Unit 2 Working with Basic Building Blocks of PHP

Introduction to PHP

- PHP is a server scripting language and a powerful tool for making dynamic and interactive Web pages.
- PHP stands for "**PHP: Hypertext Preprocessor**".
- PHP is a widely-used, open source scripting language.
- PHP scripts are executed on the server.
- PHP is free to download and use.

A Brief History of PHP

- PHP was originally created by Rasmus Lerdorf in 1994. Programmer Rasmus Lerdorf initially created a set of C scripts he called "Personal Home Page Tools" to maintain his personal homepage. The scripts performed tasks such as displaying his resume and recording his web-page traffic.
- These were released and extended to include a package called the Form Interpreter (PHP/FI). While PHP originally stood for "Personal Home Page", now it is known as "PHP: Hypertext Preprocessor".
- In 1997 Zeev Suraski and Andi Gutmans along with Rasmus rewrite PHP and released PHP version 3.0 in June 1998. After this release PHP becomes so much popular.
- The PHP version 4.0 was launched in May 2000.This version includes session handling, output buffering and support for wide variety of web server platforms.
- The PHP 5.0 version released in 2004 with object oriented programming concept.

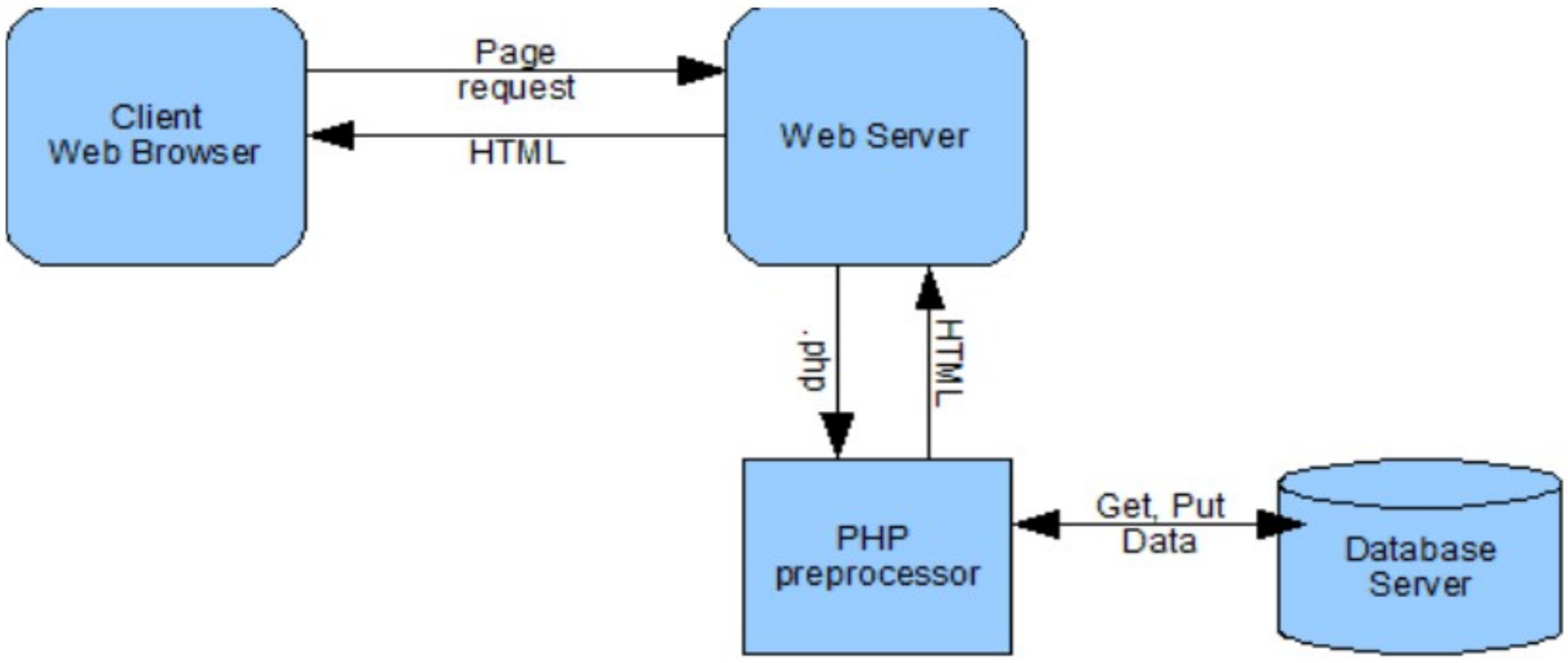
Open Source

- In general, open source refers to any program whose source code is made available for use or modification. Open source software is usually developed as a public collaboration and made freely available. It means can be used without purchasing any license.
- Open Source is a certification mark owned by the Open Source Initiative (OSI). Developers of software that is intended to be freely shared and possibly improved and redistributed by others can use the Open Source trademark if their distribution terms conform to the OSI's Open Source Definition. To summarize, the Definition model of distribution terms require that:
  - The software being distributed must be redistributed to anyone else without any restriction.
  - The source code must be made available (so that the receiving party will be able to improve or modify it).

  Example of Open Source: Linux, Apache, MySQL, PHP.

How PHP Works?

- PHP sits between your browser and the web server. When you type in the URL of a PHP website in your browser, your browser sends a request to the web server. The web server then calls the PHP script on that page. The PHP module executes the script which then sends the result in the form of HTML back to your browser which you seen on the screen.



Relationship between Apache, MySQL and PHP (AMP Module)

- AMP stands for Apache MySQL PHP

PHP

- PHP is a server side scripting that was designed for creating dynamic websites.
- PHP script runs on a web server.
- PHP is loosely type language because it converts a data type of variable automatically.
- PHP is powerful scripting language that can be run in the command line of any computer with PHP installed.
- However, PHP alone isn't enough in order to build dynamic web sites.

Apache

- To use PHP on a web site, you need a server that can process PHP scripts. Apache is a free web Server that, once installed on a computer, allows developers to test PHP scripts locally; this makes it an invaluable piece of your local development environment.
- Like all web servers, Apache accepts an HTTP request and serves an HTTP response.
- Apache is open source free software distributed by the Apache Software Foundation.

MySQL

- Additionally, dynamic websites are dependent on stored information that can be modified quickly and easily; this is the main difference between a dynamic site and a static HTML site.
- However, PHP doesn't provide a simple, efficient way to store data. This is where a relational database management system like MySQL comes into play.
- PHP provides native support for it and the database is free, open-source project.
- MySQL is a relational database management system (DBMS). Essentially, this means that MySQL allows users to store information in a table-based structure, using rows and columns to organize different pieces of data.

Creating and Saving a PHP File

- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

- A PHP script starts with **<?php** and ends with **?>**
- A PHP script can be placed anywhere in the document.
- **Syntax:**

```
<? php
      // PHP code goes here
?>
```

- The default file extension for PHP files is "**.php**".
- A PHP file normally contains HTML tags, and some PHP scripting code.
- Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page.

```
<html>
<head>
        <title>Unit-2</title>
</head>
<body>
        <?php
              echo "Hello World";
        ?>
</body>
</html>
```

Output:

Hello World

PHP Start and End Tags

- There are **four** different pairs of opening and closing tags which can be used in php.
- Two of those <? php ?> and <script language= "php"> </script> are always available.
- The other two are short tags and ASP style tags and can be turned on and off the php.ini configuration file.
- Below, we have an example of a simple PHP script which sends the text “Hello World” to the browser:

1. **Standard PHP tags**

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

2. **Script tags**

```
<script language="php">
     echo “Hello World”;
</script>
```

3. **Short tags**

```
<?
     echo "Hello World";
```

```
?>
```

- Short tags are only available when they are enabled via the short_open_tag php.ini configuration file directive, or if php was configured with the enable-short-tags option.

4. **ASP style tags**

```
<%
    echo "Hello World";
%>
```

- ASP style tags are only available when they are enabled via the asp_tags php.ini configuration file directive.

Comments in PHP

- PHP also supports comment like a C, C++, Java languages.
- PHP supports two types of comments:

Single line comment

- Single line comment started with symbol //.

```
<?php // this is the single line comment ?>
```

- If you have a single line comment, another option is to use a # sign. Here is an example of this:

```
<?php # this is a single line comment ?>
```

Multiline comment

- Multiline comment started with symbol /* and end with symbol */.
- So, the multiline comment starts with one line and can be expand to more than one line.

```
<?php /* this is the Multiline Comment */ ?>
```

PHP Echo and Print Statements

- They are both used to output data to the screen.

Echo

- The simplest use of echo is to print a string as argument, for example:

```
echo "Hello World";
```

Or equivalently:

```
echo ("Hello World");
```

- You can also give multiple arguments to the unparenthesized version of echo , separated by commas:

```
echo "Hello", "World";
```

- The parenthesized version, however, will not accept multiple arguments: echo("Hello", "World");

Input	Output
echo "Hello World";	Hello World
echo("Hello World");	Hello World
echo "Hello", "World";	Hello World

echo("Hello", "World");	Invalid Argument

Print

- The command print is very similar to echo, with two important differences:
  - print can accept only one argument.
  - print returns a value, which represents whether the print statement succeeded.
- The value returned by print will be 1 if the printing was successful and 0 if unsuccessful.
- For example, the following two lines will print exactly the same thing:

```
print("3.14159 ");  //print a string
print(3.14159);      //print a number
```

Difference between Echo and Print

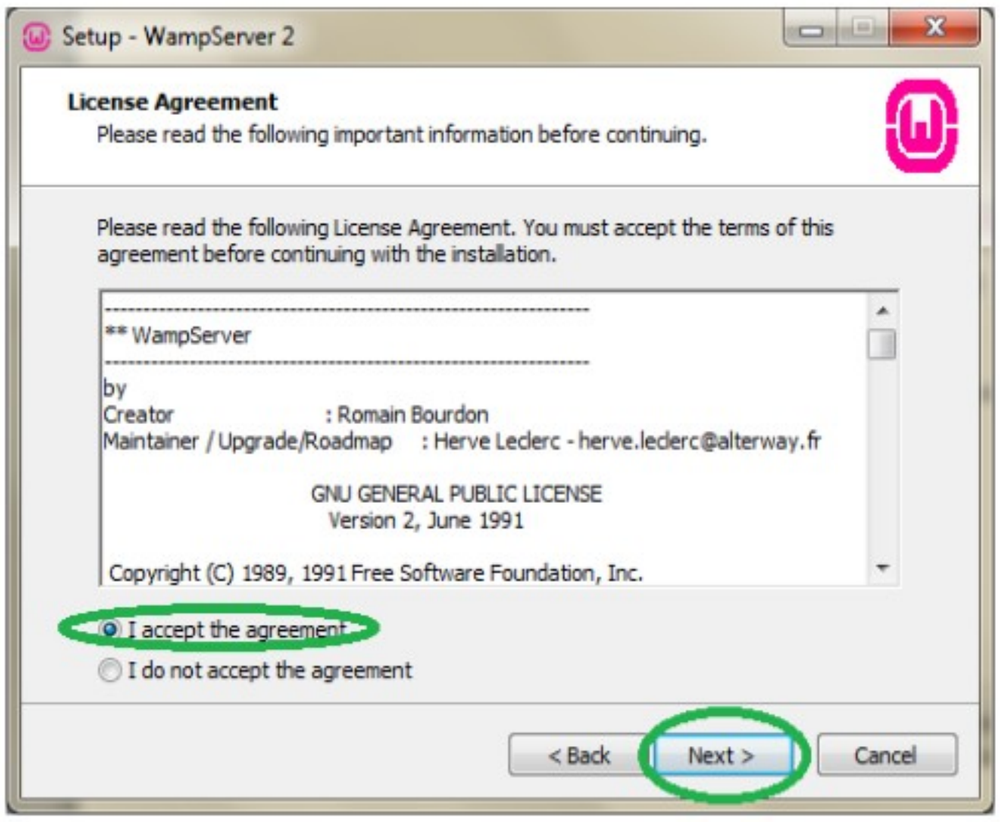
ECHO	PRINT
It can accept multiple expressions.	It cannot accept multiple expressions.
It is faster than print.	It is slower than echo.
It can pass multiple string separated as (,).	It cannot pass multiple arguments.
It is a statement used to display the output and can be used with the parentheses echo or without the parentheses echo.	It is also a statement which is used to display the output and used with the parentheses print() or without the parentheses print.
It doesn't return any value.	It always returns the value 1.

Installation of WAMP Server

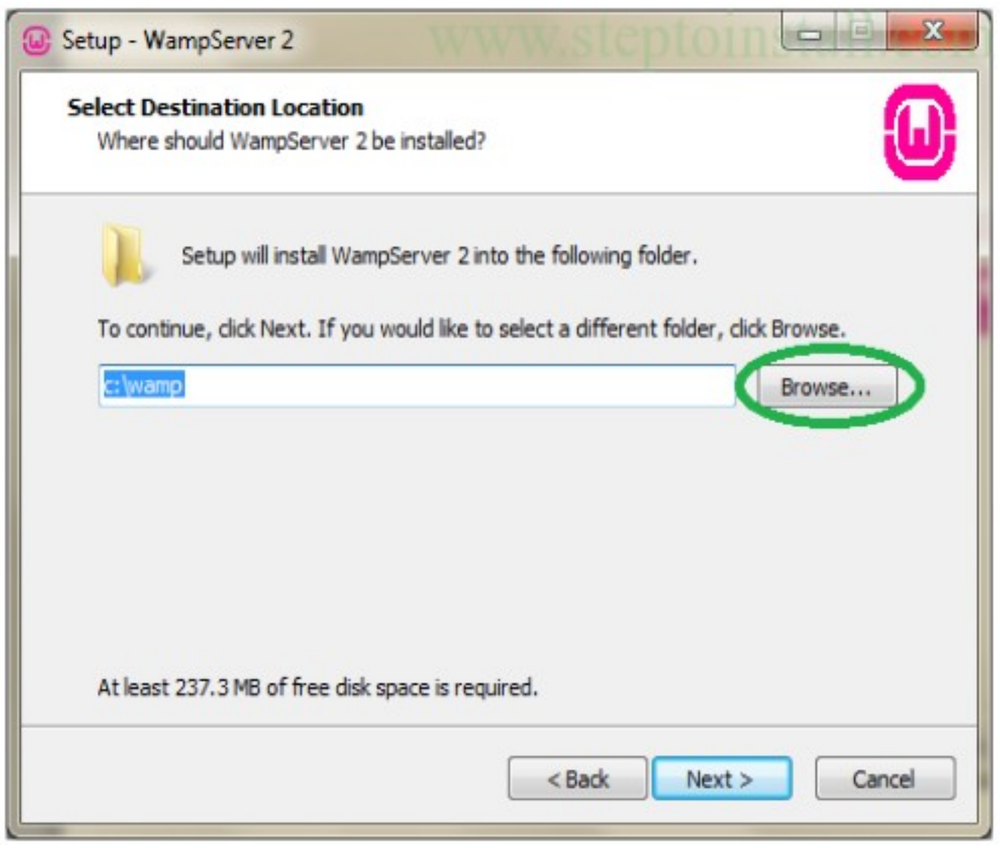
- **Step-1 : Download the WAMP Server**
- **Step-2 : Start Setup Wizard**

  You'll get below Setup Wizard window, When you Run WAMP server .exe file. It shows config of WAMP. Click 'Next' button to continue.

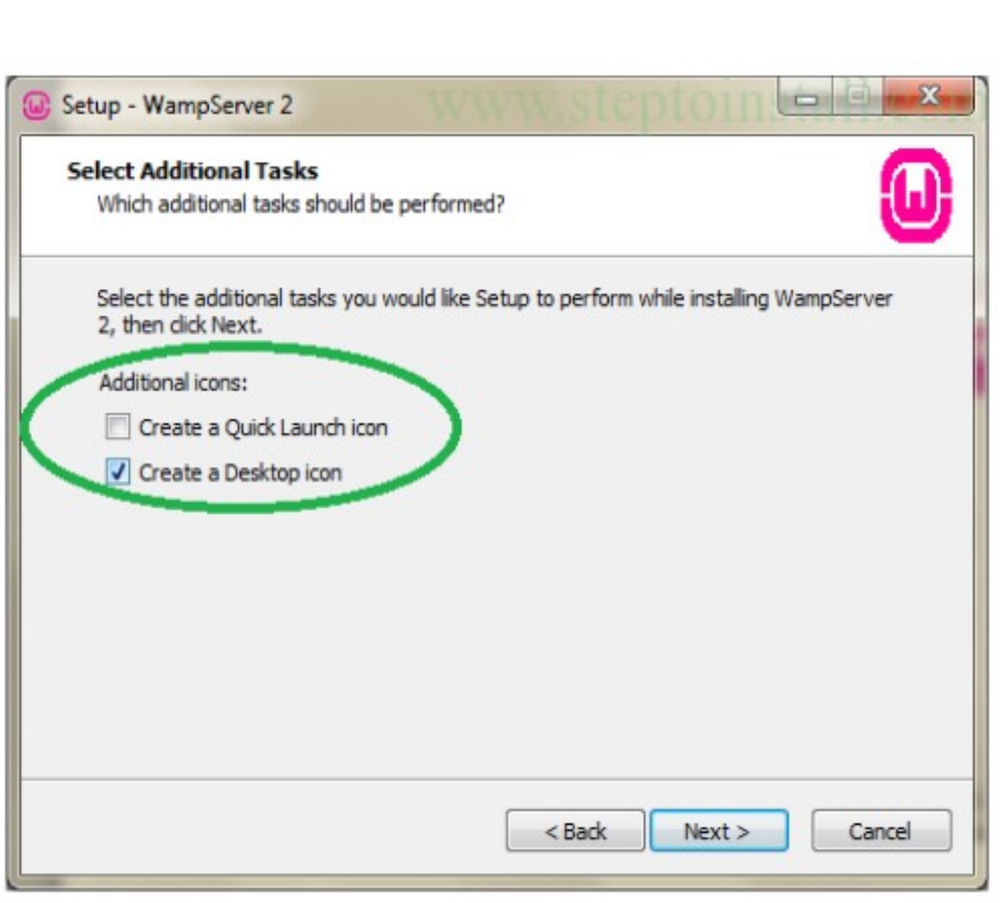
- **Step-3:** You should agree the license of WAMP Server before Installation of WAMP Server.



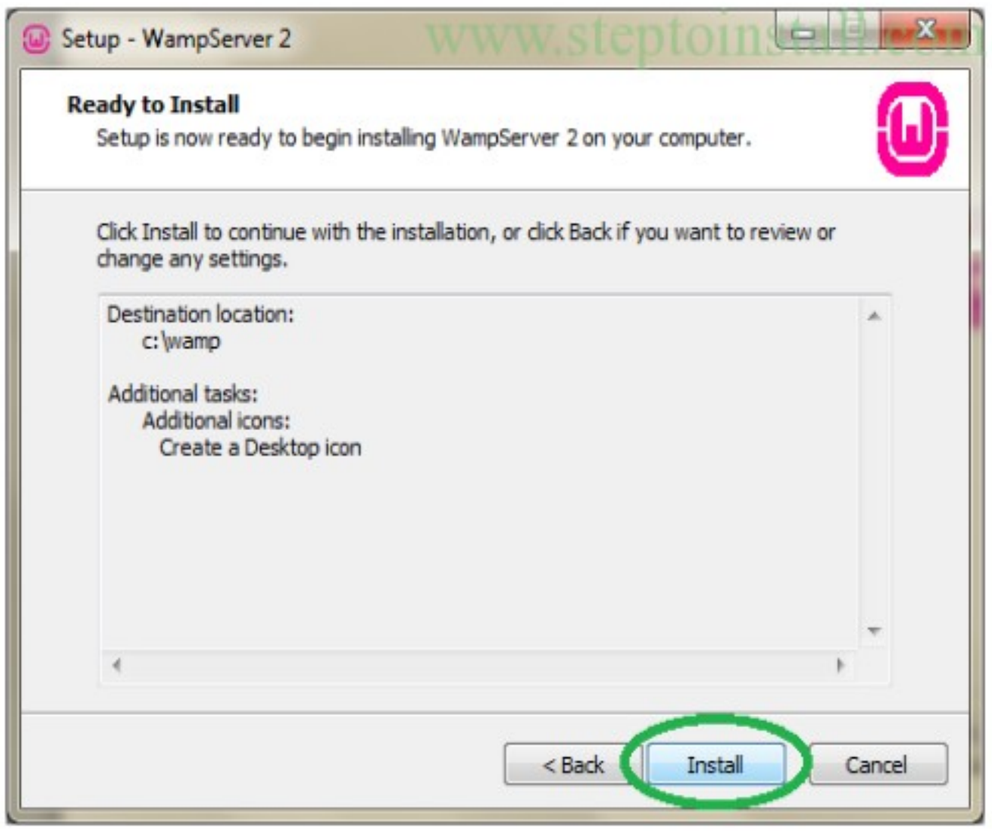
- **Step-4: Select Destination and Icon**
  Here, You should select the Server Location. Most of the server installing into C drive only. Me too installing into C drive. You can browse any other location and click 'Next'.



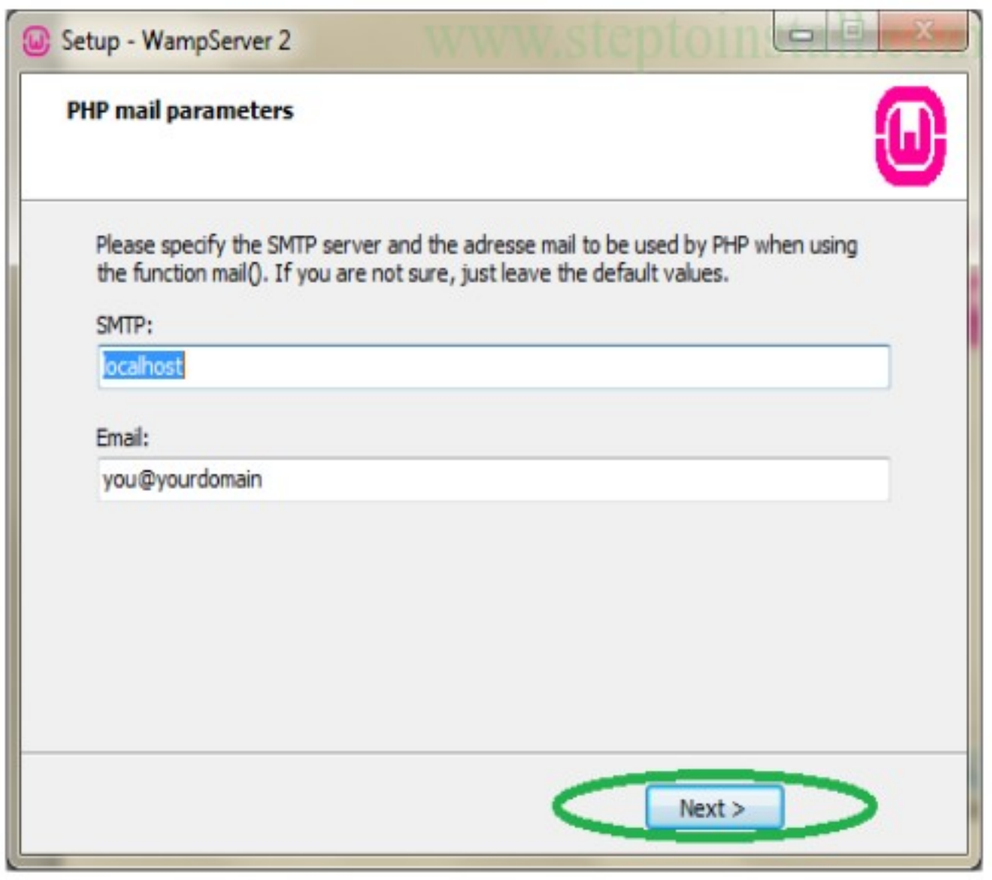
- **Step-5:** choose which icons you want.



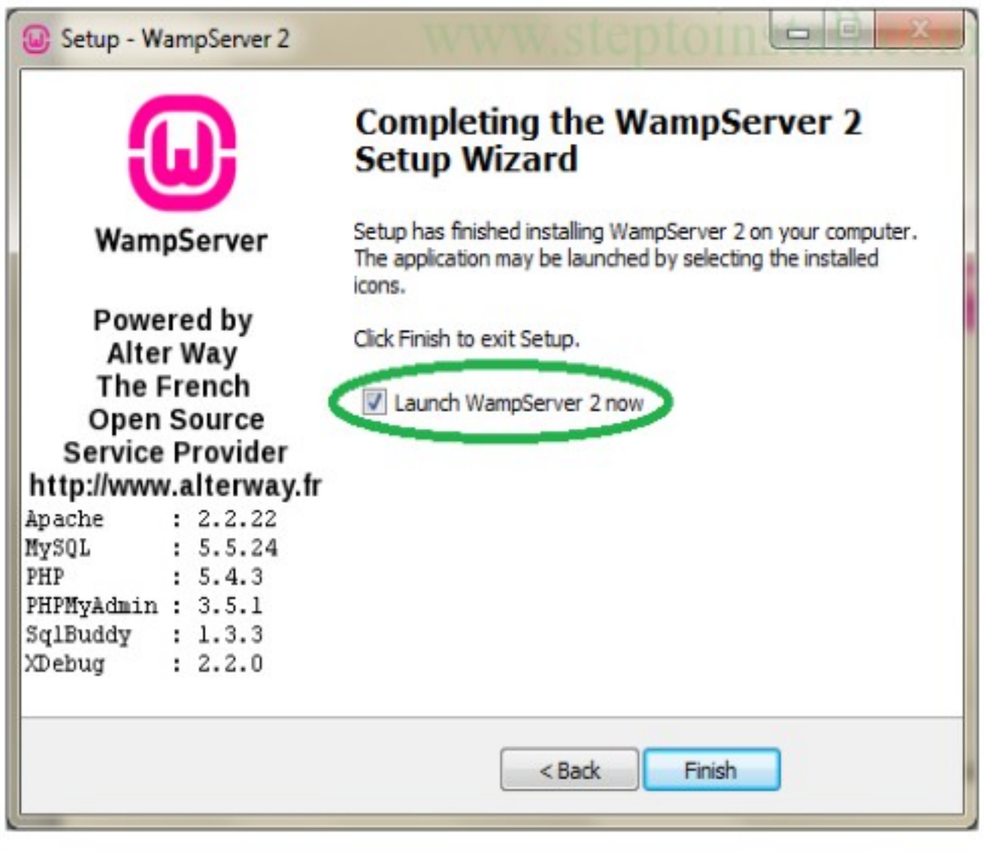
- **Step-6: Installing and Complete**
  In 'Ready to Install' window, just click 'Install' button.



- **Step-7: Specify SMTP Server and Mail**
  Leave Default value and Click 'Next'.

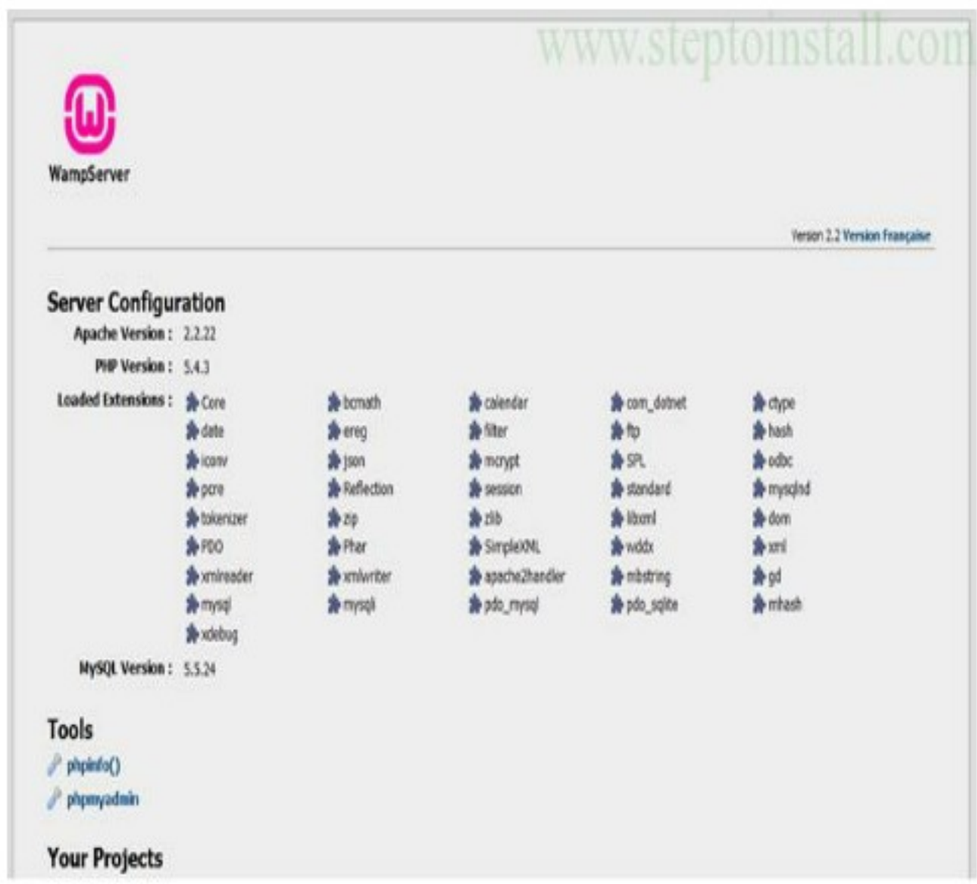


- **Step-8:** Now Installation Process is almost done. Tic the Launch and Finish it.



- **Step-9: Run WAMP Server**
  Double click the WAMP Server icon. You'll get small icon in *TaskBar*. First its color is Red and change to Orange. At-last it goes to Green color. If green color comes, Your WAMP server running successful.

- **Step-10:** Now see your WAMP server Home Page with following URL *'http://localhost/'*.



PHP Variable and Variable Rules

- Variables are "containers" for storing information.
- **Declaring Variables:** $var_name=value;
- **Rules for PHP variables:**
  - A variable starts with the $ sign, followed by the name of the variable.
  - A variable name must start with a letter or the underscore character.
  - A variable name cannot start with a number.
  - A variable name can only contain alpha-numeric character and underscore (A-z, 0-9, and _).
  - Variable names are case-sensitive ($age and $AGE are two different variables).

Example:

```
$txt = "Hello world!";      // variable $txt will hold the value Hello world!
$x = 5;                     // variable $x will hold the value 5
$y = 10.5;                  // variable $y will hold the value 10.5
```

Data Types

- PHP supports following data types:

Scalar data

- Scalar data means data that contains only a single value.

Scalar Data Type	Description	Example
Integer	A whole number	15
Float	A floating - point number	8.23
String	A series of characters	“Hello, world!”
Boolean	Represents either true or false	TRUE

Compound data

- Compound data is data that can contain more than one value.

Compound Data Type	Description
Array	An ordered map (contains names or numbers mapped to values)
Object	A type that may contain properties and methods

Special data

- Finally, PHP supports two special data types, so called because they don’t contain scalar or compound data as such, but have a specific meaning:

Special Data Type	Description
Resource or database	Contains a reference to an external resource, such as a file
Null	May only contain null as a value, meaning the variable explicitly does not contain any value

Why PHP is known as loosely typed language?

- PHP automatically converts the variable to the correct data type, depending on its value.
- In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

Changing Types (casting)

Gettype

- This function is use to get the type of the variable that means this function will return the type of the PHP variable.
- **Syntax:** gettype(arg)
- The above function will return a string representing the type of arg: integer, float, string, array, and object, NULL or unknown type.

Example:

```
<?php
    $a=10;
```

```
    echo gettype($a);
    $b="rahul";
    echo gettype($b);
?>
```

Output:-

integer string

settype

- The settype() function is used to set the type of a variable.
- **Syntax:** settype(var_name, var_type)
- **var_name:** The variable being converted.
- **var_type:** Type of the variable. Possible values: boolean, integer, float, string, array, object, null.

Example:

```
$a=65;
settype($a, "string");
```

casting

- If you want to change type temporary, i.e., want to use inside an expression, you can use type casting.
- **Syntax:** (type)$variable
- Where **type** is a type of variable you wish to cast to.

Example:

```
$var1= 10; //integer
$var2= (string) $var1; // $var1 is now set to 10 (string)
```

PHP Operators

- Operators are used to perform operations on variables and values.

Arithmetic Operators

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

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (PHP 5.6)

Logical Operators

- The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and, &&	And	$x and $y	True if both $x and $y are true
or, ||	Or	$x or $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

Assignment Operators

- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=". It means that the left operand set to the value of the assignment expression on the right.

Assignment	Same as	Description
x = y	x = y	The left operand set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1.= $txt2	Appends $txt2 to $txt1

Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

Conditional Operator

- There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true? Then value X : Otherwise value Y

Example:

```
<?php
        $number = 21;
        $result= ($number % 2 == 0) ? "Number is Even" : "Number is Odd";
        echo $result;
?>
```

Output:

Number is Odd

Comparison Operators

- The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

Bitwise Operators

- Bitwise operators allow evaluation and manipulation of specific bits within an integer.

Operator	Name	Example	Result
&	And	$x & $y	Bits that are set in both $x and $y are set.
|	Or	$x | $y	Bits that are set in either $x or $y are set.
^	Xor	$x ^ $y	Bits that are set in $x or $y but not both are set.
~	Not	~$x	Bits that are set in $x are not set, and vice versa.
<<	Shift left	$x << $y	Shift the bits of $x $y steps to the left.
>>	Shift right	$x >> $y	Shift the bits of $x $y steps to the right.

Precedence of PHP Operators

- Operator precedence determines the grouping of terms in an expression.
- This affects how an expression is evaluated.

- For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.
- Higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

PHP Constants

- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).
- To create a constant, use the define() function.
- **Syntax:** define(name, value, case-insensitive)
  - **name:** Specifies the name of the constant
  - **value:** Specifies the value of the constant
  - **case-insensitive:** Specifies whether the constant name should be case-insensitive. Default is false.

Example 1:

- The example below creates a constant with a **case-sensitive** name.

```
<?php
    define("GREETING", "Welcome to W3Schools.com!");
    echo GREETING;
?>
```

Output:

Welcome to W3Schools.com!

Example 2 :

- The example below creates a constant with a **case-insensitive** name.

```
<?php
    define("GREETING", "Welcome to W3Schools.com!", true);
    echo greeting;
?>
```

Output:

Welcome to W3Schools.com!

constant() function

- The constant() function returns the value of a constant.
- **Syntax:** constant(constant)
  - Constant: Required. Specifies the name of the constant.

Example:

```
<?php
    define("GREETING","Hello you! How are you today?");
    echo constant("GREETING");
?>
```

Output:

Hello you! How are you today?

PHP Magic constants (predefined constants)

- PHP provides a large number of predefined constants to any script which it runs.
- There are five magical constants that change depending on where they are used. For example, the value of __LINE__ depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:
- A few "magical" PHP constants are given below:

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file.
__FUNCTION__	The function name. This constant returns the function name as it was declared.
__CLASS__	The class name. This constant returns the class name as it was declared.
__METHOD__	The class method name. The method name is returned as it was declared.

Flow control statements

- Conditional statements are used to perform different actions based on different conditions.
- In PHP we have the following conditional statements:
  - **if statement** - executes some code only if a specified condition is true
  - **if...else statement** - executes code if a condition is true and another code if the condition is false
  - **if...elseif....else statement** - specifies a new condition to test, if the first condition is false
  - **switch statement** - selects one of many blocks of code to be executed

if Statement

- The if statement is used to execute some code **only if a specified condition is true**.
- **Syntax:**

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

```
}
```

Example:

```
<?php
    $number = 20;
    if ($number % 2 == 0)
    {
        echo "Number is Even";
    }
    if ($number % 2 != 0)
    {
        echo "Number is Odd";
    }
?>
```

Output:

Number is Even

if...else Statement

- Use the if....else statement to execute some code **if a condition is true and another code if the condition is false**
- **Syntax:**

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

Example:

```
<?php
    $number = 20;
    if ($number % 2 == 0)
    {
        echo "Number is Even";
    }
    else
    {
        echo "Number is Odd";
    }
```

```
?>
```

Output:

Number is Even

if...elseif....else Statement

- Use the if....elseif...else statement to **specify a new condition to test, if the first condition is false.**
- **Syntax:**

```
if (condition)
{
    code to be executed if condition is true;
}
elseif (condition)
{
    code to be executed if condition is true;
}
else
{
    code to be executed if condition is false;
}
```

Example:

```
<?php
    $a=10; $b=12; $c=30;
    if ($a > $b && $a > $c)
    {
        echo "a is max";
    }
    elseif ($b > $c)
    {
        echo "b is max";
    }
    else
    {
        echo "c is max";
    }
?>
```

Output:

c is max

switch Statement

- The switch statement is used to perform different actions based on different conditions.

- Use the switch statement to **select one of many blocks of code to be executed**.
- **Syntax:**

```
switch (n)
{
    case label1: code to be executed if n=label1; break;
            case label2: code to be executed if n=label2; break;
            case label3: code to be executed if n=label3; break;
            ...
            default: code to be executed if n is different from all labels;
}
```

- **How it works:** First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure.
- If there is a match, the block of code associated with that case is executed.
- Use **break** to prevent the code from running into the next case automatically.
- The **default** statement is used if no match is found.

Example:

```
<?php
    $a=6;
    switch($a)
    {
        case 1: echo "Monday"; break;
        case 2: echo "Tuesday"; break;
        case 3: echo "Wednesday"; break;
        case 4: echo "Thursday"; break;
        case 5: echo "Friday"; break;
        case 6: echo "Saturday"; break;
        case 7: echo "Sunday"; break;
        default: echo "Wrong Choice";
    }
?>
```

Output:

Saturday

Loops

- You want the same block of code to run over and over again. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.
- Loops execute a block of code while the specified condition is true.
- In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true.
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true.
- **for** - loops through a block of code a specified number of times.
- **foreach** - loops through a block of code for each element in an array.

while Loop

- The while loop executes a block of code as long as the specified condition is true.
- **Syntax:**

```
while (condition is true)
{
    code to be executed;
}
```

Example:

```
<?php
    $x = 1
    while($x<= 5)
    {
        echo "The number is: $x <br>";
        $x++;
    }
?>
```

Output:

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.
- **Syntax:**

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

Example:

```
<?php
    $x = 1;
    do
    {
        echo "The number is: $x <br>";
        $x++;
    } while ($x <= 5);
?>
```

Output:

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

for Loop

- PHP for loops execute a block of code a specified number of times.
- The loop is used when you know in advance how many times the script should run.
- **Syntax:**

```
for (init counter; test counter; increment counter)
{
    code to be executed;
}
```

- **Parameters:**
  - init counter: Initialize the loop counter value.
  - test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
  - increment counter: Increases the loop counter value.

Example:

```
<?php
    for ($x = 0; $x <= 10; $x++)
    {
        echo $x." ";
    }
?>
```

Output:

0 1 2 3 4 5 6 7 8 9 10

foreach Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

- **Syntax:**

```
foreach ($array as $value)
{
    code to be executed;
}
```

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Example:

```
<?php
    $colors = array("red", "green", "blue", "yellow");
    foreach ($colors as $value)
    {
        echo "$value <br>";
    }
?>
```

Output:

red
green
blue
yellow

Break and Continue

- The standard way to exit from a looping structure is for the main test condition to become false.
- The special commands break and continue offer an optional side exit from all the looping constructs, including while, do-while, and for.
- The break command exits the innermost loop construct that contains it.
- The continue command skips to the current iteration of the loop that contains it.

Example (Break): The following code prints the number 0 to 5.

```
<?php
    $i = 0;
    while ($i<100)
    {
        if ($i > 5)
        break;
        echo $i." ";
        $i=$i+1;
    }
?>
```

Output:

0 1 2 3 4 5 6

Example (Continue): The following code prints the number 0 to 9, except 5.

```
<?php
    for ($i = 0; $i<10; $i++)
    {
        if ($i == 5)
        {
                continue;
        }
        echo $i." ";
    }
?>
```

Output:

0 1 2 3 4 6 7 8 9

Difference between Break and Continue

BREAK	CONTINUE
Break is used to **terminate** the execution of the loop.	Continue is not used to terminate the execution of loop.
It **breaks** the iteration.	It **skips** the iteration.
When this statement is executed, control will come out from the loop and executes the statement immediate after loop.	When this statement is executed, it will not come out of the loop but moves/jumps to the next iteration of loop.
Break is used with loops as well as switch case.	Continue is only used in loops, it is not used in switch case.

Example :
<?php
for($A=1; $A<=5; $A++)
{
if($A>3)
{
break;
}
echo " " ,$A;
}
?>
Output : 1 2 3
	Example :
	<?php
	for($A=1; $A<=5; $A++)
	{
	if($A==3)
	{
	continue;
	}
	echo " " ,$A;
	}
	?>
	Output : 1 2 4 5