is: $s<br />";
```

```
}
```

```
?>
```

Count Length of PHP Indexed Array

PHP provides count() function which returns length of an array.

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

PHP Associative Array

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

There are two ways to define associative array:

1st way:

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

2nd way:

```
$salary["Sonoo"]="350000";
```

```
$salary["John"]="450000";
```

```
$salary["Kartik"]="200000";
```

Eg.,

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

Traversing PHP Associative Array

By the help of PHP for each loop, we can easily traverse the elements of PHP associative array.

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

Output:

Key: Sonoo Value: 550000

Key: Vimal Value: 250000

Key: Ratan Value: 200000

PHP Multidimensional Array

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

```
$emp = array
```

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

Eg.,

```
<?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/>";  
}
```

```
?>
```

PHP Array Functions

1) PHP array() function

PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.

Syntax

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

Example

```
<?php
```

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

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

2) PHP array_change_key_case() function

PHP array_change_key_case() function changes the case of all key of an array.

****** It changes case of key only.

Syntax

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

Example

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

Output:

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

3) PHP array_chunk() function

PHP array_chunk() function splits array into chunks. By using array_chunk() method, you can divide array into many parts.

Syntax

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

Example

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

Output:

```
Array (
  [0] => Array ( [0] => 550000 [1] => 250000 )
  [1] => Array ( [0] => 200000 )
)
```

4) PHP count() function

PHP count() function counts all elements in an array.

Syntax

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

Example

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

?>

Output:

4

5) PHP sort() function

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

Syntax

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

Example

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

Output:

autumn
spring
summer
winter

6) PHP array_reverse() function

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

Syntax

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

Example

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

Output:

autumn

spring
winter
summer

7) PHP array_search() function

PHP array_search() function searches the specified value in an array. It returns key if search is successful.

Syntax

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

Example

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

Output:

2

8) PHP array_intersect() function

PHP array_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.

Syntax

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

Example

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

Output

sonoo
smith

9.PHP array_push() Function

The array_push() function inserts one or more elements to the end of an array.

Syntax

```
array_push(array, value1, value2, ...)
```

Example

```
<?php
$a=array("red","green");
array_push($a,"blue","yellow");
print_r($a);
?>
```

10. PHP array_pop() Function\

Delete the last element of an array:

Syntax: array_pop(array)

Example:

```
<?php
$a=array("red","green","blue");
array_pop($a);
print_r($a);
?>
```

11.PHP array_splice() Function:

New item in an array can be inserted with the help of array_splice() function of PHP. This function removes a portion of an array and replaces it with something else. If offset and length are such that nothing is removed, then the elements from the replacement array are inserted in the place specified by the offset.

Syntax:

```
array array_splice ($input, $offset [, $length [, $replacement]])
```

eg.,

```
<?php
//Original Array on which operations is to be perform
$original_array = array( '1', '2', '3', '4', '5' );
echo 'Original array : ';
foreach ($original_array as $x)
{
    echo "$x ";
}
echo "\n";
//value of new item
$inserted_value = '11';
```

```
//value of position at which insertion is to be done
$position = 2;
//array_splice() function
array_splice( $original_array, $position, 0, $inserted_value );
echo "After inserting 11 in the array is : ";
foreach ( $original_array as $x )
{
    echo "$x ";
}
?>
```

o/p:

Original array : 1 2 3 4 5

After inserting 11 in the array is : 1 2 11 3 4 5

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.

- include
- 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.

PHP include

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

Syntax

There are two syntaxes available for include:

include 'filename ';

Or

include ('filename');

Examples

Let's see a simple PHP include example.

File: menu.html

`Home |`

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

File: include1.php

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

Output:

Home |

PHP |

Java |

HTML

This is Main Page

PHP require

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

Syntax

There are two syntaxes available for require:

require 'filename';

Or

require ('filename');

Examples

Let's see a simple PHP require example.

File: menu.html

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

File: require1.php

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

Output:

Home |

PHP |

Java |

HTML

This is Main Page

PHP - Type Casting

Implicit Type Casting:

It is coercive type casting.

```
<?php
$a = 10;
$b = '20';
$c = $a+$b;
echo "c = " . $c;
?>
```

Here \$b is automatically converted to integer.

Explicit Type Casting:

\$var = (type)expr;

Some of the type casting operators in PHP are –

- (int) or (integer) casts to an integer
- (bool) or (boolean) casts to a boolean
- (float) or (double) or (real) casts to a float
- (string) casts to a string
- (array) casts to an array
- (object) casts to an object

Example

```
$a = "10 Rs.";
$b = (int)$a;
var_dump($b);
$a = 100;
$b = (double)$a;
var_dump($b);
```

Type Casting Functions

PHP includes the following built-in functions for performing type casting –

intval():

intval(mixed \$value, int \$base = 10): int

- By default base is 10, converting to decimal, it can also be 0x,0B,0 for hexadecimal, binary and octal.

Example:

```
<?php
```

```
echo intval(42). PHP_EOL;
echo intval('042', 0) . PHP_EOL; # Octal number
echo intval('42', 8) . PHP_EOL; # octal
echo intval(0x1A) . PHP_EOL; # Hexadecimal
echo intval('0x1A', 16) . PHP_EOL; # Hexadecimal
echo intval('0x1A', 0) . PHP_EOL; # Hexadecimal
?>
```

floatval():

floatval(mixed \$value): float

example:

```
<?php
echo floatval(42). PHP_EOL;
echo floatval(4.2). PHP_EOL;
echo floatval('42') . PHP_EOL;
echo floatval('99.90 Rs') . PHP_EOL;
echo floatval('$100.50') . PHP_EOL;
echo floatval('ABC123!@#') . PHP_EOL;
echo (true) . PHP_EOL; ;
echo (false) . PHP_EOL;
?>
```

O/P:

```
42
4.2
42
99.9
0
0
1
```

strval():

strval(mixed \$value): string

```
<?php
echo strval(42). PHP_EOL;
echo strval(4.2). PHP_EOL;
echo strval(4.2E5) . PHP_EOL;
echo strval(NULL) . PHP_EOL;
echo (true) . PHP_EOL;
```



```
echo (false) . PHP_EOL;  
?>
```

o/p:

42

4.2

420000

1

UNIT-3

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.

PHP User-defined Functions

Syntax

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

Example:

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

PHP Function Arguments

PHP supports **Call by Value (default)**, **Call by Reference**, **Default argument values** and **Variable-length argument list**.

PHP Parameterized Function

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

example to **pass two argument in PHP function**.

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

Output:

Hello Sonoo, you are 27 years old

Hello Vimal, you are 29 years old

Hello John, you are 23 years old

PHP Call By Value

In case of PHP call by value, actual value is not modified if it is modified inside the function.

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

Output: Hello Call By Value

Hello

PHP Call By Reference:

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Example:

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

PHP Function: Default Argument Value

We can specify a default argument value in function.

Example:

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

```
}  
sayHello("Rajesh");  
sayHello();//passing no value  
sayHello("John");  
?>
```

PHP Function: Returning Value

Example of PHP function that returns value:

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

Output:

Cube of 3 is: 27

PHP Variable Length Argument Function

PHP supports variable length argument function. It means you can pass 0, 1 or n number of arguments in function.

Example:

```
?php  
function add(...$numbers) {  
    $sum = 0;  
    foreach ($numbers as $n) {  
        $sum += $n;  
    }  
    return $sum;  
}  
echo add(1, 2, 3, 4);  
?>
```

Output:

10

PHP Recursive Function

Recursion is we call current function within function.

```
<?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

Library Functions : PHP provides us with huge collection of built-in library functions. These functions are already coded and stored in form of functions. To use those we just need to call them as per our requirement like, var_dump, fopen(), print_r(), gettype() and so on.

PHP Date and Time:

PHP date() Function: The PHP date() function converts timestamp to a more readable date and time format.

Syntax:

date(format, timestamp)

Explanation:

- The format parameter in the date() function specifies the format of returned date and time.
- The timestamp is an optional parameter, if it is not included then the current date and time will be used.

Example:

```
?php
echo "Today's date is :";
$today = date("d/m/Y");
echo $today;
?>
```

Output:

Today's date is :05/05/2024

Formatting options available in date() function\

- d: Represents day of the month; two digits with leading zeros (01 or 31).
- D: Represents day of the week in the text as an abbreviation (Mon to Sun).
- m: Represents month in numbers with leading zeros (01 or 12).

- M: Represents month in text, abbreviated (Jan to Dec).
- y: Represents year in two digits (08 or 14).
- Y: Represents year in four digits (2008 or 2014).
- The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

The following characters can be used along with the date() function to format the time string:

- h: Represents hour in 12-hour format with leading zeros (01 to 12).
- H: Represents hour in 24-hour format with leading zeros (00 to 23).
- i: Represents minutes with leading zeros (00 to 59).
- s: Represents seconds with leading zeros (00 to 59).
- a: Represents lowercase antemeridian and post meridian (am or pm).
- A: Represents uppercase antemeridian and post meridian (AM or PM).

Example:

```
<?php
echo date("h:i:s") . "\n";
echo date("M,d,Y h:i:s A") . "\n";
echo date("h:i a");
?>
```

o/p: 03:04:17

Dec,05,2017 03:04:17 PM

03:04 pm

PHP time() Function: The time() function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1, 1970, 00:00:00 GMT).

The following characters can be used to format the time string:

- h: Represents hour in 12-hour format with leading zeros (01 to 12).
- H: Represents hour in 24-hour format with leading zeros (00 to 23).
- i: Represents minutes with leading zeros (00 to 59).
- s: Represents seconds with leading zeros (00 to 59).
- a: Represents lowercase antemeridian and post meridian (am or pm).
- A: Represents uppercase antemeridian and post meridian (AM or PM).

Example:

```
<?php
```

```
$timestamp = time();  
echo($timestamp);  
echo "\n";  
echo(date("F d, Y h:i:s A", $timestamp));  
?>
```

o/p: 1512486297

December 05, 2017 03:04:57 PM

PHP mktime() Function: The mktime() function is used to create the timestamp for a specific date and time. If no date and time are provided, the timestamp for the current date and time is returned.

Syntax:

mktime(hour, minute, second, month, day, year)

eg.,

```
<?php  
echo mktime(23, 21, 50, 11, 25, 2017);  
?>
```

The above code creates a time stamp for 25th Nov 2017, 23 hrs 21 mins 50 secs.

o/p: 1511652110

Strings in PHP

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

PHP supports only 256-character set and so that it does not offer native Unicode support.

Creating and Declaring Strings

There are 4 ways to specify a string literal in PHP.

Single quoted:

We can create a string in PHP by enclosing the text in a single-quote. In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \ and backslash through \\ inside single quoted PHP strings.

Example:

```
<?php  
$str1='Hello text  
multiple line  
text within single quoted string';  
$num=10;  
$str2='Using double "quote" directly inside single quoted string $num';  
$str3='Using escape sequences \n in single quoted string';
```

```
echo "$str1 <br/> $str2 <br/> $str3";  
?>
```

Output:

Hello text multiple line text within single quoted string

Using double "quote" directly inside single quoted string \$num

Using escape sequences \n in single quoted string

Double quoted: In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

Example:

```
<?php  
$str1="Hello text  
multiple line  
text within double quoted string";  
$num=10;  
$str2="Using double \"quote\" with backslash inside double quoted string  
$num";  
$str3="Using escape sequences \n in double quoted string";  
echo "$str1 <br/> $str2 <br/> $str3";  
?>
```

Output:

Hello text multiple line text within double quoted string

Using double "quote" with backslash inside double quoted string 10

Using escape sequences in double quoted string

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

Eg.,

```
<?php  
$str = <<<Demo
```

It is a valid example

Demo; //Valid code as whitespace or tab is not valid before closing identifier

```
echo $str;
```

```
?>
```

Output:

It is a valid example

Newdoc syntax (since PHP 5.3): Newdoc is similar to the heredoc, but in newdoc parsing is not done. It is also identified with three less than symbols <<< followed by an identifier. But here identifier is enclosed in single-quote, e.g. <<<'EXP'.

```
<?php
    $str = <<<'DEMO'
    Welcome to javaTpoint.
    Learn with newdoc example.
```

```
DEMO;
echo $str;
echo '</br>';
```

```
echo <<< 'Demo' // Here we are not storing string content in variable str.
    Welcome to javaTpoint.
    Learn with newdoc example.
```

```
Demo;
?>
```

Output:

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

The difference between newdoc and heredoc is that - Newdoc is a single-quoted string whereas heredoc is a double-quoted string.

String Functions

1. strlen() function: This function is used to find the length of a string.

Syntax

```
int strlen ( string $string )
```

Example

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

2. strrev() function: This function is used to reverse a string.

Syntax

```
string strev ( string $string )
```

Example

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

Output:lawsiaj oonoS si eman ym

3. str_replace() function: This function takes three strings as arguments. The third argument is the original string and the first argument is replaced by the second one. Count is optional and specifies number of replacements done.

Syntax

```
str_replace ( $search, $replace, $string, $count)
```

eg.,

```
<?php
echo str_replace("Geeks", "World", "Hello GeeksforGeeks!"), "\n";
echo str_replace("for", "World", "Hello GeeksforGeeks!"), "\n";
?>
```

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.

Eg.,

```
<?php
echo strpos("Hello GeeksforGeeks!", "Geeks"), "\n";
echo strpos("Hello GeeksforGeeks!", "for"), "\n";
var_dump(strpos("Hello GeeksforGeeks!", "Peek"));
?>
```

o/p: 6

11

bool(false)

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

Eg.,

Eg.,

```
<?php
```

```
echo trim(" Hello GeeksforGeeks! ");  
?>
```

6. strtoupper() function: The strtoupper() function returns string in uppercase letter.

Eg.,

```
<?php  
$input = "Welcome to geeksforgeeks";  
echo strtoupper($input);  
?>
```

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

Example:

```
<?php  
$input = "Welcome to GeeksforGeeks";  
echo strtolower($input);  
?>
```

8.str_word_count() function: This function counts the number of characters in string.

Example:

```
<?php  
$input = "Welcome to GeeksforGeeks";  
echo str_word_count($input);  
?>
```

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

Example:

```
<?php  
$input = "Welcome to geeksforgeeks";  
echo explode($input);  
?>
```

Output:

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

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

Syntax

```
substr( $string, $start, $length )
```

Example:

```
<?php
```

```
$input = "Welcome to geeksforgeeks";  
echo(substr($input,3));  
?>
```

Output:

come to geeksforgeeks

11.String Reverse: strrev()

This function reverses the order of a string.

Example:

```
<?php  
$text = "Hello, World!";  
$reversed = strrev($text);  
echo "Reversed: $reversed";  
?>
```

12. String Compare: strcmp()

This function compares two strings and returns 0 if they are equal, a positive number if the first string is greater, and a negative number if the second string is greater.

Example:

```
<?php  
$string1 = "apple";  
$string2 = "banana";  
$result = strcmp($string1, $string2);  
echo "Comparison Result: $result";  
?>
```

UNIT-4

PHP Classes

PHP also supports the concept of object-oriented programming.

PHP classes are almost like functions but with a major different that classes contain both variables and function, in a single format called object or can be said Class is made from a collection of objects.

Syntax:

```
<?php
class myFirstClass {
    var $ var _ a ;
    var $ var _ b = " constant string " ;

    function classFunction ( $ parameter 1 , $ parameter 2 ) {
        [ ..... ]
    }
    [ ..... ]
}
?>
```

1	Class	To declare a class, we have to use the keyword class and then the name of the class that we want to declare.
2	Variable declaration	To declare a variable, we have to declare keyword var followed by \$ convention and then the name of the declared variable. These are also known as member variables. These are also called properties
3	Function declaration	To declare a function, we have to declare the keyword function following the name of the declared function. These are also known as member functions, but these functions are only accessible to class only. These are also called methods
4	Braces	We have to enclose our class with curly braces { }

```
<?php
class Employee
{
    /* Member variables */
    var $name;
    var $salary;
    var $profile;
    // Constructor
    public function __construct(){
        echo 'The class "' . __CLASS__ . '" was initiated!<br>';
    }
    /* Member functions */
    function set_name($new){
        $this->name = $new;
    }
}
?>
```

Creating Objects in PHP to access the class

Objects are created using new operator.

Eg.,

```
$employee_1 = new employees;
```

Eg.,

```
<!DOCTYPE html>
```

```
<html lang = " en ">
```

```
<head>
```

```
    <meta charse t= " UTF ? 8 ">
```

```
    <meta http ? equiv = " X ? UA ? Compatible " content = " IE = edge ">
```

```
    <meta name = " viewport " content = " width = device - width, initial ? scale = 1 .0">
```

```
    <title> PHP </title>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
class employees {
```

```
    /* Member variables */
```

```
    var $name;
```

```
    var $salary;
```

```
var $profile;

/* Member functions */
function set_name($new){
    $this->name = $new;
}

function get_name(){
    echo $this->name . "<br/>";
}

function set_salary($new){
    $this->salary = $new;
}

function get_salary(){
    echo $this->salary . " <br/>";
}

function set_profile($new){
    $this->profile = $new;
}
function get_profile(){
    echo $this->profile . " <br/>";
}

}
$employee_1 = new employees;
$employee_1->set_name( "JOHN" );
$employee_1->set_salary( "10000" );
$employee_1->set_profile( "audit manager" );
$employee_1->get_name();
$employee_1->get_salary();
$employee_1->get_profile();
echo "<br/>";
?>
</body>
```

</html>

Output:

JOHN

10000

audit manager

\$ this Keyword

\$ this keyword refers to the present object, and this keyword can only be used inside a member function of a class. We can use \$ this keyword in two ways

1) To add a value to the declared variable, we have to use \$ this property inside the set _ name function

Eg.,

<?php

```
class employee {  
    public $name;  
    var $salary;  
    function set_name($name) {  
        $this->name = $name;  
    }  
    function set_salary($salary) {  
        $this->salary = $salary;  
    }  
}  
$employee_1 = new employee();  
$employee_1->set_name("JOHN");  
$employee_1->set_salary(" $ 2000000"); ?>
```

2) In order to add a value to declared variable, the property of variable can be changed directly.

Eg.,

<?php

```
class employee {  
    public $name;  
    var $salary;  
  
}  
$employee_1 = new employee();  
$employee_1->name = " JOHN ";  
$employee_1->salary = " $ 2000000 ";
```


?>

Constructors

These are a special types of functions that are automatically called whenever an object is created. These function are created by using the `__construct ()` keyword. The constructors have no Return Type, so they do not return anything not even void.

Constructor types:

- Default Constructor: It has no parameters, but the values to the default constructor can be passed dynamically.
- Parameterized Constructor: It takes the parameters, and also you can pass different values to the data members.
- Copy Constructor: It accepts the address of the other objects as a parameter.

Syntax:

```
function __construct( $ parameter 1, $ parameter 2 ) {  
    $this->name = $ parameter 1;  
    $this->state = $ parameter 2;  
}  
<?php  
class newconstructorclass  
{  
    // Constructor  
    function __construct($new){  
        echo 'the constructor class has been initiated' . "<br/>" ;  
        $this->name = $new;  
    }  
    function get_name(){  
        echo $this->name . "<br/>";  
    }  
}  
// Create a new object  
$obj = new newconstructorclass( " john " );  
$obj -> get_name();  
?>
```

Advantages of using Constructors:

1. Constructors provides the ability to pass parameters which are helpful in automatic initialization of the member variables during creation time .

- 2.The Constructors can have as many parameters as required and they can be defined with the default arguments.
- 3.They encourage re-usability avoiding re-initializing whenever instance of the class is created .
- 4.You can start session in constructor method so that you don't have to start in all the functions everytime.
- 5.They can call class member methods and functions.
- 6.They can call other Constructors even from Parent class.

Destructors

These are special functions that are automatically called whenever an object goes out of scope or gets deleted. These functions are created by using the `__destruct ()` keyword.

Syntax:

```
function __destruct() {  
[ .....]  
[ .....] }
```

Eg.,

```
<?php
```

```
class newdestructorclass
```

```
{
```

```
    // Destructor
```

```
    public function __destruct(){
```

```
        echo 'The class ' . __CLASS__ . ' was automatically destroyed when the  
        created object does not have a scope to initiate';
```

```
    }
```

```
}
```

```
// Create a new object
```

```
$obj = new newdestructorclass;
```

```
?>
```

Advantages of destructors:

- 1.Destructors give chance to objects to free up memory allocation , so that enough space is available for new objects or free up resources for other tasks.
- 2.It effectively makes programs run more efficiently and are very useful as they carry out clean up tasks.

Comparison between __constructors and __destructors:

| Constructors | Destructors |
|--|--|
| Accepts one or more arguments. | No arguments are passed. Its void. |
| function name is _construct(). | function name is _destruct() |
| It has same name as the class. | It has same name as the class with prefix ~tilda. |
| Constructor is involved automatically when the object is created. | Destructor is involved automatically when the object is destroyed. |
| Used to initialize the instance of a class. | Used to de-initialize objects already existing to free up memory for new accommodation. |
| Used to initialize data members of class. | Used to make the object perform some task before it is destroyed. |
| Constructors can be overloaded. | Destructors cannot be overloaded. |
| It is called each time a class is instantiated or object is created. | It is called automatically at the time of object deletion . |
| Allocates memory. | It deallocates memory. |
| Multiple constructors can exist in a class. | Only one Destructor can exist in a class. |
| If there is a derived class inheriting from base class and the object of the derived class is | The destructor of the derived class is called and then the destructor of |

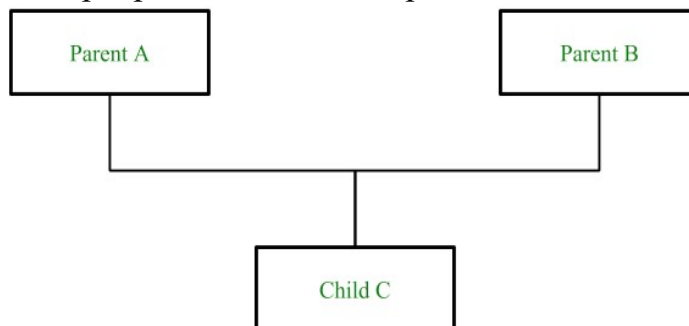
| Constructors | Destructors |
|--|---|
| created,
the constructor of base class is created and then the constructor of the derived class. | base class just the reverse order of constructor. |
| The concept of copy constructor is allowed where an object is initialized from the address of another object . | No such concept is allowed. |

Class Access Specifier

| Class Member Access Specifier | Access from own class | Accessible from derived class | Accessible by Object |
|-------------------------------|-----------------------|-------------------------------|----------------------|
| Private | Yes | No | No |
| Protected | Yes | Yes | No |
| Public | Yes | Yes | Yes |

Multiple Inheritance in PHP

Multiple Inheritance is the property of the Object Oriented Programming languages in which child class or sub class can inherit the properties of the multiple parent classes or super classes.



PHP doesn't support multiple inheritance but by using Interfaces in PHP or using **Traits in PHP** instead of classes, we can implement it.

Traits (Using Class along with Traits): The trait is a type of class which enables multiple inheritance. Class can extend only one other class but can extend more than one trait at a time.

Syntax:

```
class child_class_name extends parent_class_name {  
    use trait_name;  
    ...  
    ...  
    child_class functions  
}
```

Example:

```
<?php  
// Class Geeks  
class Geeks {  
    public function sayhello() {  
        echo "Hello";  
    }  
}  
// Trait forGeeks  
trait forGeeks {  
    public function sayfor() {  
        echo " Geeks";  
    }  
}  
class Sample extends Geeks {  
    use forGeeks;  
    public function geeksforgeeks() {  
        echo "\nGeeksforGeeks";  
    }  
}  
$test = new Sample();  
$test->sayhello();  
$test->sayfor();  
$test->geeksforgeeks();  
?>
```

Overloading in PHP

Function overloading in PHP is used to dynamically create properties and methods. Function overloading contains same function name and that function performs different task according to number of arguments. In PHP function overloading is done with the help of magic function `__call()`. This function takes function name and arguments.

Property and Rules of overloading in PHP:

1. All overloading methods must be defined as Public.
2. After creating the object for a class, we can access a set of entities that are properties or methods not defined within the scope of the class.
3. Such entities are said to be overloaded properties or methods, and the process is called as overloading.
4. For working with these overloaded properties or functions, PHP magic methods are used.
6. Most of the magic methods will be triggered in object context except `__callStatic()` method which is used in a static context.

Types of Overloading in PHP: There are two types of overloading in PHP.

- Property Overloading
- Method Overloading

Property Overloading: PHP property overloading is used to create dynamic properties in the object context. Following operations are performed with overloaded properties in PHP.

- Setting and getting overloaded properties.
- Evaluating overloaded properties setting.
- Undo such properties setting.

Before performing the operations, we should define appropriate magic methods, which are,

`__set()`: triggered while initializing overloaded properties.

`__get()`: triggered while using overloaded properties with PHP print statements.

`__isset()`: This magic method is invoked when we check overloaded properties with `isset()` function

`__unset()`: Similarly, this function will be invoked on using PHP `unset()` for overloaded properties.

Example.,

```
<?php
```

```
//
```

```
class GFG {
```

```
// Location of overloading data
private $data = array();
// Overloading not used here
public $declared = 1;
// Overloading used when accessed
// outside the class
private $hidden = 2;
// Function definition
public function __set($name, $value) {
    echo "Setting '$name' to '$value'\n";
    $this->data[$name] = $value;
}

// Function definition
public function __get($name) {
    echo "Getting '$name: ";
    if (array_key_exists($name, $this->data)) {
        return $this->data[$name];
    }
    $trace = debug_backtrace();
    return null;
}

// Function definition
public function __isset($name) {
    echo "Is '$name' set?\n";
    return isset($this->data[$name]);
}

// Definition of __unset function
public function __unset($name) {
    echo "Unsetting '$name'\n";
    unset($this->data[$name]);
}

// getHidden function definition
```

```
        public function getHidden() {
            return $this->hidden;
        }
    }

    // Create an object
    $obj = new GFG;
    // Set value 1 to the object variable
    $obj->a = 1;
    echo $obj->a . "\n";
    // Use isset function to check
    // 'a' is set or not
    var_dump(isset($obj->a));
    // Unset 'a'
    unset($obj->a);
    var_dump(isset($obj->a));
    echo $obj->declared . "\n\n";
    echo "Private property are visible inside the class ";
    echo $obj->getHidden() . "\n\n";
    echo "Private property are not visible outside of class\n";
    echo $obj->hidden . "\n";
    ?>
```

Output:

```
Setting 'a' to '1'
Getting 'a': 1
Is 'a' set?
bool(true)
Unsetting 'a'
Is 'a' set?
bool(false)
1
```

Private property are visible inside the class 2

Private property are not visible outside of class
Getting 'hidden:

Method Overloading: It is a type of overloading for creating dynamic methods that are not declared within the class scope. PHP method overloading allows function call on both object and static context. The related magic functions are,

- `__call()` – triggered while invoking overloaded methods in the object context.
- `__callStatic()` – triggered while invoking overloaded methods in static context.

Example:

```
<?php
class GFG {
    public function __call($name, $arguments) {
        echo "Calling object method '$name' "
            . implode(' ', $arguments). "\n";
    }
    public static function __callStatic($name, $arguments) {

        echo "Calling static method '$name' "
            . implode(' ', $arguments). "\n";
    }
}
// Create new object
$obj = new GFG;
$obj->runTest('in object context');
GFG::runTest('in static context');
?>
```

Output:

Calling object method 'runTest' in object context

Calling static method 'runTest' in static context

Function Overriding: Function overriding is same as other OOPs programming languages. In function overriding, both parent and child classes should have same function name with and number of arguments. The purpose of overriding is to change the behavior of parent class method. The two methods with the same name and same parameter is called overriding.

Example:

```
<?php
// PHP program to implement
// function overriding
```

```
// This is parent class
class P {

    // Function geeks of parent class
    function geeks() {
        echo "Parent";
    }
}

// This is child class
class C extends P {

    // Overriding geeks method
    function geeks() {
        echo "\nChild";
    }
}

// Reference type of parent
$p = new P;

// Reference type of child
$c= new C;

// print Parent
$p->geeks();

// Print child
$c->geeks();
?>
```

PHP Form Handling:

The form request may be get or 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.

Eg.,

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 visible on the URL browser so it is secured.

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 Form Validation

PHP methods and arrays used in form processing are:

1.isset(): This function is used to determine whether the variable or a form control is having a value or not.

Eg.,

```
If (isset($_POST['submit']))
```

```
{
```

```
    $number= $_GET["number"];
```

```
}
```

2. `$_GET[]`: It is used to retrieve the information from the form control through the parameters sent in the URL. It takes the attribute given in the url as the parameter.

e.g, `$number= $_GET["number"];`

3. `$_POST[]`: It is used to retrieve the information from the form control through the HTTP POST method. It takes name attribute of corresponding form control as the parameter.

Eg., `$number= $_POST["number"];`

4. `$_REQUEST[]`: It is used to retrieve an information while using a database.

5. `$_SERVER`: `$_SERVER` is a PHP super global variable which holds information about headers, paths, request method and script locations.

PHP form Validation Example:

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
.error {color: #FF0000;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<?php
```

```
// define variables and set to empty values
```

```
$nameErr = $emailErr = $genderErr = $websiteErr = "";
```

```
$name = $email = $gender = $comment = $website = "";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
```

```
    if (empty($_POST["name"])) {
```

```
        $nameErr = "Name is required";
```

```
    } else {
```

```
        $name = test_input($_POST["name"]);
```

```
// check if name only contains letters and whitespace
if (!preg_match("/^[a-zA-Z-' ]*$/",$name)) {
    $nameErr = "Only letters and white space allowed";
}
}

if (empty($_POST["email"])) {
    $emailErr = "Email is required";
} else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $emailErr = "Invalid email format";
    }
}

if (empty($_POST["website"])) {
    $website = "";
} else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows
    dashes in the URL)
    if (!preg_match("/^b(?:(:https?|ftp):\\V|www\\.)[-a-z0-9+&@#\\/%?~_!|:.,;]*[-a-z0-9+&@#\\/%?~_]/i",$website)) {
        $websiteErr = "Invalid URL";
    }
}

if (empty($_POST["comment"])) {
    $comment = "";
} else {
    $comment = test_input($_POST["comment"]);
}

if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
} else {
```

```

    $gender = test_input($_POST["gender"]);
}
}

```

```

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>

```

<h2>PHP Form Validation Example</h2>

```

<p><span class="error">* required field</span></p>
<form          method="post"          action="<?php          echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    Name: <input type="text" name="name" value="<?php echo $name;?>">
    <span class="error">* <?php echo $nameErr;?></span>
    <br><br>
    E-mail: <input type="text" name="email" value="<?php echo $email;?>">
    <span class="error">* <?php echo $emailErr;?></span>
    <br><br>
    Website:  <input  type="text"   name="website"   value="<?php   echo
$website;?>">
    <span class="error"><?php echo $websiteErr;?></span>
    <br><br>
    Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
    <br><br>
    Gender:
    <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
    <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
    <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
    <span class="error">* <?php echo $genderErr;?></span>

```

```
<br><br>
<input type="submit" name="submit" value="Submit">
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
</body>
</html>
```

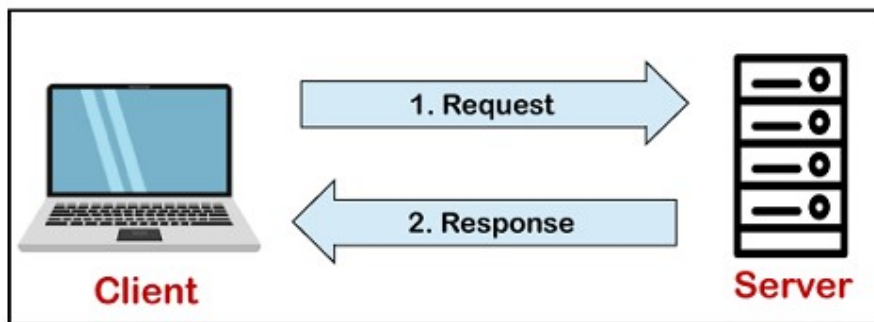
MYSQL INTRODUCTION:

MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language.

MySQL is a Relational Database Management System (RDBMS) software that provides many things, which are as follows:

- It allows us to implement database operations on tables, rows, columns, and indexes.
- It defines the database relationship in the form of tables (collection of rows and columns), also known as relations.
- It provides the Referential Integrity between rows or columns of various tables.
- It allows us to updates the table indexes automatically.
- It uses many SQL queries and combines useful information from multiple tables for the end-users.

The process of MySQL environment is the same as the **client-server model**.



The core of the MySQL database is the MySQL Server. This server is available as a separate program and responsible for handling all the database instructions, statements, or commands. The working of MySQL database with MySQL Server are as follows:

- MySQL creates a database that allows you to build many tables to store and manipulate data and defining the relationship between each table.

- Clients make requests through the GUI screen or command prompt by using specific SQL expressions on MySQL.
- Finally, the server application will respond with the requested expressions and produce the desired result on the client-side.

MySQL Features

- Relational Database Management System (RDBMS)
- Easy to use
- It is secure
- Client/ Server Architecture
- Free to download
- It is scalable
- Speed
- High Flexibility
- Compatible on many operating systems
- Allows roll-back
- MySQL allows transactions to be rolled back, commit, and crash recovery.
- Memory efficiency
- Its efficiency is high because it has a very low memory leakage problem.
- High Performance
- MySQL uses Triggers, Stored procedures, and views that allow the developer to give higher productivity.
- Platform Independent
- Partitioning
- MySQL provides a unified visual database graphical user interface tool named "MySQL Workbench" to work with database architects, developers, and Database Administrators.
- Dual Password Support

Disadvantages/Drawback of MySQL

- MySQL version less than 5.0 doesn't support ROLE, COMMIT, and stored procedure.
- MySQL does not support a very large database size as efficiently.
- MySQL doesn't handle transactions very efficiently, and it is prone to data corruption.

- MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.
- MySQL doesn't support SQL check constraints.

MySQL Data Types

MySQL supports a lot number of SQL standard data types in various categories. It uses many different data types that can be broken into the following categories: numeric, date and time, string types, spatial types, and JSON data types.

Numeric Data Types

Numeric data types are used to store numeric values, including integers and decimal numbers. MySQL uses all the standard ANSI SQL numeric data types.

- **INT** – A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** – A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** – A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** – A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals

and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

- **DOUBLE(M,D)** – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** – An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL. Internally, MySQL stores DECIMAL values using a binary format that allocates storage for the integer and fractional parts of the number separately. This binary format efficiently packs 9 digits into 4 bytes of storage

Date and Time Data Types

The MySQL date and time data types are as follows –

- **DATE** – A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** – A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** – A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** – Stores the time in a HH:MM:SS format.
- **YEAR(M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

String Data Types

The following list describes the common string data types in MySQL –

CHAR(M) – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

VARCHAR(M) – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.

BLOB or TEXT – A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data. The difference between the two is that the sorts and comparisons on the stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

TINYBLOB or TINYTEXT – A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.

MEDIUMBLOB or MEDIUMTEXT – A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

LOB or LONGTEXT – A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LOB or LONGTEXT.

ENUM – An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

The MySQL Varchar Data Type

The MySQL VARCHAR data type is used to store variable-length character strings, having a length up to 65,535 bytes.

In MySQL, when you store text in a VARCHAR column, it needs a little extra space to keep track of how long the text is. This extra space can be either 1 or 2

bytes, depending on the length of the text. If the text is short (less than 255 characters), it uses 1 byte for length. For longer text, it uses 2 bytes.

The total size of data plus the length info cannot exceed 65,535 bytes for a row in a table.

MySQL - Boolean Datatype

MySQL considers the value 0 as FALSE and 1 as TRUE. We can also store NULL values using the TINYINT datatype. The Boolean values (such as TRUE and FALSE) are not case-sensitive.

Spatial Data Type

It is a special kind of data type which is used to hold various geometrical and geographical values.

Data Types	Descriptions
GEOMETRY	It is a point or aggregate of points that can hold spatial values of any type that has a location.
POINT	A point in geometry represents a single location. It stores the values of X, Y coordinates.
POLYGON	It is a planar surface that represents multisided geometry. It can be defined by zero or more interior boundary and only one exterior boundary.
LINESTRING	It is a curve that has one or more point values. If it contains only two points, it always represents Line.
GEOMETRYCOLLECTION	It is a kind of geometry that has a collection of zero or more geometry values.
MULTILINESTRING	It is a multi-curve geometry that has a collection of linestring values.
MULTIPOINT	It is a collection of multiple point elements. Here, the point cannot be connected or ordered in any way.
MULTIPOLYGON	It is a multisurface object that represents a collection of multiple polygon elements. It is a type of two-dimensional geometry.

JSON Data Type

The JSON data type has the following advantages over storing JSON-format strings in a string column:

- It provides automatic validation of JSON documents. If we stored invalid documents in JSON columns, it would produce an error.
- It provides an optimal storage format.

UNIT 5

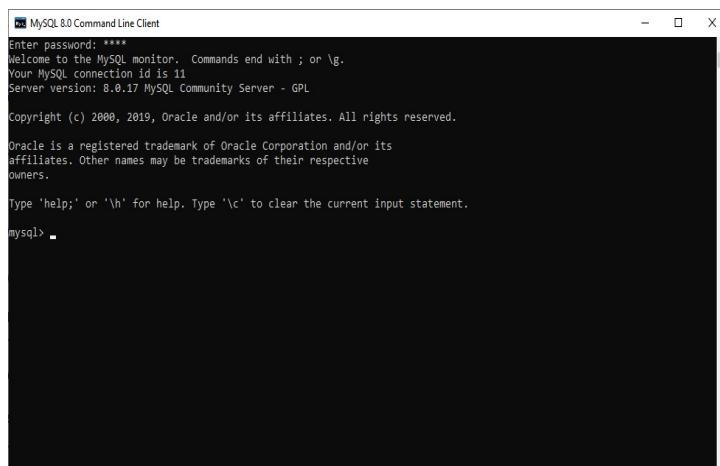
Using MySQL Client

What is a MySQL client?

MySQL client is a common name for tools that are designed to connect to MySQL Server. Client programs are used to send commands or queries to the server and allow managing data in the databases stored on the server.

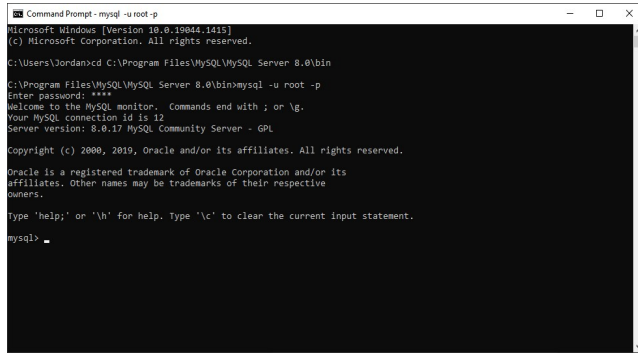
How to use MySQL Command Line Client

To access MySQL Server from the command-line client, open the program and enter the password. After that, you will be able to use the client.



You can also access MySQL Command Line Client from Command Prompt. For this:

- Open Command Prompt.
- Navigate to the bin folder. For example: `cd C:\Program Files\MySQL\MySQL Server 8.0\bin`
- Run the `mysql -u root -p` command.
- Enter the password.



```
Command Prompt - mysql -u root -p
Microsoft Windows [Version 10.0.19044.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jordan>cd C:\Program Files\MySQL\MySQL Server 8.0\bin
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.17 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

How to create a user from the command line

CREATE USER 'username' IDENTIFIED BY 'password';

You need to grant certain privileges to this user.

GRANT SELECT ON *.* TO 'username';

GRANT ALL PRIVILEGES ON *.* TO 'username';

How to create a database from the command line

CREATE DATABASE dbname;

To start working with the newly created database, execute the query:

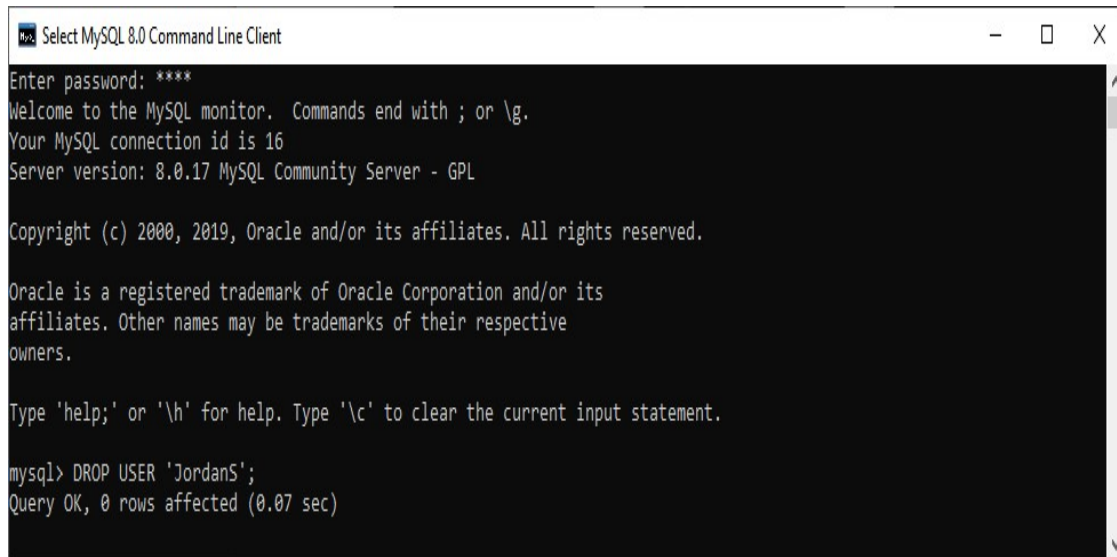
USE dbname;

How to delete a MySQL database from the command line.

DROP DATABASE dbname;

How to delete a MySQL user account

DROP USER 'username';



```
Select MySQL 8.0 Command Line Client
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.17 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current input statement.

mysql> DROP USER 'JordanS';
Query OK, 0 rows affected (0.07 sec)
```

MySQL Client options and query syntax

To get a full list of MySQL Client commands, enter help or \h at the prompt.

Using phpMyAdmin

phpMyAdmin is a GUI-based application which is used to manage MySQL database.

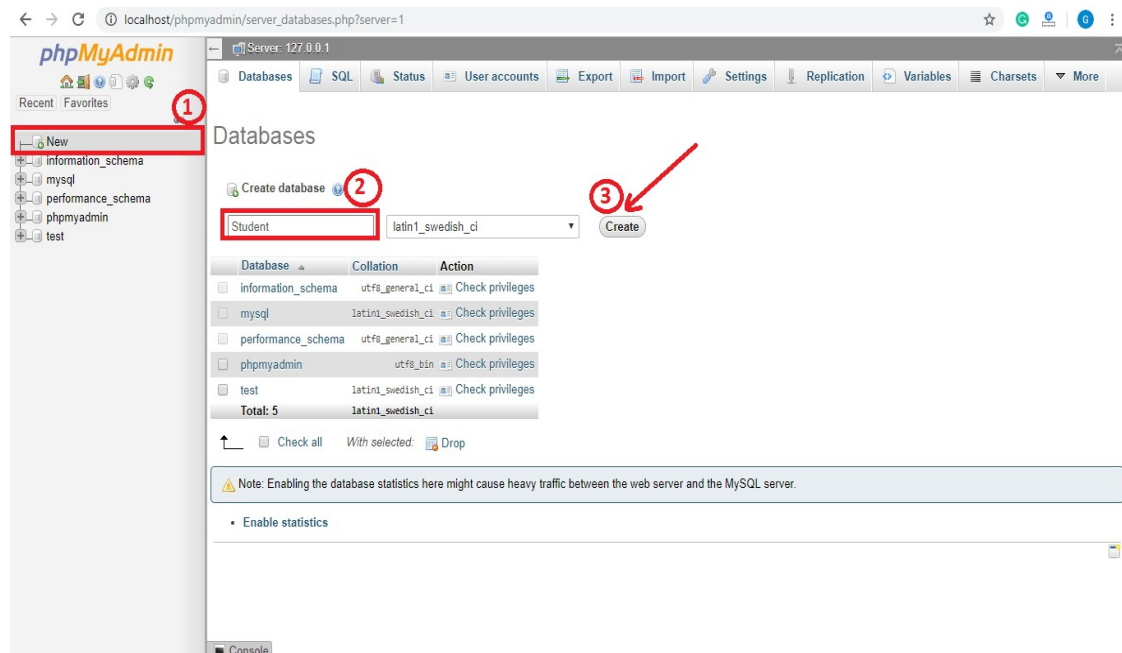
Features of phpMyAdmin

- phpMyAdmin can create, alter, browse, and drop databases, views, tables, columns, and indexes.
- It can display multiple results sets through queries and stored procedures.
- phpMyAdmin use stored procedure and queries to display multiple results sets.
- It supports foreign keys and InnoDB tables.
- phpMyAdmin can track the changes done on databases, views, and tables.
- We can also create PDF graphics of our database layout.
- phpMyAdmin can be exported into various formats such as XML, CSV, PDF, ISO/IEC 26300 - OpenDocument Text and Spreadsheet.
- It supports mysqli, which is the improved MySQL extension.
- phpMyAdmin can interact with 80 different languages.

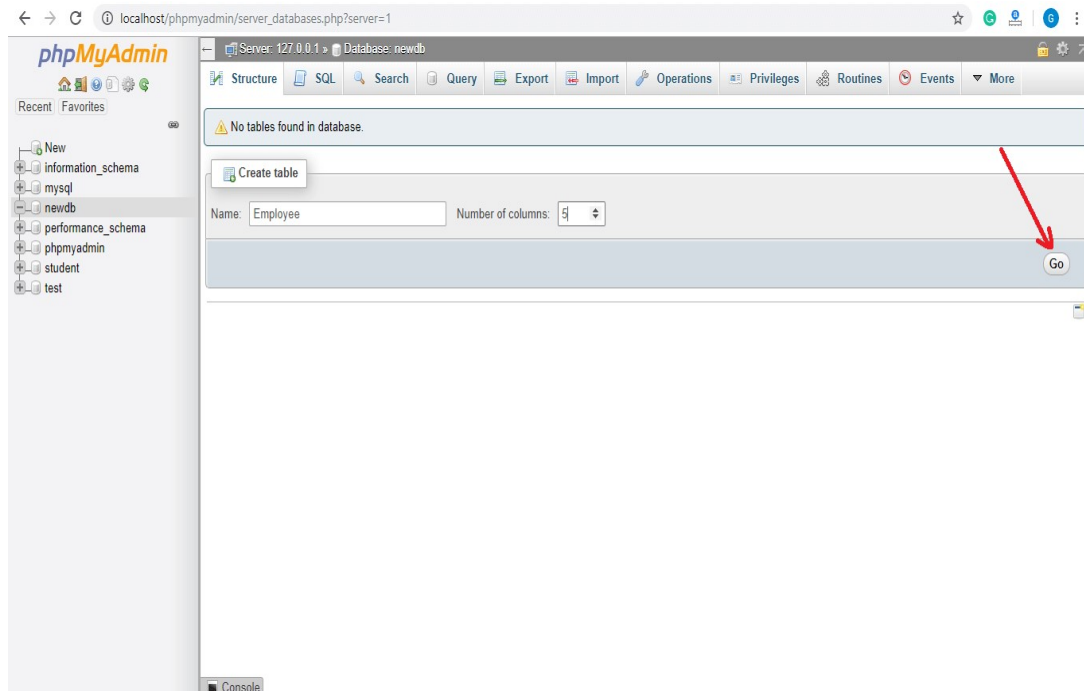
- phpMyAdmin can edit, execute, and bookmark any SQL-statements and even batch-queries.
- By using a set of pre-defined functions, it can transform stored data into any format. For example - BLOB-data as image or download-link.
- It provides the facility to backup the database into different forms.

How to work with phpMyAdmin?

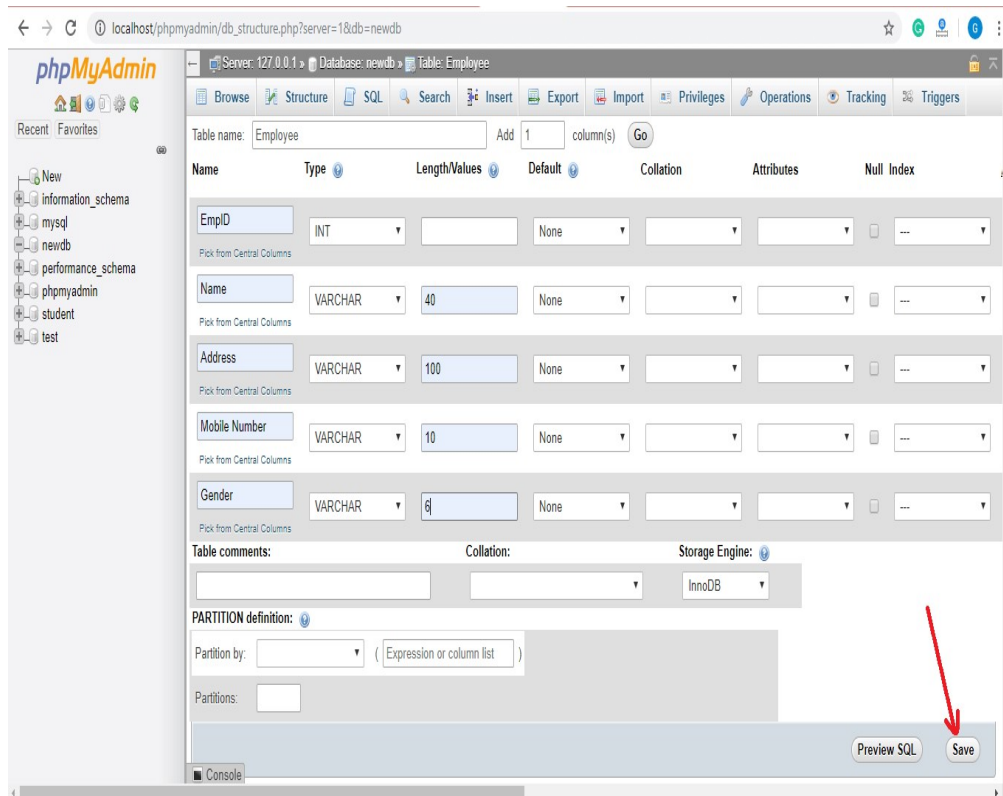
Click on New (1) to create a database and enter the database name in Create database (2) field and then click on Create (3) button. We can create any number of databases.



Enter the table name, number of columns, and click on Go. A message will show that the table is created successfully.



Now enter the field name, type, their size, and any constraint here and save it.



The table is created successfully. We can make changes in the table from here.

Server: 127.0.0.1 » Database: newdb » Table: employee

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	EmpID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	Name	varchar(40)	latin1_swedish_ci		No	None			Change Drop More
3	Address	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
4	Mobile Number	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
5	Gender	varchar(6)	latin1_swedish_ci		No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Fulltext Add to central columns Remove from central columns

Print Propose table structure Track table Move columns Normalize

Add 1 column(s) after Gender Go

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	EmpID	0	A	No	

Create an index on 1 columns Go

Partitions

No partitioning defined!

Console

Using PHP with MySQL:

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

- MySQLi extension (the "i" stands for improved)
- PDO (PHP Data Objects)

Connecting to MySQL:

The function used to connect PHP to MYSQL :

Procedure oriented

mysqli.connect(servername,username,password,databasename)

- Database name is optional

eg.,

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Object oriented

mysqli(servername,username,password,dbname)

- Dbname is optional

eg.,

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = mysqli($servername, $username, $password);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

PDO: This command requires valid database to connect to. If no database is specified, an exception is thrown.

Eg.,

```
<?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();
}
?>
```

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

MySQLi Object-Oriented:

```
$conn->close();
```

MySQLi Procedural:

```
mysqli_close($conn);
```

PDO:

```
$conn = null;
```

PHP MySQL Create Database

Example (MySQLi Object-oriented)

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

```
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}
```

```
mysqli_close($conn);
```

```
?>
```

Example (PDO)

```
<?php
```

```
$servername = "localhost";
```

```
$username = "username";
```

```
$password = "password";
```

```
try {
```

```
    $conn = new PDO("mysql:host=$servername", $username, $password);
```

```
    // set the PDO error mode to exception
```

```
    $conn->setAttribute(PDO::ATTR_ERRMODE,  
PDO::ERRMODE_EXCEPTION);
```

```
    $sql = "CREATE DATABASE myDBPDO";
```

```
    // use exec() because no results are returned
```

```
    $conn->exec($sql);
```

```
    echo "Database created successfully<br>";
```

```
} catch(PDOException $e) {
```

```
    echo $sql . "<br>" . $e->getMessage();
```

```
}
```

```
$conn = null;
```

```
?>
```

PHP MySQL Create Table

Example

```
<?php
```

```
$host = 'localhost:3306';
```

```
$user = ";
```

```
$pass = ";
```

```
$dbname = 'test';
```

```
$conn = mysqli_connect($host, $user, $pass,$dbname);
```

```
if(!$conn){
```

```
    die('Could not connect: '.mysqli_connect_error());
```

```
}
```

```
echo 'Connected successfully<br/>';
```



```
$sql = "create table emp5(id INT AUTO_INCREMENT,name VARCHAR(20)
NOT NULL,
emp_salary INT NOT NULL,primary key (id))";
if(mysqli_query($conn, $sql)){
    echo "Table emp5 created successfully";
}else{
    echo "Could not create table: ". mysqli_error($conn);
}
mysqli_close($conn);
?>
```

PHP MySQL Insert Data

Example

```
<?php
$host = 'localhost:3306';
$user = "";
$pass = "";
$dbname = 'test';
$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
    die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';
$sql = 'INSERT INTO emp4(name,salary) VALUES ("sonoo", 9000)';
if(mysqli_query($conn, $sql)){
    echo "Record inserted successfully";
}else{
    echo "Could not insert record: ". mysqli_error($conn);
}
mysqli_close($conn);
?>
```

PHP MySQL Update Record

Example

```
<?php
$host = 'localhost:3306';
$user = "";
$pass = "";
$dbname = 'test';
```

```
$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
    die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$id=2;
$name="Rahul";
$salary=80000;
$sql = "update emp4 set name=\"$name\", salary=$salary where id=$id";
if(mysqli_query($conn, $sql)){
    echo "Record updated successfully";
}else{
    echo "Could not update record: ". mysqli_error($conn);
}
mysqli_close($conn);
?>
```

PHP MySQL Delete Record

Example

```
<?php
$host = 'localhost:3306';
$user = "";
$pass = "";
$dbname = 'test';

$conn = mysqli_connect($host, $user, $pass,$dbname);
if(!$conn){
    die('Could not connect: '.mysqli_connect_error());
}
echo 'Connected successfully<br/>';

$id=2;
$sql = "delete from emp4 where id=$id";
if(mysqli_query($conn, $sql)){
    echo "Record deleted successfully";
}
```

```
}else{  
echo "Could not deleted record: ". mysqli_error($conn);  
}  
mysqli_close($conn);  
?>
```

PHP MySQL Select Query

```
<?php  
$host = 'localhost:3306';  
$user = "";  
$pass = "";  
$dbname = 'test';  
$conn = mysqli_connect($host, $user, $pass,$dbname);  
if(!$conn){  
    die('Could not connect: '.mysqli_connect_error());  
}  
echo 'Connected successfully<br/>';
```

```
$sql = 'SELECT * FROM emp4';  
$retval=mysqli_query($conn, $sql);  
  
if(mysqli_num_rows($retval) > 0){  
    while($row = mysqli_fetch_assoc($retval)){  
        echo "EMP ID : {$row['id']} <br> ".  
            "EMP NAME : {$row['name']} <br> ".  
            "EMP SALARY : {$row['salary']} <br> ".  
            "-----<br>";  
    } //end of while  
}else{  
    echo "0 results";  
}  
mysqli_close($conn);  
?>
```

Output:

```
Connected successfully  
EMP ID :1  
EMP NAME : ratan
```

EMP SALARY : 9000

EMP ID :2

EMP NAME : karan

EMP SALARY : 40000

EMP ID :3

EMP NAME : jai

EMP SALARY : 90000

mysqli_query(\$conn, \$sql): The mysqli_query() function executes/performs the given query on the database. For SELECT query this function returns the rows retrieved as an object if the execution is successful.

Syntax

mysqli_query(\$con, query)

1 **con(Mandatory)**

This is an object representing a connection to MySQL Server.

2 **query(Mandatory)**

This is a string value representing the query to be executed.

mode(Optional)

3 This is a integer value representing the result mode. You can pass *MYSQLI_USE_RESULT* or *MYSQLI_STORE_RESULT* as values to this parameter.

mysqli_num_rows(\$retval): returns number of tuples stored in the object retval or num of rows in the result set if a query.

```
<?php
```

```
$host = 'localhost:3306';
```

```
$user = '';
```

```
$pass = '';
```

```
$dbname = 'test';
```

```
$conn = mysqli_connect($host, $user, $pass,$dbname);
```

```
if(!$conn){
```

```
    die('Could not connect: '.mysqli_connect_error());
```

```
}
```

```
echo 'Connected successfully<br/>';
```

```
$sql = 'SELECT * FROM emp4';  
$retval=mysqli_query($conn, $sql);  
$num= mysqli_num_rows($retval);  
Echo $num;  
mysqli_close($conn);  
?>
```

mysqli_fetch_assoc(\$retval)

Fetch a result row as an associative array:

Eg.,

```
<?php
```

```
$con = mysqli_connect("localhost","my_user","my_password","my_db");  
if (mysqli_connect_errno()) {  
    echo "Failed to connect to MySQL: " . mysqli_connect_error();  
    exit();  
}
```

```
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";  
$result = mysqli_query($con, $sql);
```

```
// Associative array
```

```
$row = mysqli_fetch_assoc($result);  
printf ("%s (%s)\n", $row["Lastname"], $row["Age"]);
```

```
// Free result set
```

```
mysqli_free_result($result);
```

```
mysqli_close($con);
```

```
?>
```