Introduction to PHP

PHP is one of the most widely used server side scripting language for web development. Popular websites like Facebook, Yahoo, Wikipedia etc are developed using PHP.

PHP is so popular because it's very simple to learn, code and deploy on server, hence it has been the first choice for beginners since decades.

What is PHP?

PHP stands for **Hypertext Pre-Processor**. PHP is a scripting language used to develop static and dynamic webpages and web applications. Here are a few important things you must know about PHP:

1. PHP is an Interpreted language, hence it doesn't need a compiler.
2. To run and execute PHP code, we need a Web server on which PHP must be installed.
3. PHP is a server side scripting language, which means that PHP is executed on the server and the result is sent to the browser in plain HTML.
4. PHP is open source and free.

Is PHP the right language?

If you are still confused about whether you should learn PHP or is PHP the right language for your web project, then here we have listed down some of the features and usecases of PHP language, which will help you understand how simple yet powerful PHP scripting language is and why you should learn it.

1. PHP is **open source** and **free**, hence you can freely download, install and start developing using it.
2. PHP has a very **simple and easy to understand syntax**, hence the learning curve is smaller as compared to other scripting languages like JSP, ASP etc.
3. PHP is **cross platform**, hence you can easily develop and move/deploy your PHP code/project to almost all the major operating systems like Windows, Linux, Mac OSX etc.
4. All the popular **web hosting services support PHP**. Also the web hosting plans for PHP are generally the amongst the cheapest plans becasue of its popularity.
5. Popular Content Management Systems like **Joomla**, **Drupal** etc are developed in PHP and if you want to start your own website, you can easily do that with PHP.
6. With PHP, you can create static and dynamic webpages, perform file handling operations, send emails, access and modify browser cookies, and almost everything else that you might want to implement in your web project.
7. PHP is **fast** as compared to other scripting languages like JSP and ASP.
8. PHP has in-built support for MYSQL, which is one of the most widely used Database management system.

Uses of PHP

To further fortify your trust in PHP, here are a few applications of this amazing scripting language:

1. It can be used to **create Web applications** like Social Networks(Facebook, Digg), Blogs(Wordpress, Joomla), eCommerce websites(OpenCart, Magento etc.) etc.
2. **Comman Line Scripting**. You can write PHP scripts to perform different operations on any machine, all you need is a PHP parser for this.
3. **Create Facebook applications** and easily integrate Facebook plugins in your website, using Facebook's PHP SDK. Check this link for more information.
4. **Sending Emails** or building email applications because PHP provides with a robust email sending function.
5. Wordpress is one of the most used blogging(CMS) platform in the World, and if you know PHP, you can try a hand in **Wordpress plugin development**.

Install PHP on your Local Machine

To run PHP scripts on any machine(server, computer etc) we need PHP installed on it.

Requirements for PHP

To run PHP scripts, we need the following services:

1. **PHP Parser**: To execute PHP scripts, PHP installation is required.
2. **Web Server**: Because PHP is mostly used to develop websites, hence most of its implementations comes bundled with Apache Web Server, which is required to host the application developed in PHP over HTTP.
3. **Database**: Any one database management system, which is generally MySQL, as PHP comes with a native support for MYSQL.

Now, you can install all the 3 services separately yourself, or you can simply download and install softwares which automatically installs all the above services.

The most popular such software package is **XAMPP**.

What is XAMPP?

XAMPP stands for:

X: Cross Platform, as it supports all the moder operating systems like Windows, Mac OSX, Linux etc.

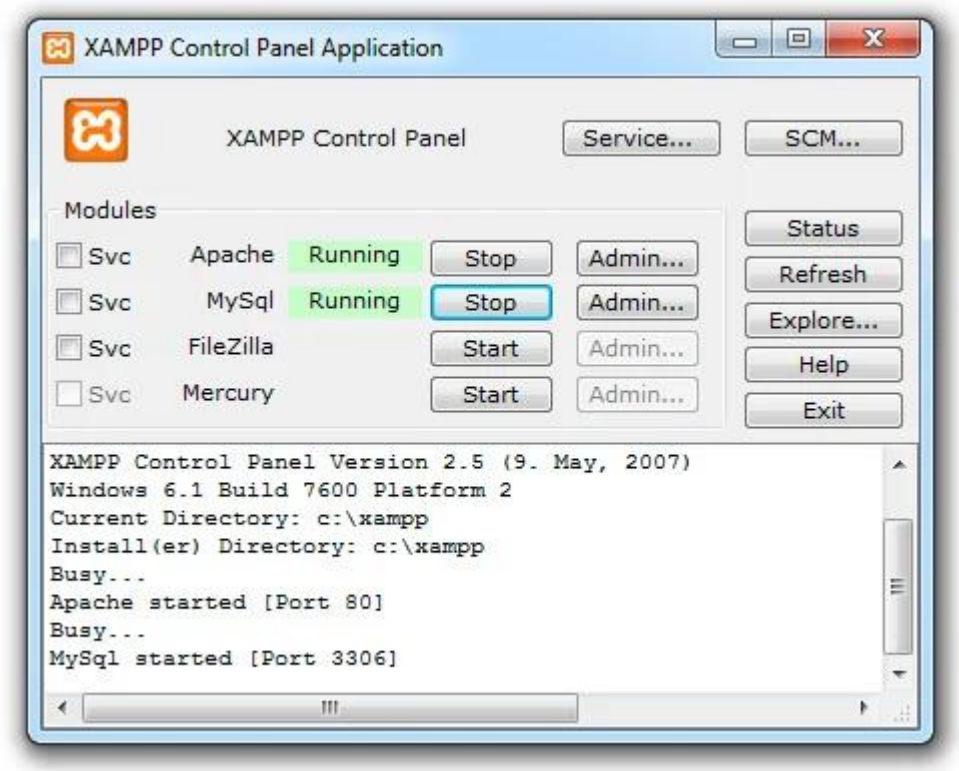
A: Apache Web Server

M: MySQL database management system.

P: PHP installation

P: Perl scripting language

You can easily download and install XAMPP .

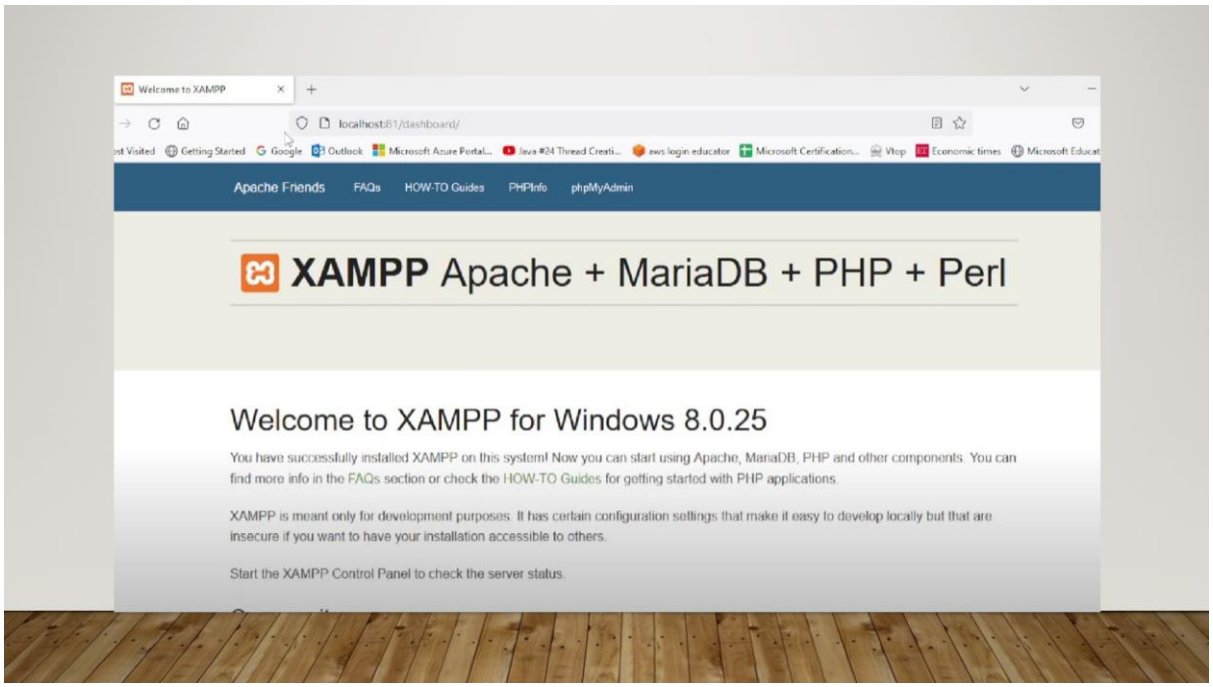


You can easily control, stop and restart, various services using the XAMPP Control Panel.

Upon successful installation, a folder with name **xampp** will be created in the C drive(by default). In the folder **xampp** there are many sub-folders like apache, cgi-bin, FileZillaFTP etc, but the most important sub-folders are:

1. **htdocs**: This is the folder in which we will keep all our PHP files.
2. **mysql**: This folder contains all the files for the MySQL database. By default the MySQL databse runs on port number 3306.
3. **php**: This folder holds all the installation files for PHP. All the configurations for the current PHP installation is saved in **php.ini** file which is stored in this folder.

If everything seems fine and the apache server is running(check in the XAMPP control panel), open your **Web browser** and enter `localhost` in the address bar and hit enter. You will see the default page of Apache web server.



Other Software Packages

XAMPP is not the only available package, although it is the most widely used one. Here are a few other Operating System specific options that you can download and install:

- **WAMP**: It's for Windows (Window, Apache, MySQL and PHP)
- **MAMP**: It's for Mac OSX (Macintosh, Apache, MySQL and PHP)
- **LAMP**: It's for Linux (Linux, Apache, MySQL and PHP)

PHP Example

Hello World script in PHP

All the php code is written inside php code tags which is `<?php` and `?>`. And the file with php code is saved with an extension `.php`. Here is a simple example:

```
<?php

// code statements

?>
```

Hence a simple **Hello, World!** program in PHP will be:

```
<?php

echo "Hello, World!";

?>
```

```
Hello, World!
```

`echo` is a command used in PHP to display anything on screen.

If we want to include this php code in an HTML document, we can do so like this:

```
<!DOCTYPE>

<html>

    <body>
```

```
    <?php

        echo "<h1>Hello, World!</h1>";

    ?>

    </body>

</html>
```

Code Editor or IDE and save the file with the name **filename.php**

Now go to your C directory where XAMPP was installed, and open **xampp** → **htdocs** and create a new folder with name **folderfilename** and put your php code file **filename.php** in the **folderfilename** folder.

Now visit the following link in your browser: **localhost/ folderfilename /filename.php** and you would see **Hello, World!** written on the screen. Notice that *Hello, World!* is written as a heading because in the HTML code we specified that *Hello, World!* should be printed as a heading as put it inside the heading tag `<h1>`

PHP Syntax Rules

Below we have a listed down the main syntax rules that you must follow while writing php code.

1. All the php code in a php script should be enclosed within `<?php` and `?>`, else it will not be considered as php code. Adding php code inside the PHP tags is known as **Escaping to php**.

```
<?php   ...    ?>
```

   Apart from the standard `<?php` and `?>`, you can also use the **Short-open tags**:

```
<?   ...    ?>
```

   Or use the HTML script tags, like we do for adding javascript code in HTML document:

```
<script language="PHP">   ...    </script>Every expression in PHP ends with a semicolon ;
```

2. **Commenting PHP code**: Both single line and multi-line comments are supported in PHP. For single line comment, we can either use `#` or `//` before the comment line. For example,

```
3. <?php
4.      # This is also a single line comment
5.      # another line of comment
6.
```

```
7.      // This is a single line comment

8.      echo "Example for Single line Comments";
```

```
?>
```

And for multi-line comments, we use /* … */. For example,

```
<?php
    /*
        This is also a single line comment
        another line of comment
    */
    echo "Example for Multi-line Comments";
?>
```

9. **PHP is case sensitive**, which means that a variable $tiger is not same as $Tiger. Both of these, represent two different variables here.

   But all the predefined keywords and functions like if, else, echo etc are case insensitive.

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

```
    ECHO "Hello, World!";

?>
```

```
Hello, World!

Hello, World!
```

10. PHP uses **curly braces to define a code block**.

```
<?php

    if($zero == 0)

    {

        echo "If condition satisfied";

        echo "This is a code block";

    }

?>
```

Variables in PHP

When we want to store any information(data) on our computer/laptop, we store it in the computer's memory space. Instead of remembering the complex address of that memory space where we have stored our data, our operating system provides us with an option to create folders, name them, so that it becomes easier for us to find it and access it.

Similarly, in any programming/scripting language, when we want to use some data value in our program/script, we can store it in a memory space and name the memory space so that it becomes easier to access it. The name given to the memory space is called a **Variable**.

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

Syntax:

```
<?php

    $variableName = value;

?>
```

Creating a Variable

In the example below we have create a different types of variables:

```
<?php

    $str = "I am a string";
```

```
    $int = 4;

    $float = 3.14;

    // You can name your variable anything

    $wow = "Wow!";

?>
```

In PHP, you don't have to declare the variable first and then use it, like it's done in Java, C++ etc, but in PHP the variable is created at the moment you assign it a value, like Python.

Also, in PHP we do not have to specify the type of data that we will be storing in a variable. You can create a variable and save any type of data in it. Hence PHP is quite **loosely typed language**.

Rules for creating Variables in PHP

Here we have a few basic rules that you must keep in mind while creating variables in PHP. All the rules are explained with help of simple examples.

1. A variable name will always start with a `$` sign, followed by the variable name.
2. A variable name should not start with a numeric value. It can either start with an alphabet or an underscore sign `_`.

```
3. <?php

4.      $var = "I am a variable";
```

```
5.      $_var = 5;
6.      // Invalid variable name
7.      $7var = "I am a variable too";
    ?>
```

8. A variable name can only contain alphabets, numbers and underscore symbol _.
9. Variable names in PHP are case-sensitive, which means $love is not same as $Love.

```
10.<?php
11.     $car = "Jaguar E-Pace";
12.     echo "My favorite car is $car";
13.     // below statement will give error
14.     echo "I love $Car";
    ?>
```

```
My favorite car is Jaguar E-Pace
Notice: Undefined variable: Car
```

PHP echo and print functions

two of the most important methods of PHP which are not built-in functions of PHP language, but they are **language constructs**.

```
<?php

    echo "Hello, I am a language construct";

?>
```

You must be wondering, **What is a language construct?** A language construct is accepted/executed by the PHP parser as it is, in other words the PHP parser doesn't have to parse and modify a language construct to execute it, as it is ready for execution by default. Hence, language constructs are faster than any built-in functions.

PHP Echo

echo() function is used to print or output one or more strings. We have specifically mentioned string here because, the syntax of the echo function is:

```
echo(string)
```

Although you can use echo() function to output anything, as PHP parser will automaticallly convert it into string type.

echo doesn't need parenthesis, although you can use parenthesis if you want.

```php
<?php

    echo "I am open";

    echo ("I am enclosed in parenthesis");

?>
```

Copy

```
I am open

I am enclosed in parenthesis
```

Printing a basic sentence

We have already covered it multiple times, still:

```php
<?php

    echo "I am a sentence";

?>
```

Copy

```
I am a sentence
```

Printing multiple strings using comma ,

Here is how you can use multiple strings as parameters for `echo`.

```php
<?php

    echo 'This','is','a','broken','sentence';

?>
```

```
This is a broken sentence
```

Printing a multiline text(string)

```php
<?php
    echo "This is a
     multiline sentence
     example";
?>
```

```
This is a multiline sentence example
```

Printing a string variable

```php
<?php
    $str = "I am a string variable";
    echo $str;
?>
```

```
I am a string variable
```

Printing a string variable with some text

Below we have explained how using **double quotes** and **single quotes** leads to different output when we use a string variable with some plain text in echo.

```
<?php

    $weird = "Stupid";

    echo "I am $weird";

    echo 'I am $weird';

?>
```

```
I am Stupid

I am $weird
```

As you can see, when we use **double quotes** the value of the string variable gets printed, while if we use **single quotes**, the variable is printed as it is.

Escaping special characters

As seen in previous examples, a **double quote** is required by echo to confirm what has to be printed. But what if you want to print the **double quotes** too? In such cases, we use an **escape sequence** to escape special characters from their special meanings. In PHP, a backslash \ is used to escape special characters.

Below we have a simple example:

```
<?php

    echo "Hello, this is a \"beautiful\" picture";
```

```
?>
```

```
Hello, this is a "beautiful" picture
```

As you can see, the **double quotes** are printed in the output.

PHP Print

The PHP `print` is exactly the same as `echo`, with same syntax and same usage. Just replace `echo` with `print` in all the above examples, and they will work just fine.

Data Types in PHP 5

PHP Data types specify the different types of data that are supported in PHP language. There are total 8 data types supported in PHP, which are categorized into 3 main types. They are:

1. **Scalar Types**: `boolean`, `integer`, `float` and `string`.
2. **Compound Types**: `array` and `object`.
3. **Special Types**: `resource` and `NULL`.

PHP Boolean

A boolean data type can have two possible values, either **True** or **False**.

```
$a = true;

$b = false;
```

NOTE: Here the values true and false are not enclosed within quotes, because these are not strings.

PHP Integer

An Integer data type is used to store any non-decimal numeric value within the range -2,147,483,648 to 2,147,483,647.

An integer value can be negative or positive, but it cannot have a decimal.

```
$x = -2671;

$y = 7007;
```

PHP Float

Float data type is used to store any decimal numeric value.

A float(floating point) value can also be either negative or positive.

```
$a = -2671.01;

$b = 7007.70;
```

PHP String

String data type in PHP and in general, is a sequence of characters(or anything, it can be numbers and special characters too) enclosed within quotes. You can use single or double quotes.

```php
$str1 = "Hello";

$str2 = "What is your Roll No?";

$str3 = "4";



echo $str1;

echo "<br/>";

echo $str2;

echo "<br/>";

echo "Me: My Roll number is $str3";
```

```
Hello

What is your Roll No?

Me: My Roll number is 4
```

PHP NULL

NULL data type is a special data type which means **nothing**. It can only have one value, and that is NULL.

If you create any variable and do not assign any value to it, it will automatically have NULL stored in it.

Also, we can use NULL value to empty any variable.

```php
// holds a null value

$a;
```

```php
$b = 7007.70;

// we can also assign null value

$b = null;
```

PHP Constants

Constants are variables whose value cannot be changed. In other words, once you set a value for a constant, you cannot change it.

In PHP, there are two ways to define a constant:

1. Using the `define()` method.
2. Using the `const` keyword.

Another very important point to remember is that while naming a constant, **we don't have to use** `$` symbol with the constant's name.

Using `define()`

Below is the syntax for using the `define()` function to create a constant.

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

Parameters:

1. name: Name of the constant
2. value: Value of the constant
3. case-insensitive: Specifies whether the constant name is case sensitive or not. It's default value is false, which means, by default, the constant name is case sensitive.

Example

Below we have a simple example where we have not mentioned the parameter **case-insensitive**, hence it will automatically take the default value for that.

```
<?php

    define(OMG, "Oh! my God.");

    echo OMG;

?>
```

```
Oh! my God.
```

For the above constant definition, if we try to use the following statement we will get an error, because the constant OMG is case sensitive.

```
<?php

    define(OMG, "Oh! my God.");

    echo omg;
```

```
?>
```

```
omg
```

`echo` will consider it as a string and will print it as it is, with a subtle **NOTICE** saying that the string variable is not defined.

Now let's see another example where we will specify the **case-insensitive** parameter.

```
<?php

    define(OMG, "Oh! my God.", true);

    echo omg;

?>
```

```
Oh! my God.
```

Using the `const` Keyword

We can also define constants in PHP using the `const` keyword. But we can only use the `const` keyword to define scalar constants, i.e. only integers, floats, booleans and strings, while `define()` can be used to define array and resource constants as well, although they are not used oftenly.

Example

```
<?php

    const OMG = "Oh! my God.";
```

```
    echo OMG;

?>
```

```
Oh! my God.
```

When we define a constant using the `const` keyword, the constant name is always case sensitive.

PHP Operators

Operators are used to perform operations on PHP variables and simple values.

In PHP there are total 7 types of operators, they are:

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Increment/Decrement Operators
5. Logical Operators
6. String Operators
7. Array Operators

There are a few additional operators as well like, Type operator, Bitwise operator, Execution operators etc.

Based on how these operators are used, they are categorised into 3 categories:

1. **Unary Operators**: Works on a single operand(variable or value).

2. **Binary Operators**: Works on two operands(variables or values).
3. **Ternary Operators**: Works on three operands.

PHP Arithmetic Operators

These operators are used to perform basic arithmetic operations like addition, multiplication, division, etc.

Name	Operator	What does it do?	Example
Addition	+	It is used to perform normal addition.	$a + $b
Subtraction	-	It is used to perform normal subtraction.	$a - $b
Multiplication	*	It is used to perform multiplication.	$a * $b
Division	/	It is used to perform division.	$a / $b
Exponent	**	It returns the first operand raised to the power the second operand. $a ** $b = ab	$a ** $b

Modulus(or, Remainder)	%	It returns the remainder of first operand divided by the second operand	$a % $b

---

PHP Assignment Operators

Assignment operators are used to assign values to variables, either as it is or after performing some arithmetic operation on it. The most basic assignment operator is **equal to**=.

Operator	Usage
=	$a = $b, will save the value of variable $b to the variable $a
+-	$a += $b is same as $a + $b
-=	$a -= $b is same as $a - $b
*=	$a *= $b is same as $a * $b
/=	$a /= $b is same as $a / $b
%=	$a %= $b is same as $a % $b

Name	Operator	What does it do?	Example
Equal	==	It returns true if left operand is equal to the right operand.	$a == $b
Identical	===	It returns true if left operand is equal to the right operand and they are of the same type.	$a === $b
Not Equal	!=	It returns true if left operand is not equal to the right operand.	$a != $b
Not Identical	!==	It returns true if left operand is not equal to the right operand, and they are of different type as well.	$a !== $b
Greater than	>	It returns true if left operand is greater than the right operand.	$a > $b
Less than	<	It returns true if left operand is less than the right operand.	$a < $b
Greater than or equal to	>=	It returns true if left operand is greater than or equal to the right operand.	$a >= $b

Less than or equal to	<=	It returns true if left operand is less than or equal to the right operand.	$a <= $b

So basically, the assignment operator provides us with shorthand techniques to perform arithmetic operations.

PHP Comparison Operators

As the name suggest, these are used to compare two values.

PHP Increment/Decrement Operators

These operators are **unary operators**, i.e they require only one operand.

Operator	Usage
++$a	**Pre Increment**, It will first increment the operand by 1(add one to it) and then use it or return it.
$a++	**Post Increment**, It will first return the operand and then increment the operand by 1.
--$b	**Pre Decrement**, It will first decrement the operand by 1(subtract one from it) and then use it or return it.

$b--	**Post Decrement**, It will first return the operand and then decrement the operand by 1.

These operators are very useful and handy when use loops or when we have simply increment any value by one in our program/script.

PHP Logical Operators

Logical operators are generally used when any action depends on two or more conditions.

Name	Operator	What does it do?	Example
And	and or &&	It returns true if both the operands(or expressions) returns true.	$a && $b
Or	or or ||	It returns true if any one out of the two operands(or expressions) returns true, or both return true.	$a || $b
Xor	xor	It returns true if any one out of the two operands(or expressions) returns true, but not when both return true.	$a xor $b
Not	!	This is a **unary operator**. It returns true, if the operand(or expression) returns false.	!$a

PHP String Operators

String operators are used to perform operations on string. There are only two string operators, generally PHP built-in functions are used to perform various operations on strings, we will learn about them in coming tutorials.

Name	Operator	What does it do?	Example
Concatenation	. (a dot)	It is used to concatenate(join together) two strings.	$a.$b
Concatenation Assignment	.=	It is used to append one string to another.	$a .= $b

Here we have a simple example to demonstrate the usage of both the string operators.

```php
<?php

    $a = "study";

    $b = "well";

    // concatenating $a and $b

    echo $a.$b;



    // appending $b to $a
```

```
    $a .= $b

    echo $a;

?>
```

```
studywell

studywell
```

PHP Array Operators

These operators are used to compare arrays.

Name	Operator	What does it do?	Example
Equal	==	It returns true if both the arrays have same key/value pairs.	$a == $b
Identical	===	It returns true if both the arrays have same key/value pairs, in same order and of same type.	$a === $b
Not Equal	!=	It returns true if both the arrays are not same.	$a != $b
Not Identical	!==	It returns true if both the arrays are not identical, based on type of value etc.	$a !== $b

Union(Join)	+	It joins the arrays together	$a + $b

PHP if, else and else if Conditional Statements

While writing programs/scripts, there will be scenarios where you would want to execute a particular statement only **if** some condition is satisfied. In such situations we use **Conditional statements**.

In PHP, there are 4 different types of Conditional Statements.

1. if statements
2. if...else statements
3. if...elseif...else statements
4. Switch statement

The if statement

When we want to execute some code when a condition is **true**, then we use if statement.

Syntax:

```
if(condition)

{

    // code to be executed if 'condition' is true
```

```
}
```

Here is a simple example,

```
<?php



$age = 20;



if($age <= 25)

{

    echo "You are not allowed to consume
alchohol";

}

?>
```

The if...else statement

When we want to execute some code when a condition is **true**, and some other code when that condition is **false**, then we use the if...else pair.

Syntax:

```
if(condition)

{

    // code to be executed if 'condition' is true
```

```
}

else

{

    // code to be executed if 'condition' is false

}
```

Here is a simple example,

```
<?php



$age = 26;



if($age <= 25)

{

    echo "You are not allowed to consume
alchohol";

}

else

{

    echo "Enjoy the drinks";

}

?>
```

```
Enjoy the drinks
```

The if...else...elseif statement

When we want to execute different code for different set of conditions, and we have more than 2 possible conditions, then we use if...elseif...else pair.

Syntax:

```
if(condition1)
{
    // code to be executed if 'condition1' is true
}
elseif(condition2)
{
    // code to be executed if 'condition2' is true
}
else
{
    /* code to be executed if both 'condition1'
     and 'condition2' are false */
}
```

Here is a simple example,

```php
<?php

// speed in kmph

$speed = 110;



if($speed < 60)

{

    echo "Safe driving speed";

}

elseif($speed > 60 && $speed < 100)

{

    echo "You are burning extra fuel";

}

else

{

    // when speed is greater than 100

    echo "Its dangerous";

}

?>
```

```
ts dangerous
```

In the example above, we have also used logical operator `&&`. Logical operators are very useful while writing multiple conditions together.

PHP Switch Statement

You must have used a lift which is used to go up and down a building, all you have to do is press the button with the floor number where you want to go, or a TV remote, using which you can change the channel on your TV just by selecting that channel number on the TV remote.

The `switch` statement

A `switch` statement is used to perform different actions, based on different conditions.

Using a `switch` statement, we can specify multiple conditions along with the code to be executed when that condition is **true**, thereby implementing a menu style program.

Syntax:

```
switch(X)

{

    case value1:

        // execute this code when X=value1

        break;
```

```
    case value2:
        // execute this code when X=value2
        break;
    case value3:
        // execute this code when X=value3
        break;
    ...
    default:
        /* execute this when X matches none of
           of the specified options */
}
```

You can specify as many options as you want using a single switch code block.

X can be a variable or an expression.

In a switch statement, we provide the deciding factor which can be a variable or an expression to our switch statement, and then we specify the different **cases**, each with a **value**, a piece of code and a **break** statement.

break statement is specified to break the execution of the switch statement once the action related to a specified value has been performed.

If we do not specify a break statement, then all the switch cases, after the matched case, will get executed, until the next break statement.

The **default** statement is executed if no matching case is there.

```php
<?php

$car = "Jaguar";



switch($car)

{

    case "Audi":

        echo "Audi is amazing";

        break;

    case "Mercedes":

        echo "Mercedes is mindblowing";

        break;

    case "Jaguar":

        echo "Jaguar is the best";

        break;

    default:

        echo "$car is Ok";
```

```
}

?>
```

```
Jaguar is the best
```

PHP while and do...while Loop

As the name suggests, a **Loop** is used to execute something over and over again.

For example, if you want to display all the numbers from 1 to 1000, rather than using `echo` statement 1000 times, or specifying all the numbers in a single `echo` statement with newline character `\n`, we can just use a loop, which will run for 1000 times and every time it will display a number, starting from 1, incrementing the number after each iteration or cycle.

In a Loop, we generally specify a condition or a LIMIT up till which the loop will execute, because if we don't specify such a condition, how will we specify when the loop should end and not go on for infinite time.

PHP while Loop

The `while` loop in PHP has two components, one is a condition and other is the code to be executed. It executes the given code until the specified condition is true.

Syntax:

```php
<?php

while(condition)

{

    /*

        execute this code till the

        condition is true

    */

}

?>
```

For example, let's take the problem mentioned in the beginning of this tutorial. Let's print numbers from 1 to 10.

```php
<?php


$a = 1;


while($a <= 10)

{

    echo "$a | ";

    $a++;    // incrementing value of a by 1
```

```
}

?>
```

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

Note: We have added a symbol | just to separate the numbers, it has no functional use in the above code.

PHP do...while Loop

The do...while loop is a little different from all the loops in PHP because it will execute at least one time, even if the condition is false, can you guess how? Well because the condition is checked after the loop's execution, hence the first time when the condition is checked, the loop has already executed once.

Syntax:

```
<?php

do {

    /*

        execute this code till the

        condition is true

    */

} while(condition)

?>
```

Let's implement the above example using do...while loop,

```php
<?php


$a = 1;


do {
    echo "$a | ";
    $a++;    // incrementing value of a by 1
} while($a <= 10)
?>
```

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

Let's take another example where even if the condition is **false**, still the loop will be executed once.

```php
<?php


$a = 11;


do {
```

```
    echo $a;

    $a++;    // incrementing value of a by 1

} while($a <= 10)

?>
```

```
11
```

As we can see clearly, that the condition in the above do...while loop will return **false** because value of variable $a is **11** and as per the condition the loop should be executed only if the value of $a is less than or equal to **10**.

PHP for and foreach Loop

PHP for Loop

The for loop in PHP doesn't work like while or do...while loop, in case of for loop, we have to declare beforehand how many times we want the loop to run.

Syntax:

```
<?php

for(initialization; condition;
increment/decrement)

{

    /*

        execute this code till the
```

```
        condition is true

    */

}

?>
```

The parameters used have following meaning:

1. **initialization**: Here we initialize a variable with some value. This variable acts as the loop counter.
2. **condition**: Here we define the condition which is checked after each iteration/cycle of the loop. If the condition returns **true**, then only the loop is executed.
3. **increment/decrement**: Here we increment or decrement the loop counter as per the requirements.

Example

Again, lets try to print numbers from 1 to 10, this time we will be using the for loop.

```
<?php



for($a = 1; $a <= 10; $a++)

{
```

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

}

?>
```

```
1

2

3

4

5

6

7

8

9

10
```

Nested for Loops

We can also use a for loop inside another for loop. Here is a simple example of nested for loops.

```
<?php



for($a = 0; $a <= 2; $a++)

{

    for($b = 0; $b <= 2; $b++)

    {
```

```
        echo "$b $a ";

    }

}


?>
```

Copy

```
0 0
1 0
2 0
0 1
1 1
2 1
0 2
1 2
2 2
```

PHP foreach Loop

The foreach loop in PHP is used to access key-value pairs of an array. This loop only works with arrays and you do not have to initialise any loop counter or set any condition for exiting from the loop, everything is done implicitly(internally) by the loop.

Syntax:

```
<?php

foreach($array as $var)
```

```
{
    /*
        execute this code for all the
        array elements

        $var will represent all the array
        elements starting from first element,
        one by one
    */
}
?>
```

Here is a simple example.

```
<?php

$array = array("Jaguar", "Audi", "Mercedes",
"BMW");

foreach($array as $var)
{
```

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

}


?>
```

```
Jaguar

Audi

Mercedes

BMW
```

PHP break statement

To recall, in switch code blocks, we used break statement to break out of the switch block when a valid case block gets executed.

Let's see an example for simple switch code:

```
<?php


$a = 1;


switch($a)

{

    case 1:

        echo "This is case 1";
```

```
        break;

    case 2:

        echo "This is case 2";

        break;

    default:

        echo "This is default case";

}


?>
```

```
This is case 1
```

But what if we forget to add the break statement at the end of each **case** block in the switch statement? In that case, the execution will still start from the matching **case**, but will not exit out of the switch statement and will keep on executing every code statement below it until the next break statement.

```
<?php


$a = 2;


switch($a)
```

```
{
    case 1:
        echo "This is case 1";
    case 2:
        echo "This is case 2";
    default:
        echo "This is default case";
}

?>
```

```
This is case 2
This is default case
```

Using break in Loops

In Loops, the break statement is very useful for situations when you want to exit out of the loop(stop the loop), if some condition is satisfied.

Let's take a simple example of a for loop to understand how we can use break statement in loops.

In the example below, we want to find the first number divisible by 13, which is between 1762 and 1800, starting from 1762.

```php
<?php


$x = 13;


for($i = 1762; $i < 1800; $i++)

{

    if($i % $x == 0)

    {

        echo "The number is $i";

        break;

    }

}


?>
```

```
The number is 1768
```

In the above script, we start our for loop from 1762, and run it till 1800. In every iteration/cycle, we check whether the incremented number is divisible by 13. When we find the first number for which the remainder is 0, we exit from the loop using the break statement.

The usage of break statement is same for all the different types of loops.

PHP Functions

What is a function: A function is a named block of code that performs a specific task.
→Why do you need functions in the first place?
Sometimes, you need to perform the same task multiple times in a script.

There are two types of functions.

1. Built-in functions.
2. User defined functions.

In PHP there are thousands of **built-in functions** which we can directly use in our program/script.

PHP User Defined Functions

Let's understand how we can define our own functions in our program and use those functions.

Syntax:

```php
<?php

function function_name()
{
    // function code statements
}

?>
```

Few Rules to name Functions

1. A **function name** can only contain alphabets, numbers and underscores. No other special character is allowed.
2. The name should start with either an alphabet or an underscore. It should not start with a number.
3. And last but not least, function names are not case-sensitive.
4. The opening curly brace `{` after the function name marks the start of the function code, and the closing curly brace `}` marks the end of function code.

Advantages of User-defined Functions

As we have already seen a simple example of using a function above, you must have understood how time-saving it can be for large programs. Here are a few advantages of using functions for you:

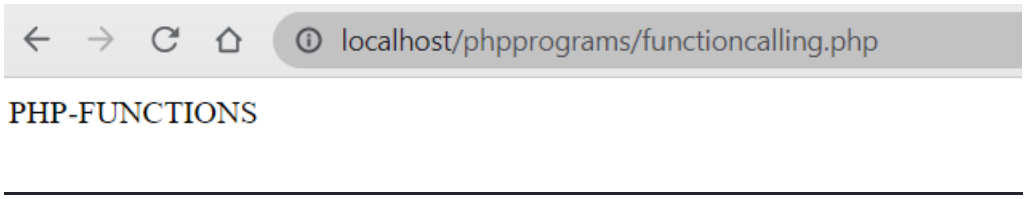
1. **Reuseable Code**: As it's clear from the example above, you write a function once and can use it for a thousand times in your program.
2. **Less Code Repetition**: For example, we just had one line of code in the function, but what if we have 10 lines of code. So rather than repeating all those lines of code over and over again, we can just create a function for them and simply call the function.
3. **Easy to Understand**: Using functions in your program, makes the code more readable and easy to understand.

Example for call a Function:

functioncalling.php

```php
<?php
// defining the function

function greetings()
{
    echo "PHP-FUNCTIONS";
}


// calling the function
greetings();



?>
```

OUTPUT:



PHP Function Arguments:

An argument is nothing but a variable.

Arguments are specified after the function name, in parentheses, separated by comma. When we define a function, we must define

the number of arguments it will accept and only that much arguments can be passed while calling the function.

Syntax:

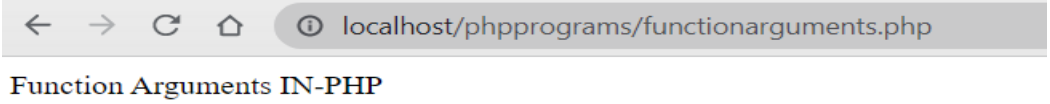
```php
<?php
/*
    we can have as many arguments as we
    want to have in a function
*/
function function_name(argument1, argument2)
{
    // function code statements
}

?>
```

Function Arguments Example:

functionarguments.php

```php
<?php
// defining the function with argument

function greetings($AWP)
{
    echo "Function Arguments $AWP";
}


greetings("IN-PHP");

?>
```

OUTPUT:



Function Arguments IN-PHP

PHP Default Function Arguments

Sometimes function arguments play an important role in the function code execution. In such cases, if a user forgets to provide the argument while calling the function, it might lead to some error.

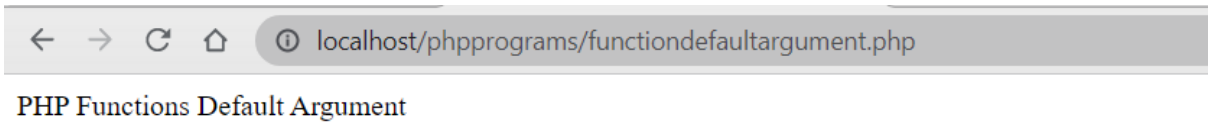
To avoid such errors, we can provide a default value for the arguments which is used when no value is provided for the argument when the function is called.

Example:

functiondefaultargument.php

```php
<?php

// defining the function with default argument
function greetings($AWP = "Default Argument")
{
    echo " PHP Functions $AWP";
}


greetings();

?>
```

OUTPUT:



PHP Functions Default Argument

PHP Function Returning Values

Yes, functions can even return results. When we have functions which are defined to perform some mathematical operation etc, we

would want to output the result of the operation, so we return the result.

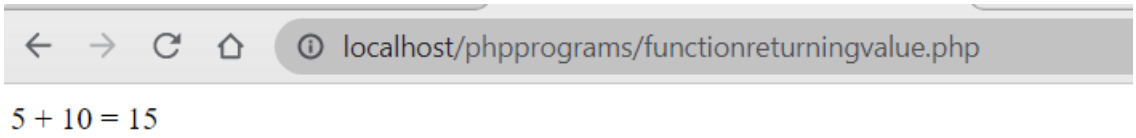
return statement is used to return any variable or value from a function in PHP.

Example.

functionreturningvalue.php

```php
<?php

function add($a, $b)
{
    $sum = $a + $b;
    // returning the result
    return $sum;
}

echo "5 + 10 = " . add(5, 10) . "";

?>
```

OUTPUT:



PHP Arrays

An **array** is used to store multiple values, generally of same type, in a single variable.

For example if you have a list of festivals that you want to store together, or may be a list of stationary items, or list of colors, then you can either keep them in separate variables, but then you will have to create a lot of variables and they won't be related to each other.

In such case, PHP arrays are used. **Arrays** can store multiple values together in a single variable and we can traverse all the values stored in the array using the foreach loop.

Creating an Array

We can create an array in PHP using the array() function.

Syntax:

```php
<?php
/*
    this function takes multiple values
    separated by comma as input to create
    an aray
*/
array();

?>
```

In PHP there are 3 types of array:

1. **Indexed or Numeric Array**: These are arrays with numeric index.
2. **Associative Array**: These are arrays which have a named key as index, the key can be numeric or text.

3. **Multidimensional Array**: These are arrays which contain one or more arrays.

Example

```php
<?php

/* a simple array with car names */

$carnames = array("BMW", "ERTIGA", "AUDI");

?>
```

Advantages of Array

Here are a few advantages of using array in our program/script:

1. It's very easy to define simple list of related data, rather than creating multiple variables.
2. It's super easy to use and traverse using the `foreach` loop.
3. PHP provides built-in functions for sorting array, hence it can be used for sorting information as well.

PHP Indexed Arrays or Numeric Arrays

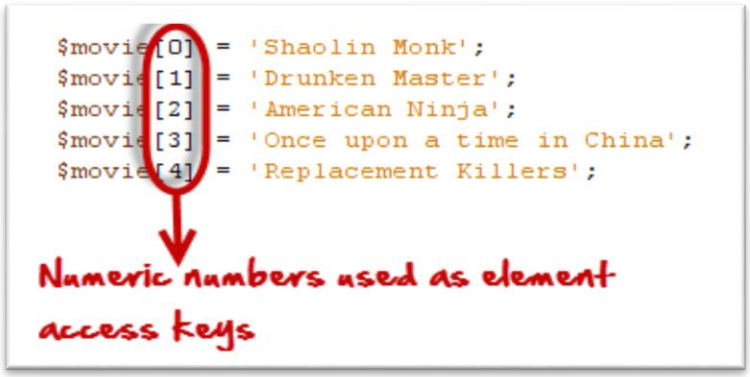
An **indexed array** is a simple array in which data elements are stored against numeric indexes. All the array elements are represented by an index which is a numeric value starting from `0` for the first array element.

Creating an Indexed Array

```
<?php
        $variable_name[n] = value;
?>
```

HERE,

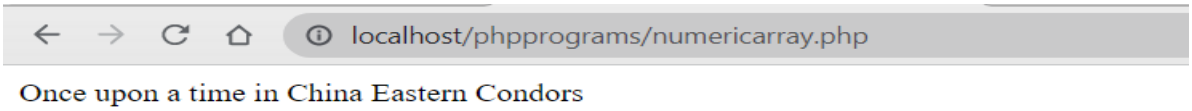
- "$variable_name…" is the name of the variable
- "[n]" is the access index number of the element
- "value" is the value assigned to the array element.



Example:

numericarray.php

```php
<?php
$movie[0]="Shaolin Monk";
$movie[1]="Drunken Master";
$movie[2]="American Ninja";
$movie[3]="Once upon a time in China";
$movie[4]="Replacement Killers";
echo $movie[3];
$movie[3] = " Eastern Condors";
echo $movie[3];
?>
```



2. PHP Associative Array

- Associative array differ from numeric array in the sense that associative arrays use **descriptive names** for id keys.
- Below is the syntax for creating associative array in php.

```
<?php
$variable_name['key_name'] = value;
                    or
$variable_name = array('keyname' => value);
?>
```

HERE,

- "$variable_name…" is the name of the variable
- "['key_name']" is the access index number of the element
- "value" is the value assigned to the array element.

Let's suppose that we have a group of persons, and we want to assign the gender of each person against their names.

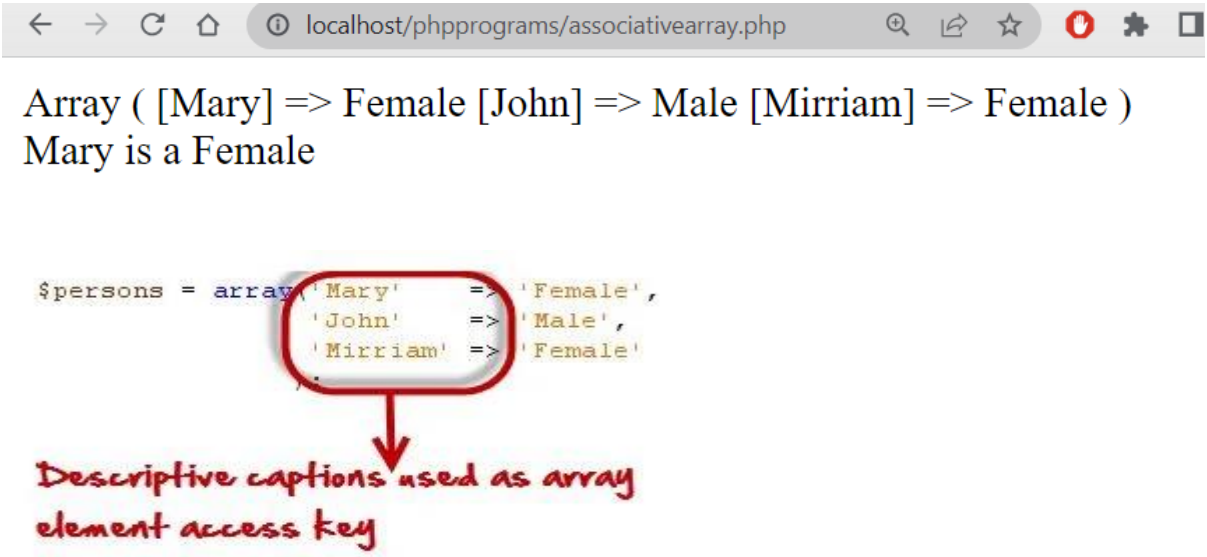
We can use an associative array to do that.

The code below helps us to do that.

associativearray.php

```php
<?php
$persons = array("Mary" => "Female",
                 "John" => "Male",
                 "Mirriam" => "Female");
print_r($persons);
echo "<br>";
echo "Mary is a " . $persons["Mary"];
?>
```

OUTPUT:



HERE,

- Associative array are also very useful when retrieving data from the database.
- The field names are used as id keys.

3. PHP Multi-dimensional arrays

These are arrays that contain other nested arrays.

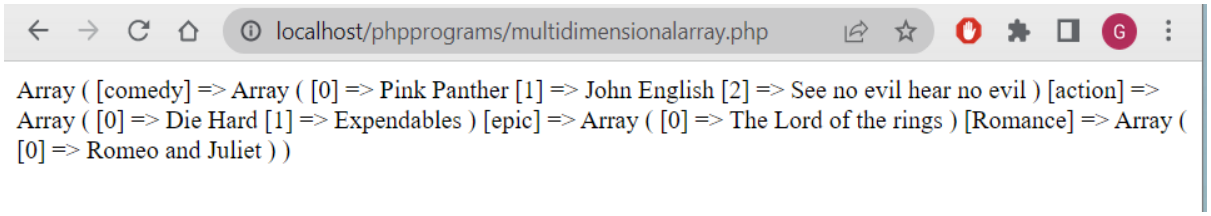
The advantage of multidimensional arrays is that they allow us to group related data together.

Example:

multidimensionalarray.php

```php
<?php
$movies =array(
"comedy"   => array("Pink Panther", "John English", "See no evil hear no evil"),
"action"   => array("Die Hard", "Expendables"),
"epic"     => array("The Lord of the rings"),
"Romance"  => array("Romeo and Juliet")
);
print_r($movies);
?>
```

OUTPUT:



Advantages of Multidimensional Array

Here are a few advantages of using multidimensional array in our program/script:

1. Detailed information can be stored in multidimensional array.
2. On top level, it can either be kept indexed or associative, which makes it more user-friendly, as they can use it as per their requirements.

PHP Array Functions

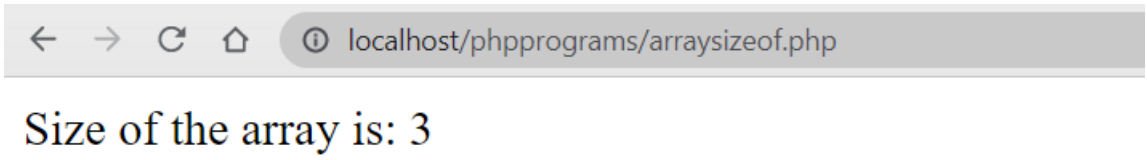
sizeof($arr)

This function returns the size of the array or the number of data elements stored in the array.

arraysizeof.php

```php
<?php

$length = array("functions", "arrays", "php");
echo "Size of the array is: ". sizeof($length);

?>
```

OUTPUT:



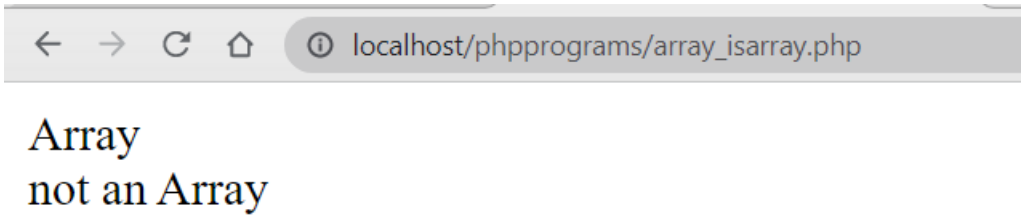
is_array($arr)

To check whether the provided data is in form of an array, we can use the is_array() function. It returns True if the variable is an array and returns False otherwise.

array_isarray.php

```php
<?php

$awp = array("php", "functions", "Arrays");

// using ternary operator
echo is_array($awp) ? 'Array' : 'not an Array';
echo "<br>";

$mysubject = "php";

// using ternary operator
echo is_array($mysubject) ? 'Array' : 'not an Array';

?>
```

OUTPUT:



in_array($var, $arr)

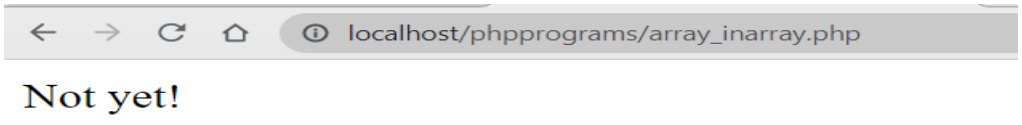
When using an array, we may often want to check whether a certain value is present in the array or not. For example, if get a list of certain cars, like we do in almost all our examples, to check if a certain car is added into the array, we can use the **in_array** function.

Example:

array_inarray.php

```php
<?php

$awp = array("php", "client", "server");

// new concept car by lamborghini
$concept = "DRUPAL";

echo in_array($concept, $awp) ? 'Added to the Lineup' : 'Not yet!'

?>
```

OUTPUT:



As we can see unlike the other functions above, this one takes **two arguments**, one is the value to be searched in the array, and the second one is the array itself.

print_r($arr)

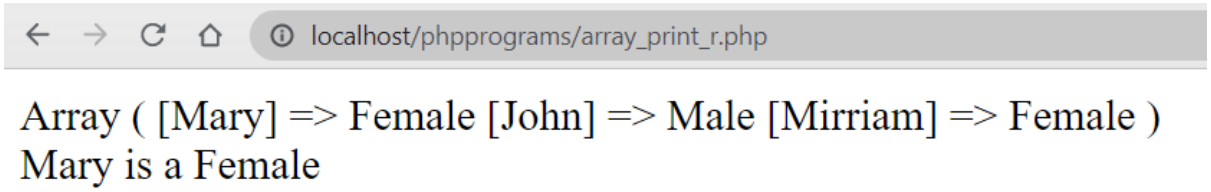
Although this is not an array function, but it deserves a special mention here, as we can use this function to print the array in the most descriptive way possible. This function prints the complete representation of the array, along with all the keys and values.

Simply, **print_r** prints human readable information of a variable.

array_print_r.php

```php
<?php
$persons = array("Mary" => "Female",
                 "John" => "Male",
                 "Mirriam" => "Female");
print_r($persons);
echo "<br>";
echo "Mary is a " . $persons["Mary"];
?>
```

OUTPUT:



array_merge($arr1, $arr2)

If you want to combine two different arrays into a single array, you can do so using this function. It doesn't matter whether the arrays

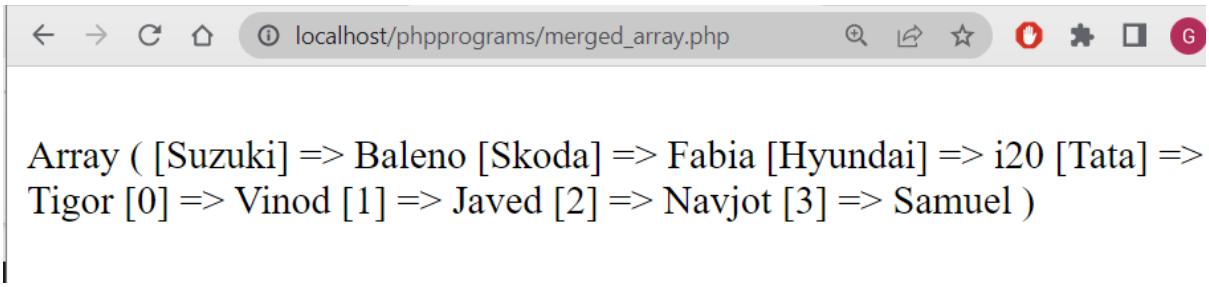
to be combined are of same type(indexed, associative etc) or different types, using this function we can combine them into one single array.

Let's take an example where we will merge an indexed array and an associative array.

merged_array.php

```php
<?php

$hatchbacks = array(
        "Suzuki" => "Baleno",
        "Skoda" => "Fabia",
        "Hyundai" => "i20",
        "Tata" => "Tigor"
    );

// friends who own the above cars
$friends = array("Vinod", "Javed", "Navjot", "Samuel");

// let's merge the two arrays into one
$merged = array_merge($hatchbacks, $friends);

print_r($merged);

?>
```

OUTPUT:



PHP Date and Time

The PHP `date()` function is used to format a date and/or a time.

The PHP Date() Function

The PHP `date()` function formats a timestamp to a more readable date and time.

Syntax

```
date(format,timestamp)
```

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

Get a Date

The required *format* parameter of the date() function specifies how to format the date (or time).

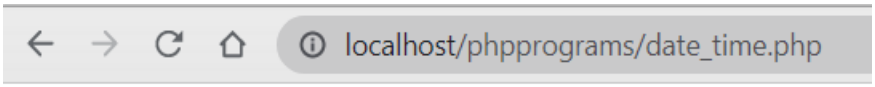
Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.
The example below formats today's date in three different ways:

date_time.php

```php
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```



Today is 2023/04/23
Today is 2023.04.23
Today is 2023-04-23
Today is Sunday

- **Introduction to PHP Superglobal Variables**

In PHP, some of the variables are predefined in the package itself for them to be used in inappropriate places. These predefined variables play a major role in webpage development in PHP. These variables are called "**superglobal**" variables.

The term globalization refers to the context that it can be accessed globally irrespective of the scope you can access it anywhere in the program. These variables are used in web page development in PHP and these variables have a predefined functionality and can be used whenever required.

List of PHP Superglobal Variables:

The superglobal variables in PHP can be accessed anywhere and anytime whenever required for its functionality.

The PHP super global variables are :

1) $_GET["FormElementName"]: It is used to collect value from a form(HTML script) sent with method='get'. information sent from a form with the method='get' is visible to everyone(it display on the browser URL bar).

2) **$_POST["FormElementName"]:** It is used to collect value in a form with method="post". Information sent from a form is invisible to others.(can check on address bar)

3) $_REQUEST["FormElementName"]: This can be used to collect data with both post and get method.

4) $_FILES["FormElementName"] : It can be used to upload files from a client computer/system to a server. OR
$_FILES["FormElementName"]["ArrayIndex"]: : Such as File Name, File Type, File Size, File temporary name.

5) $_SESSION["VariableName"]: A session variable is used to store information about a single user, and are available to all pages within one application.

6) **$_COOKIE["VariableName"]:** A cookie is used to identify a user. cookie is a small file that the server embedded on user computer.

7) $_SERVER["ConstantName"]: $_SERVER holds information about headers, paths, and script locations.

Eg
$_SERVER["SERVER_PORT"],$_SERVER["SERVER_NAME"],$_SERVER["REQUEST_URI"]

PHP Form Handling

When we develop a website or a web application, we often have to create forms to take input from users, like a **Login** form or a **Registration** form.

- **A form is an area that can contain form elements.**
- **Form elements are elements that allow the user to enter information (like text fields ,drop-down menus, radio buttons, checkboxes, etc.) in a form.**
- **A form is defined with the <form> tag.**

```
<form>
 <input>
 <input>
</form>
```

Input

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

1. Text Fields

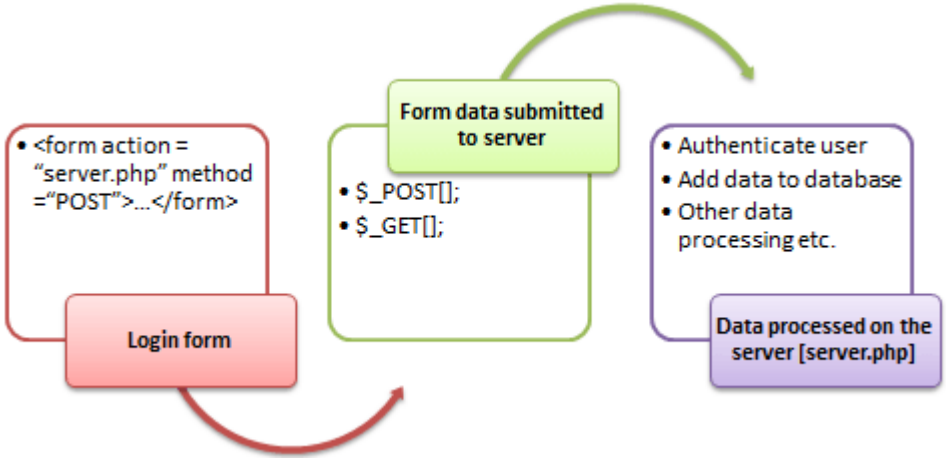
Text fields are used when you want the user to type letters, numbers, etc. in a form.

```
<form>
First name:
<input type="text" name="firstname">
<br>
Last name:
<input type="text" name="lastname">
</form>
```

PHP Registration Form using GET, POST Methods with Example

Form:

- When you login into a website or into your mail box, you are interacting with a form.
- Forms are used to get input from the user and submit it to the web server for processing.
- The diagram below illustrates the form handling process.



- A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.
- The form is defined using the <form>…</form> tags and GUI items are defined using form elements such as input.

→Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

1.PHP POST method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method is ideal when you do not want to display the form post values in the URL.
- A good example of using post method is when submitting login details to the server.

It has the following syntax.

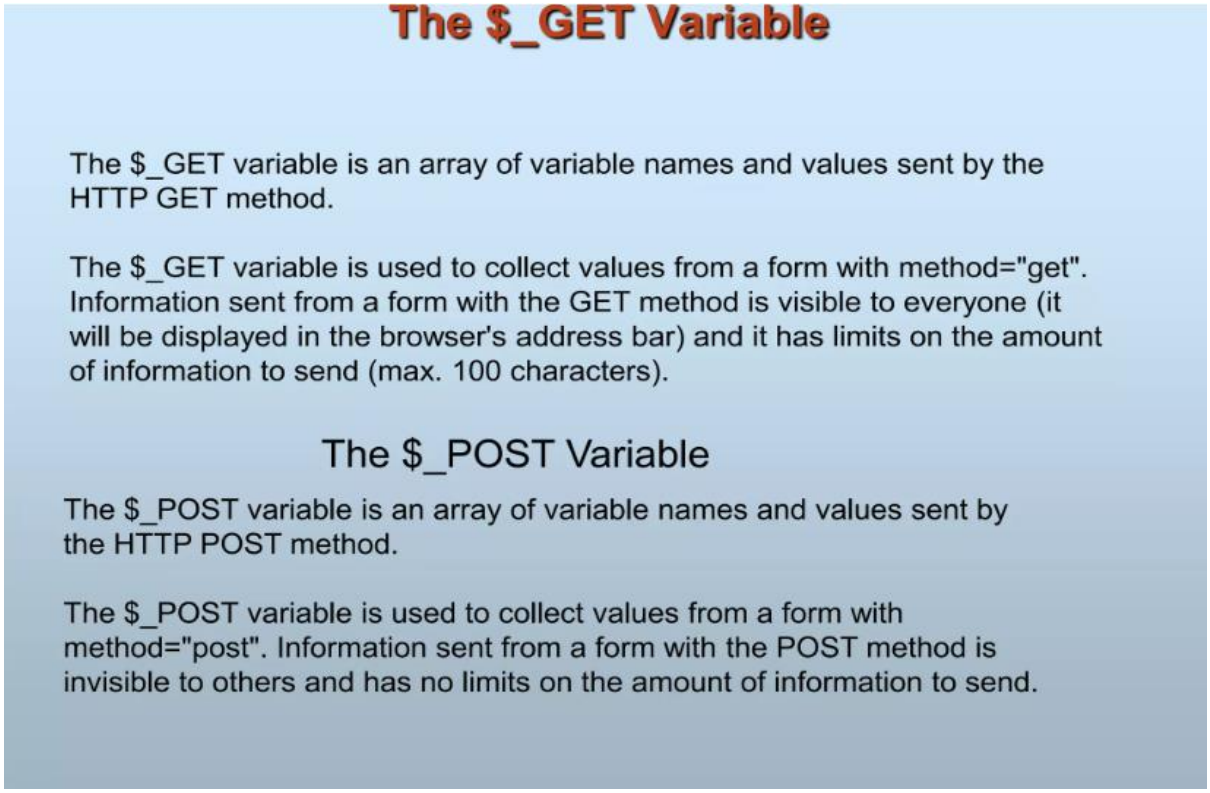
```
<?php
 $_POST['variable_name'];
?>
```

HERE,

- "$_POST[…]" is the PHP array

"'variable_name'" is the URL variable name

PHP provides two superglobals $_GET and $_POST for collecting form-data for processing.



By Using POST Method:

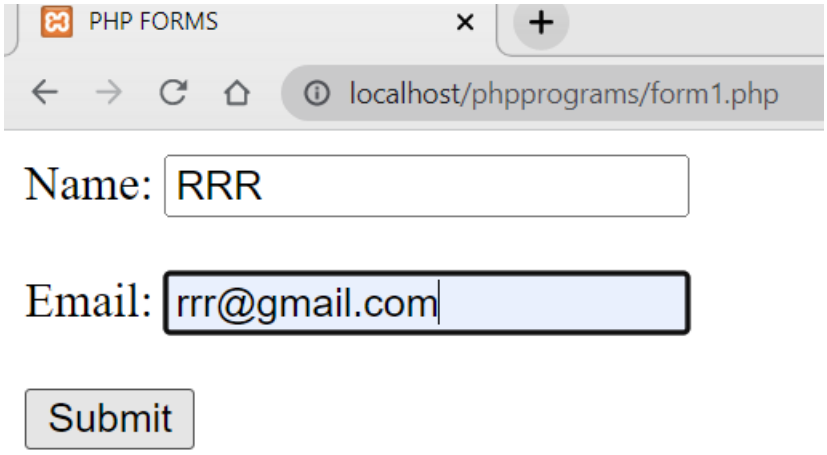
form1.php

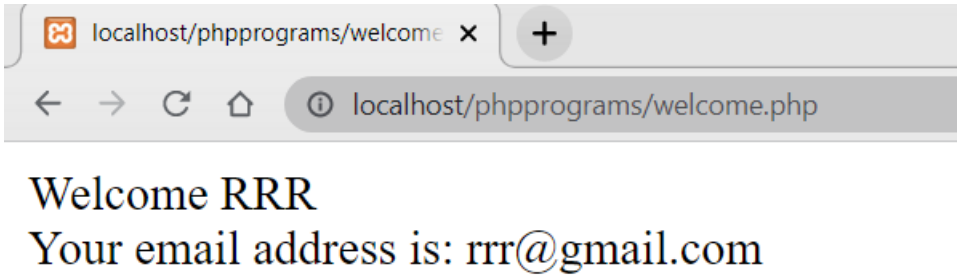
```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta http-equiv="X-UA-Compatible" content="IE=edge">
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <title>PHP FORMS</title>
8   </head>
9   <body>
10      <form action="welcome.php" method="post">
11          Name: <input type="text" name="username"/><br> <br>
12          Email: <input type="text" name="email" /><br> <br>
13          <input type="submit" value="Submit"/>
14          </form>
15  </body>
16  </html>
```

welcome.php

```php
<html>
<body>

Welcome <?php echo $_POST["username"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

OUTPUT:





2. PHP GET method

This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.

The array variable can be accessed from any script in the program; it has a global scope.
This method displays the form values in the URL.
It's ideal for search engine forms as it allows the users to book mark the results.
It has the following syntax.

```
<?php
$_GET['variable_name'];
?>
```

HERE,

- "$_GET[…]" is the PHP array

"'variable_name'" is the URL variable name.

Example:

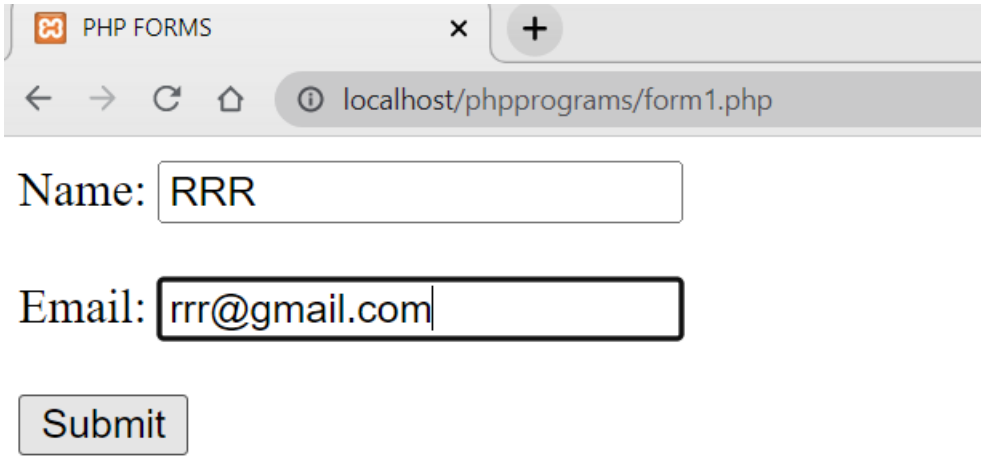
form1.php

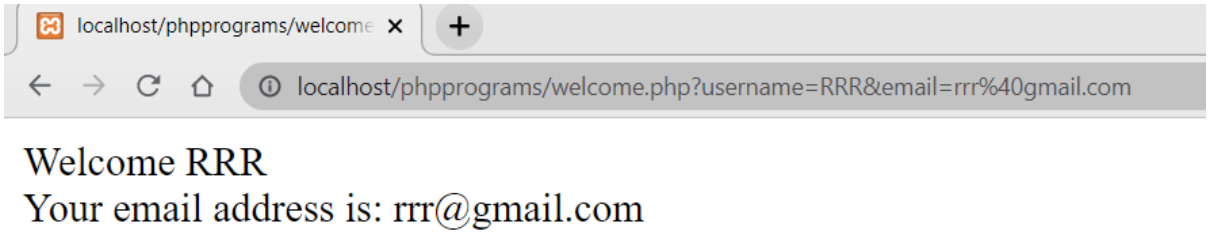
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PHP FORMS</title>
</head>
<body>
    <form action="welcome.php" method="get">
        Name: <input type="text" name="username"/><br> <br>
        Email: <input type="text" name="email" /><br> <br>
        <input type="submit" value="Submit"/>
        </form>
</body>
</html>
```

welcome.php

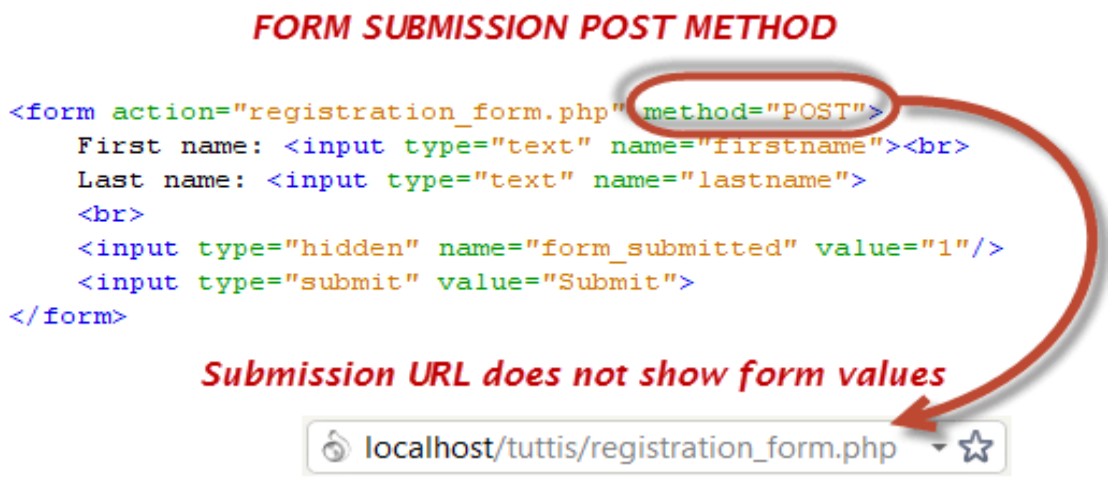
```php
<html>
<body>

Welcome <?php echo $_GET["username"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

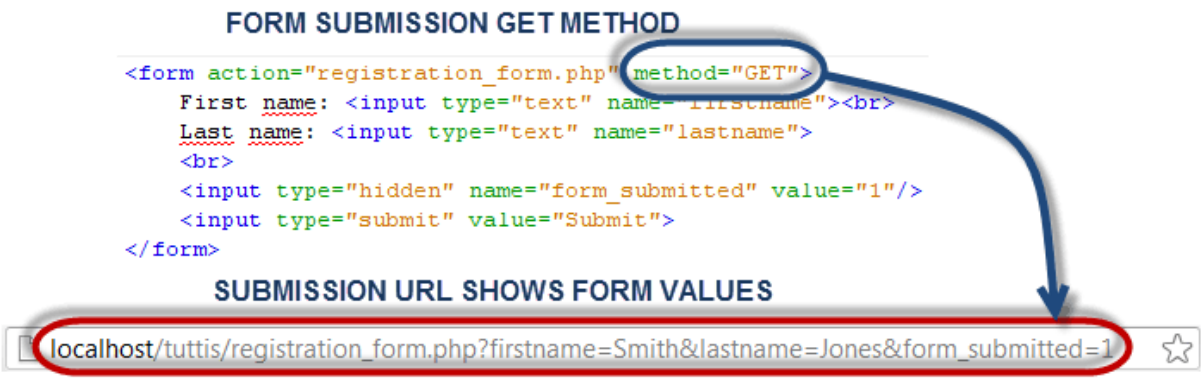
OUTPUT:





POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL





State Management in PHP

HTTP is a stateless protocol which means every user request is processed independently and it has nothing to do with the requests processed before it. Hence there is no way to store or send any user specific details using HTTP protocol.

But in modern applications, user accounts are created and user specific information is shown to different users, for which we need to have knowledge about who the user(or what he/she wants to see etc) is on every webpage.

PHP provides for two different techniques for state management of your web application, they are:

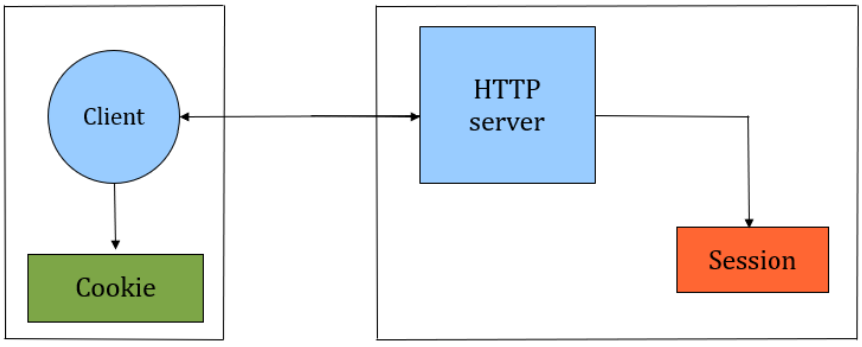
1. Server Side State Management(Session)
2. Client Side State Management(Cookies)

Persistence and HTTP

Recall http is a stateless protocol. It remembers nothing about previous transfers

Two ways to achieve persistence:

- PHP cookies
- PHP sessions



PHP Cookie

In internet programming, a cookie is a packet of information sent from the server to client, and then sent back to the server each time it is accessed by the client.

Introduces state into HTTP (remember: **HTTP is stateless**)

Cookies are **transferred** between **server** and **client** according to **http**.

PHP supports http cookies

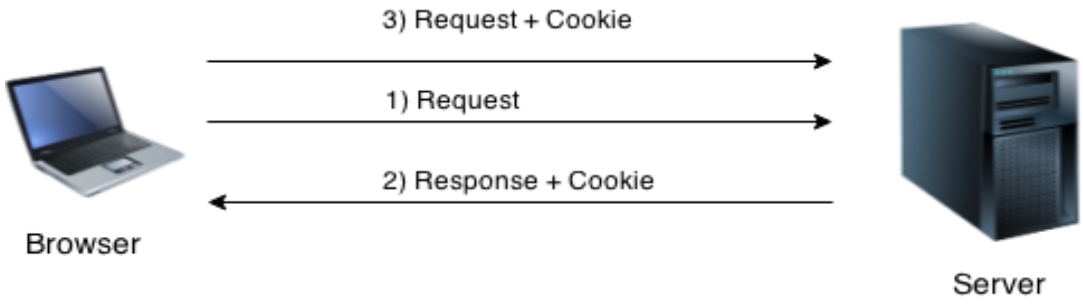
Cookies can also be thought of as tickets used to identify clients and their orders

How Cookies are implemented

- Cookies are sent from the server to the client via “Set-Cookie” headers

Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure

- The **NAME** value is a URL-encoded name that identifies the cookie.
- The **PATH** and **DOMAIN** specify where the cookie applies



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

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

PHP setcookie() function:

Syntax

```
setcookie(name, value, expire, path, domain, secure)
```

setcookie(name,value,expire,path,domain,secure)

Parameter	Description
name	(Required). Specifies the name of the cookie
value	(Required). Specifies the value of the cookie
expire	(Optional). Specifies when the cookie expires. e.g. **time()+3600*24*30** will set the cookie to expire in **30 days**. If this parameter is not set, the cookie will expire at the end of the session (when the browser closes).
path	(Optional). Specifies the server path of the cookie. If set to "**/**", the cookie will be available within the entire domain. If set to "**/phptest/**", the cookie will only be available within the test directory and all sub-directories of **phptest**. The default value is the current directory that the cookie is being set in.
domain	(Optional). Specifies the domain name of the cookie. To make the cookie available on all subdomains of example.com then you'd set it to ".example.com". Setting it to www.example.com will make the cookie only available in the www subdomain
secure	(Optional). Specifies whether or not the cookie should only be transmitted over a secure **HTTPS** connection. **TRUE** indicates that the cookie will only be set if a secure connection exists. Default is **FALSE**.

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

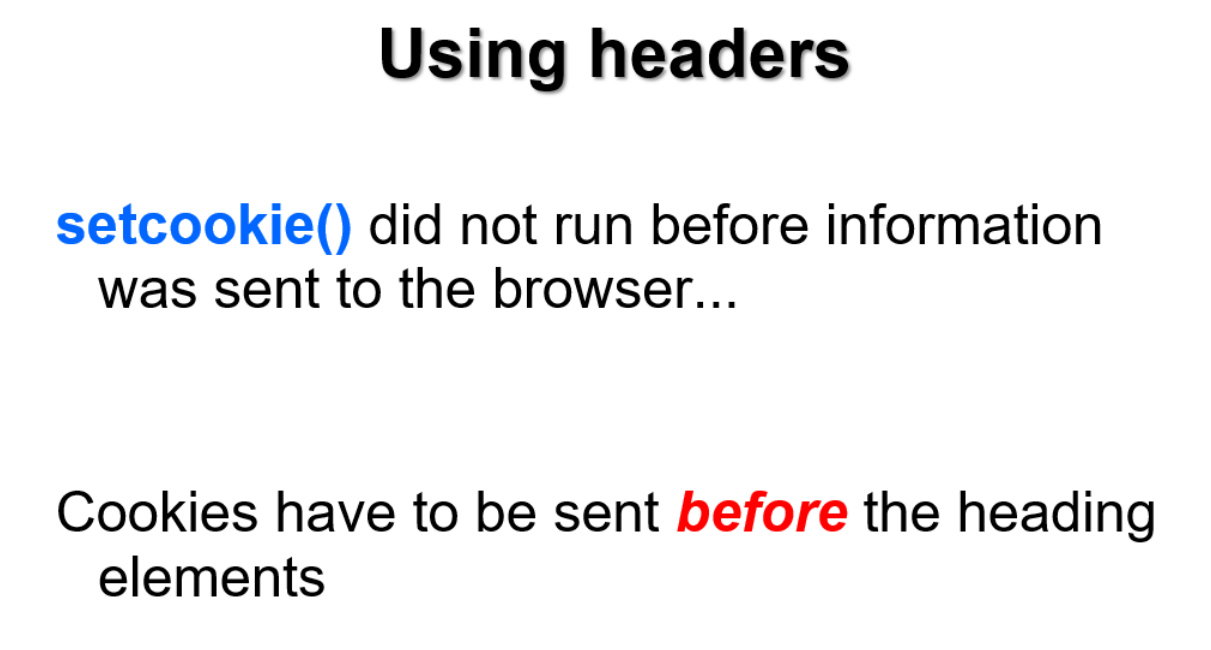
PHP $_COOKIE

PHP $_COOKIE superglobal variable is used to get cookie.

Example

1. $value=$_COOKIE["CookieName"];//returns cookie value

PHP Cookie Example:



cookie.php

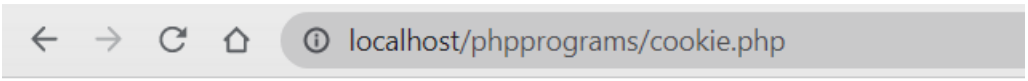
```php
<?php
setcookie("user", "grdevi");
?>
<html>
<body>
<?php
if(!isset($_COOKIE["user"])) {
    echo "Sorry, cookie is not found!";
} else {
    echo "<br/>Cookie Value: " . $_COOKIE["user"];
}
?>
</body>
</html>
```

Output:

```
Sorry, cookie is not found!
```

Firstly cookie is not set. But, if you *refresh* the page, you will see cookie is set now.

Output:



Cookie Value: CENTURION

PHP Delete Cookie

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

```
<?php
setcookie ("CookieName", "", time() - 3600);// set the expiration date to one hour ago
?>
```

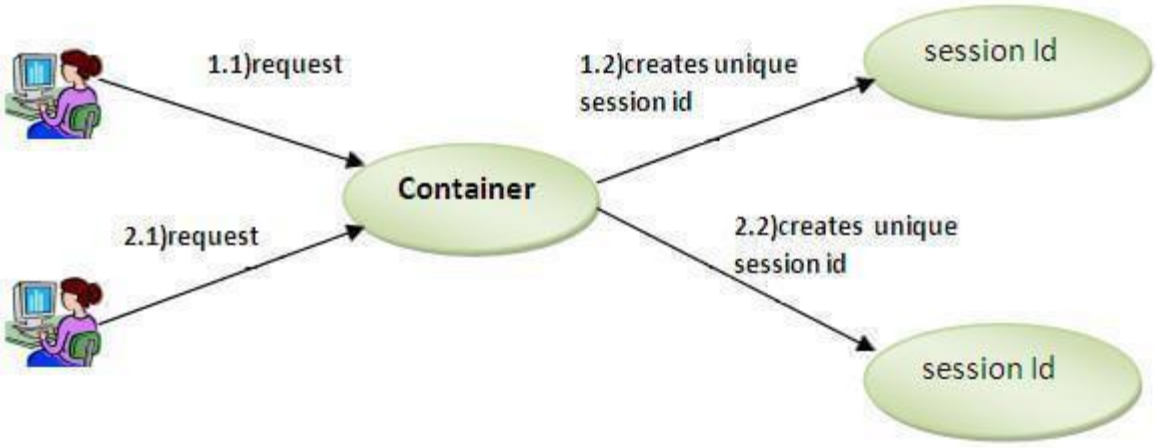
PHP SESSIONS

- A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and **are available to all pages in one application**.

Starting a PHP Session

- Before you can store user information in your PHP session, **you must first start up the session.**
- **Note: The session_start() function must appear BEFORE the <html> tag:**
- <?php session_start(); ?>
  <html>
  <body>
  </body>
  </html>
- The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

PHP session creates unique user id for each browser to recognize the user and avoid conflict between multiple browsers.



PHP session_start() function

PHP session_start() function is used to start the session. It starts a new or resumes existing session. It returns existing session if session is created already. If session is not available, it creates and returns new session.

Syntax

```
1. bool session_start ( void )
```

Example

```
1. session_start();
```

PHP $_SESSION

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

Example: Store information

```
1. $_SESSION["user"] = "Sachin";
```

Example: Get information

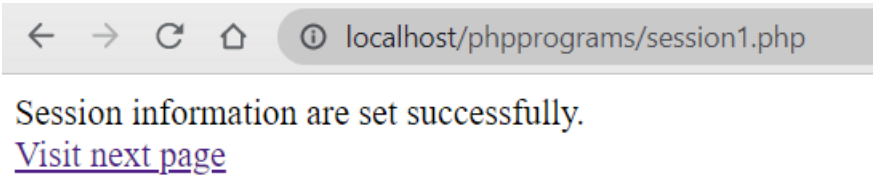
```
1. echo $_SESSION["user"];
```

PHP Session Example

session1.php

```php
<?php
session_start();
?>
<html>
<body>
<?php
$_SESSION["user"] = "Sachin";
echo "Session information are set successfully.<br/>";
?>
<a href="session2.php">Visit next page</a>
</body>
</html>
```

OUTPUT:

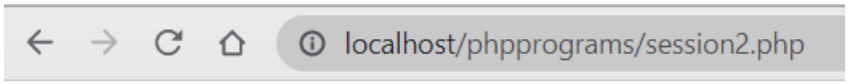


Create session2.php for next page:

session2.php

```php
<?php
session_start();
?>
<html>
<body>
<?php
echo "User is: ".$_SESSION["user"];
?>
</body>
</html>
```

When click Visit next page



User is: Sachin

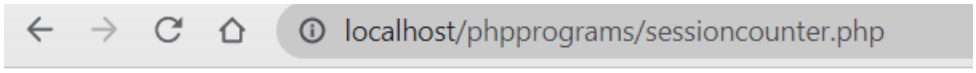
PHP Session Counter Example

sessioncounter.php

```php
<?php
    session_start();

    if (!isset($_SESSION['counter'])) {
        $_SESSION['counter'] = 1;
    } else {
        $_SESSION['counter']++;
    }
    echo ("Page Views: ".$_SESSION['counter']);
?>
```

NOTE:I Clicked refresh button 9 times so total count it shown.

OUTPUT:



Page Views: 9

DESTROYING A SESSION

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

- The unset() function is used to free the specified session variable:

```
<?php
session_start();
if(isset($_SESSION['views']))
  unset($_SESSION['views']);
?>
```

- You can also completely destroy the session by calling the session_destroy() function:

```
<?php
session_destroy();
?>
```

- **Note:** session_destroy() will reset your session and you will lose all your stored session data.

PHP File Handling

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

File Handling Operations

File handling starts with creating a file, reading its content, writing into a file to appending data into an existing file and finally closing the file. Php provides pre-defined functions for all these operations, so let's start by knowing these functions.

1. **Create a File**: fopen()
2. **Open a File**: fopen()
3. **Read a File**: fread()
4. **Write to a File**: fwrite()
5. **Append to a File**: fwrite()
6. **Close a File**: fclose()
7. **Delete a File**: unlink()

   Following are the different modes along with the literal which should be passed as argument in the fopen() function.

PHP File Create/Write

PHP Create File - fopen()

The `fopen()` function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use `fopen()` on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "data.txt". The file will be created in the same directory where the PHP code resides:

Example:

```
$fp= fopen("data.txt", "w")
```

Modes	Description
r	Read only. Starts at the beginning of the file
r+	Read/Write. Starts at the beginning of the file
w	Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist
w+	Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist
a	Append. Opens and writes to the end of the file or creates a new file if it doesn't exist
a+	Read/Append. Preserves file content by writing to the end of the file
x	Write only. Creates a new file. Returns FALSE and an error if file already exists
x+	Read/Write. Creates a new file. Returns FALSE and an error if file already exists

Write or Append data to File in PHP

To write content to a file we can use fwrite() function in PHP. To use fwrite() function to write content to a file, we first need to open the file resource in **write** or **append** mode.

PHP Write to File - fwrite()

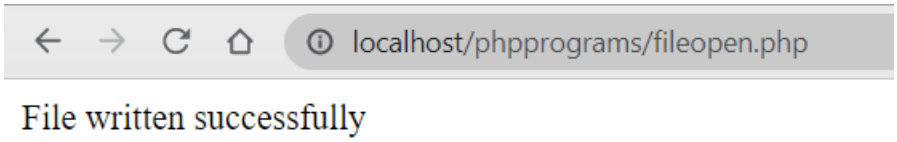
The `fwrite()` function is used to write to a file.

The first parameter of `fwrite()` contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "data.txt":

fileopen.php

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

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



PHP File Open/Read/Close

PHP Read File - fread()

The `fread()` function reads from an open file.

The first parameter of `fread()` contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "study.txt" file to the end:

```
fread($myfile,filesize("study.txt"));
```

PHP Close File - fclose()

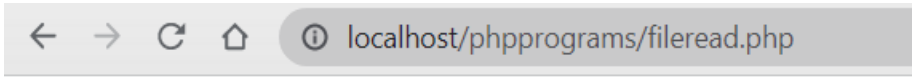
The `fclose()` function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The `fclose()` requires the name of the file (or a variable that holds the filename) we want to close:

fileread.php

```php
<?php
// Opening a file
$myfile = fopen("study.txt", "r");

// reading the entire file using
// fread() function
echo fread($myfile, filesize("study.txt"));

// closing the file
fclose($myfile);
?>
```



THIS IS FILE READ OPERATION

MYSQL

- MySQL is one of the most popular relational database system being used on the Web today. It is freely available and easy to install, however if you have installed Wampserver it already there on your machine. MySQL database server offers several advantages:
- MySQL is easy to use, yet extremely powerful, fast, secure, and scalable.
- MySQL runs on a wide range of operating systems, including UNIX or Linux, Microsoft Windows, Apple Mac OS X, and others.
- MySQL supports standard SQL (Structured Query Language).
- MySQL is ideal database solution for both small and large applications.
- MySQL is developed, and distributed by Oracle Corporation.
- MySQL includes data security layers that protect sensitive data from intruders.
- MySQL database stores data into tables like other relational database. A table is a collection of related data, and it is divided into rows and columns.

Example:

Each row in a table represents a data record that are inherently connected to each other such as information related to a particular person, whereas each column represents a specific field such as id, first_name, last_name, email, etc. The structure of a simple MySQL table that contains person's general information may look something like this:

```
+----+------------+-----------+----------------------+
| id | first_name | last_name | email                |
+----+------------+-----------+----------------------+
|  1 | Peter      | Parker    | peterparker@mail.com |
|  2 | John       | Rambo     | johnrambo@mail.com   |
|  3 | Clark      | Kent      | clarkkent@mail.com   |
|  4 | John       | Carter    | johncarter@mail.com  |
|  5 | Harry      | Potter    | harrypotter@mail.com |
+----+------------+-----------+----------------------+
```

Tip: Websites like Facebook, Twitter, Wikipedia uses MySQL for their storage need. So you can easily understand what MySQL is capable of.

What is the major difference between MySQL and SQL?

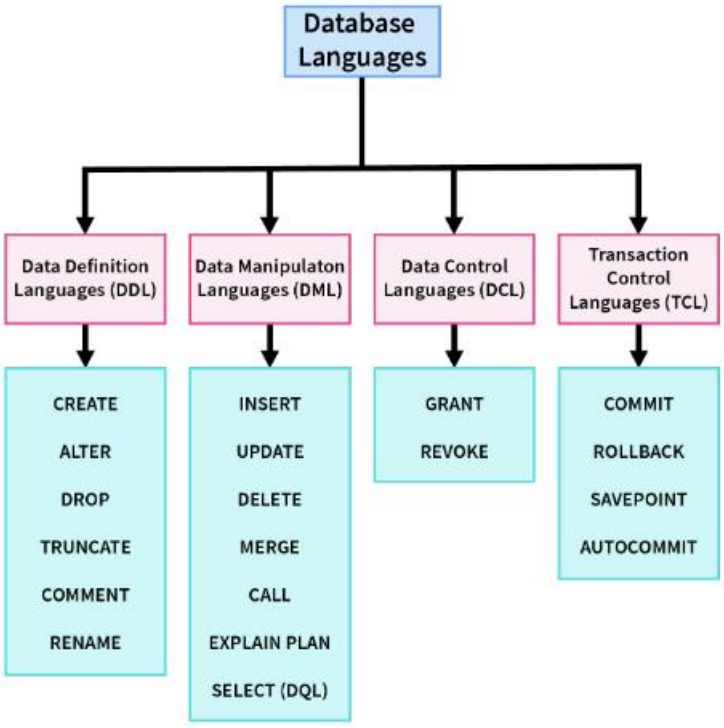
SQL is a query programming language for managing RDBMS. In contrast, MySQL is an RDBMS (Relational Database Management System) that employs SQL. So, the major difference between the two is that MySQL is software, but SQL is a database language.

Why is MySQL better than other databases?

MySQL is generally faster and more efficient than other relational database management systems (RDBMS), so it is often the preferred choice for applications that require high performance.

Languages in MYSQL:

These 4-languages implemented in MYSQL Software.



What are the features of MySQL in PHP?

MySQL Features

- Relational Database Management System (RDBMS) MySQL is a relational database management system. ...
- Easy to use. MySQL is easy to use. ...

- It is secure. ...
- Client/ Server Architecture. ...
- Free to download. ...
- It is scalable. ...
- Speed. ...
- High Flexibility.

PHP Connect to MySQL Server

→Ways of Connecting to MySQL through PHP

- In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server. PHP offers two different ways to connect to MySQL server:

1.)**MySQLi** (Improved MySQL) and

2.**PDO** (PHP Data Objects) extensions.

- While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only. MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server. Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API which is relatively easy for beginners to understand.
- **Tip:** The PHP's MySQLi extension provides both speed and feature benefits over the PDO extension, so it could be a better choice for MySQL-specific projects.

1.Connecting to MySQL Database Server

In PHP you can easily do this using the **mysqli_connect()** function. All communication between PHP and the MySQL database server takes place through this connection. Here're the basic syntaxes for connecting to MySQL using MySQLi and PDO extensions:

1. Syntax: MySQLi, Procedural way

```
$link = mysqli_connect("hostname", "username", "password", "database");
```

2. Syntax: MySQLi, Object Oriented way

```
$mysqli = new mysqli("hostname", "username", "password", "database");
```

3. Syntax: PHP Data Objects (PDO) way

```
$pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");
```

The hostname parameter in the above syntax specify the host name (e.g. localhost), or IP address of the MySQL server, whereas the username and password parameters specifies the credentials to access MySQL server, and the database parameter, if provided will specify the default MySQL database to be used when performing queries.

2. Closing the MySQL Database Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, if you want to close it earlier you can do this by simply calling the PHP **mysqli_close()** function.

Example: Procedural Object Oriented PDODownload

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
   die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Print host information
echo "Connect Successfully. Host info: " . mysqli_get_host_info($link);

// Close connection
mysqli_close($link);
?>
```

3.PHP MySQL Create Database

→Creating MySQL Database Using PHP

- Before saving or accessing the data, we need to create a database first. The CREATE DATABASE statement is used to create a new database in MySQL.

- Let's make a SQL query using the CREATE DATABASE statement, after that we will execute this SQL query through passing it to the PHP **mysqli_query()** function to finally create our database. The following example creates a database named demo.

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
   die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt create database query execution
$sql = "CREATE DATABASE demo1";
if(mysqli_query($link, $sql)){
   echo "Database created successfully";
} else{
   echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

4. PHP MySQL Create Tables

→Creating Tables inside MySQL Database Using PHP

- create some tables inside the database that will actually hold the data. A table organizes the information into rows and columns.

❑The SQL CREATE TABLE statement is used to create a table in database.

Let's make a SQL query using the CREATE TABLE statement, after that we will execute this SQL query through passing it to the PHP **mysqli_query()** function to finally create our table.

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt create table query execution
$sql = "CREATE TABLE persons(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    email VARCHAR(70) NOT NULL UNIQUE
)";
if(mysqli_query($link, $sql)){
    echo "Table created successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}
 // Close connection
mysqli_close($link);
?>
```

5. PHP MySQL INSERT Query

→Inserting Data into a MySQL Database Table

- execute SQL query to insert records into a table.

❑The **INSERT INTO** statement is used to insert new rows in a database table.

Insert into <tablename>(field1,filed2,field3..)values(value1,value2,value3…);

- Let's make a SQL query using the INSERT INTO statement with appropriate values, after that we will execute this insert query through passing it to the PHP mysqli_query() function to insert data in table.
- Here's an example, which insert a new row to the persons table by specifying values for the first_name, last_name and email fields.

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
   die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('Peter', 'Parker',
'peterparker@mail.com')";
if(mysqli_query($link, $sql)){
   echo "Records inserted successfully.";
} else{
   echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

If you remember from the preceding chapter, the id field was marked with the AUTO_INCREMENT flag. This modifier tells the MySQL to automatically assign a value to this field if it is left unspecified, by incrementing the previous value by 1.

→Inserting Multiple Rows into a Table

You can also insert multiple rows into a table with a single insert query at once. To do this, include multiple lists of column values within the INSERT INTO statement, where column values for each row must be enclosed within parentheses and separated by a comma.

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
   die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES
        ('John', 'Rambo', 'johnrambo@mail.com'),
        ('Clark', 'Kent', 'clarkkent@mail.com'),
        ('John', 'Carter', 'johncarter@mail.com'),
        ('Harry', 'Potter', 'harrypotter@mail.com')";
if(mysqli_query($link, $sql)){
   echo "Records added successfully.";
} else{
   echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

→Insert Data into a Database from an HTML Form

we can insert data into database obtained from an HTML form. Let's create an HTML form that can be used to insert new records to persons table.

Step 1: Creating the HTML Form

Here's a simple HTML form that has three text <input> fields and a submit button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Add Record Form</title>
</head>
<body>
<form action="insert.php" method="post">
   <p>
      <label for="firstName">First Name:</label>
      <input type="text" name="first_name" id="firstName">
   </p>
   <p>
      <label for="lastName">Last Name:</label>
      <input type="text" name="last_name" id="lastName">
   </p>
   <p> <label for="emailAddress">Email Address:</label>
      <input type="text" name="email" id="emailAddress">
   </p>

   <input type="submit" value="Submit">
</form>
</body>
</html>
```

Step 2: Retrieving and Inserting the Form Data

When a user clicks the submit button of the add record HTML form, in the example above, the form data is sent to 'insert.php' file. The 'insert.php' file connects to the MySQL database server, retrieves forms fields using the PHP $_REQUEST variables and finally execute the insert query to add the records. Here is the complete code of our 'insert.php' file:

Note: The mysqli_real_escape_string() function escapes special characters in a string and create a legal SQL string to provide security against SQL injection.

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
   die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Escape user inputs for security
$first_name = mysqli_real_escape_string($link, $_REQUEST['first_name']);
$last_name = mysqli_real_escape_string($link, $_REQUEST['last_name']);
$email = mysqli_real_escape_string($link, $_REQUEST['email']);

// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('$first_name',
'$last_name', '$email')";
if(mysqli_query($link, $sql)){
   echo "Records added successfully.";
} else{
   echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

OUTPUT: