UNIT – 1:

Introducing PHP – What is PHP? Why use PHP? Evolution of PHP, Installing PHP, Other ways to run PHP, Creating your first script.
PHP Language Basics – Using variables, Understanding Data Types, Operators and Expressions, Constants. Decisions and Loops – Making Decisions, Doing Repetitive Tasks with Looping, Mixing Decisions and Looping with HTML.

Strings – Creating and Accessing Strings, Searching Strings, Replacing Text with Strings, Dealing with Upper and Lowercase, Formatting Strings.

Arrays – Creating Arrays, Accessing Array Elements, Looping through Arrays with for-each, Creating Function,

Reading Data in Web pages: setting up web pages to communicate with PHP, Handling Text Fields, Text Areas, Checkboxes, Radio Buttons, List Boxes, Password Controls, Image Maps, File Uploads, Buttons, and PHP Browser.

What is PHP

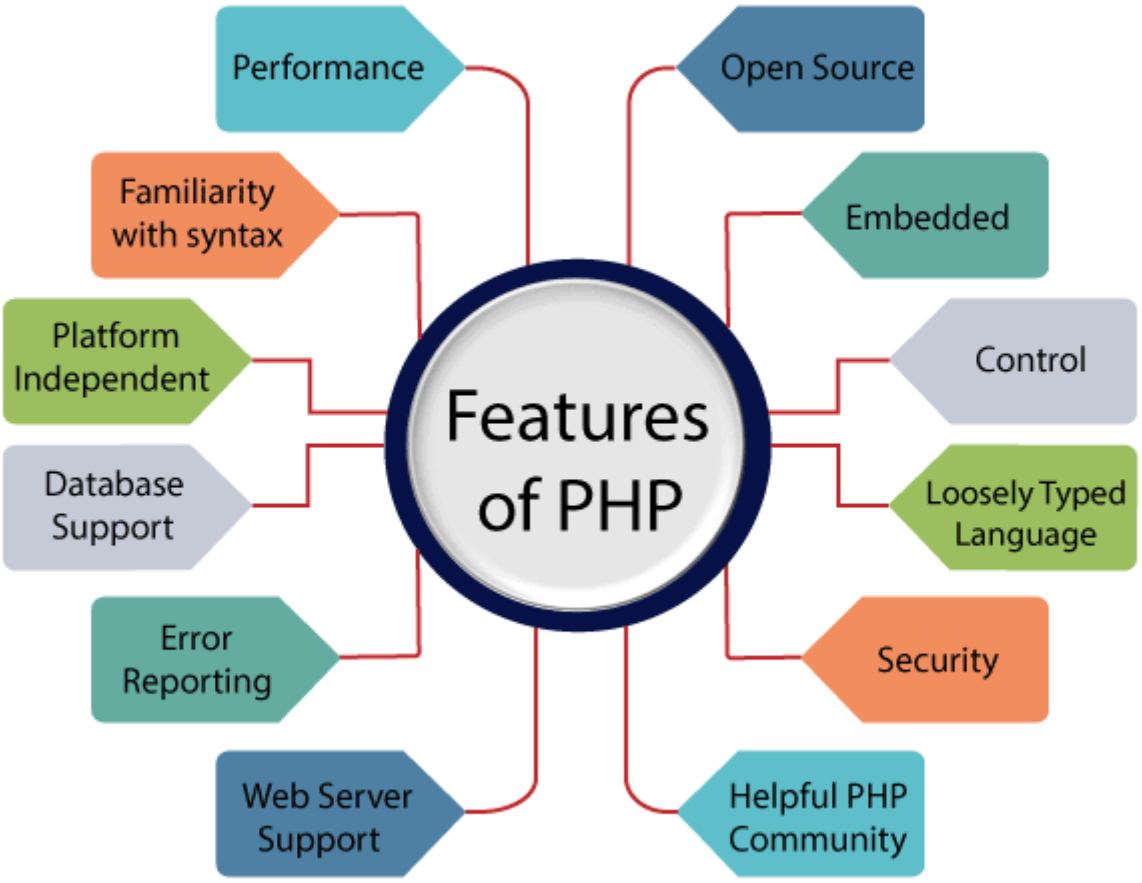
- PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side.
- PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page.).
- PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995.
- PHP stands for **Hypertext Preprocessor.** Which gives you a good idea of its core purpose: to process information and produce hypertext (HTML) as a result. (Developers love **recursive acronyms**, and PHP: Hypertext Preprocessor is a good example of one.)
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.
- PHP is simple and easy to learn language.

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

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

PHP Features

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


Features of PHP
Performance
Familiarity with syntax
Platform Independent
Database Support
Error Reporting
Web Server Support
Open Source
Embedded
Control
Loosely Typed Language
Security
Helpful PHP Community

Performance:

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

Open Source:

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

Familiarity with syntax:

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

Embedded:

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

Platform Independent:

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

Database Support:

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

Error Reporting -

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

Loosely Typed Language:

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

Web servers Support:

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

Security:

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

Control:

Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

A Helpful PHP Community:

It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

Web Development

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

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

Prerequisite

Before learning PHP, you must have the basic knowledge of **HTML, CSS,** and **JavaScript**.

HTML - HTML is used to design static webpage.

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

JavaScript - JavaScript is used to design an interactive website.

Version	Release date	Supported until	Notes
1.0	8 June 1995		Officially called "Personal Home Page Tools (PHP Tools)". This is the first use of the name "PHP".
2.0	1 November 1997		Officially called "PHP/FI 2.0". This is the first release that could actually be characterised as PHP, being a standalone language with many features that have endured to the present day.
3.0	6 June 1998	20 October 2000	Development moves from one person to multiple developers. Zeev Suraski and Andi Gutmans rewrite the base for this version.
4.0	22 May 2000	23 June 2001	Added more advanced two-stage parse/execute tag-parsing system called the Zend engine.
4.1	10 December 2001	12 March 2002	Introduced "superglobals" ($_GET, $_POST, $_SESSION, etc.)
4.2	22 April 2002	06 September 2002	Disabled register_globals by default. Data received over the network is not inserted directly into the global namespace anymore, closing possible security holes in applications.
4.3	27 December 2002	31 March 2005	Introduced the command-line interface (CLI), to supplement the CGI.
4.4	11 July 2005	07 August 2008	Fixed a memory corruption bug, which required breaking binary compatibility with extensions compiled against PHP version 4.3.x.
5.0	13 July 2004	05 September 2005	Zend Engine II with a new object model.
5.1	24 November 2005	24 August 2006	Performance improvements with introduction of compiler variables in re-engineered PHP Engine. Added PHP Data Objects (PDO) as a consistent interface for accessing databases.
5.2	02 November 2006	06 January 2011	Enabled the filter extension by default. Native JSON support.
5.3	30 June 2009	14 August 2014	Namespace support; late static bindings, jump label (limited goto), anonymous functions, closures, PHP archives (phar), garbage collection for circular references, improved Windows support, sqlite3, mysqlnd as a replacement for libmysql as underlying library for the extensions that work with MySQL, fileinfo as a replacement for mime_magic for better MIME support, the Internationalization extension, and deprecation of ereg extension.

5.4	01 March 2012	03 September 2015	Trait support, short array syntax support. Removed items: register_globals, safe_mode, allow_call_time_pass_reference, session_register(), session_unregister() and session_is_registered(). Built-in web server.Several improvements to existing features, performance and reduced memory requirements.
5.5	20 June 2013	10 July 2016	Support for generators, finally blocks for exceptions handling, OpCache (based on Zend Optimizer+) bundled in official distribution.
5.6	28 August 2014	31 December 2018	Constant scalar expressions, variadic functions, argument unpacking, new exponentiation operator, extensions of the use statement for functions and constants, new phpdbg debugger as a SAPI module, and other smaller improvements.
6.x	Not released	N/A	Abandoned version of PHP that planned to include native Unicode support.
7.0	03 December 2015	10 January 2019	Zend Engine 3 (performance improvements and 64-bit integer support on Windows), uniform variable syntax,AST-based compilation process, added Closure::call(), bitwise shift consistency across platforms, ?? (null coalesce) operator,[Unicode code point escape syntax,return type declarations,scalar type (integer, float, string and boolean) declarations, <=>"spaceship"three-way comparison operator,generator delegation, anonymous classes, simpler and more consistently available CSPRNG API, replacement of many remaining internal PHP "errors" with the more modern exceptions,and shorthand syntax for importing multiple items from a namespace.
7.1	01 December 2016	01 December 2019	void return type,class constant visibility modifiers
7.2	30 November 2017	30 November 2020	Object parameter and return type declaration,Libsodium extension,abstract method overriding, parameter type widening
7.3	06 December 2018	06 December 2021	Flexible Heredoc and Nowdoc syntax,support for reference assignment and array deconstruction with list(),PCRE2 support, hrtime() function
7.4	28 November 2019	28 November 2022	Typed properties 2.0,preloading,null-coalescing assignment operator, improve openssl_random_pseudo_bytes,Weak References, FFI – foreign function interface, always available hash extension,password hash registry,multibyte string splitting, reflection for references,unbundle ext/wddx, new custom object serialization mechanism

8.0	26 November 2020	26 November 2023	Just-In-Time (JIT) compilation,arrays starting with a negative index,stricter/saner language semantics (validation for abstract trait methods),[saner string to number comparisons, saner numeric strings,TypeError on invalid arithmetic/bitwise operators,reclassification of various engine errors,consistent type errors for internal functions, fatal error for incompatible method signatures, locale-independent float to string conversion,variable syntax tweaks, attributes,named arguments,match expression,constructor property promotion,union types,mixed type,static return type, nullsafe operator, non-capturing catches, throw expression,JSON extension is always available.
8.1	?? November 2021	?? November 2024	Explicit octal integer literal notation

Install PHP

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

- **WAMP** for Windows
- **LAMP** for Linux
- **MAMP** for Mac
- **SAMP** for Solaris
- **FAMP** for FreeBSD
- **XAMPP** (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail, etc.

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

How to install XAMPP server on windows

We will learn how to install the XAMPP server on windows platform step by step. Follow the below steps and install the XAMPP server on your system.

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl.

https://www.apachefriends.org/download.html

What is XAMPP?

XAMPP is an abbreviation where *X stands for Cross-Platform, A stands for Apache, M stands for MYSQL, and the Ps stand for PHP and Perl*, respectively. It is an open-source package of web solutions that includes Apache distribution for many servers and command-line executables along with modules such as Apache server, MariaDB, PHP, and Perl.

XAMPP helps a local host or server to test its website and clients via computers and laptops before releasing it to the main server. It is a platform that furnishes a suitable environment to test and verify the working of projects based on Apache, Perl, MySQL database, and PHP through the system of the host itself. Among these technologies, Perl is a programming language used for web development, PHP is a backend scripting language, and MariaDB is the most vividly used database developed by MySQL.

INSTALLATION PROCESS OF XAMPP

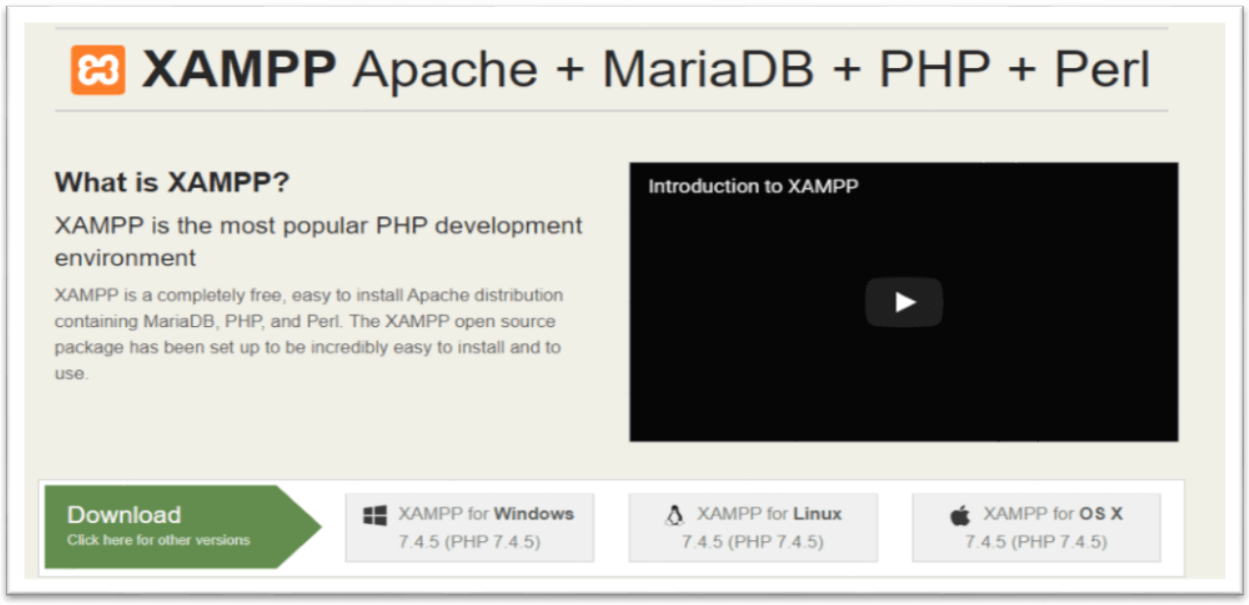
XAMPP is a cross-platform stack of software that provides web solutions based on technologies like MariaDB, Apache Server, Perl, and PHP. Further, it is supported by many file formats, such as .EXE, .ZIP and .7z- .7zip. Out of the three, the .EXE extension is the easiest to operate upon while installation. In this topic, we will discuss steps to be followed to download and install XAMPP software successfully on your desktops. Since it is a cross-platform software, it is supported by a number of operating systems, including Windows, Linux, and MAC OS. The process to be followed for installation of XAMPP will be explained for all three operating systems:

The installation process in Windows

STEP 1- Open any web browser and visit

https://www.apachefriends.org/index.html. On the home page, you can find the option to download XAMPP for three platforms- Windows, MAC, and Linux. Click on **XAMPP for Windows**. The latest version available on the website is **7.4.5.**

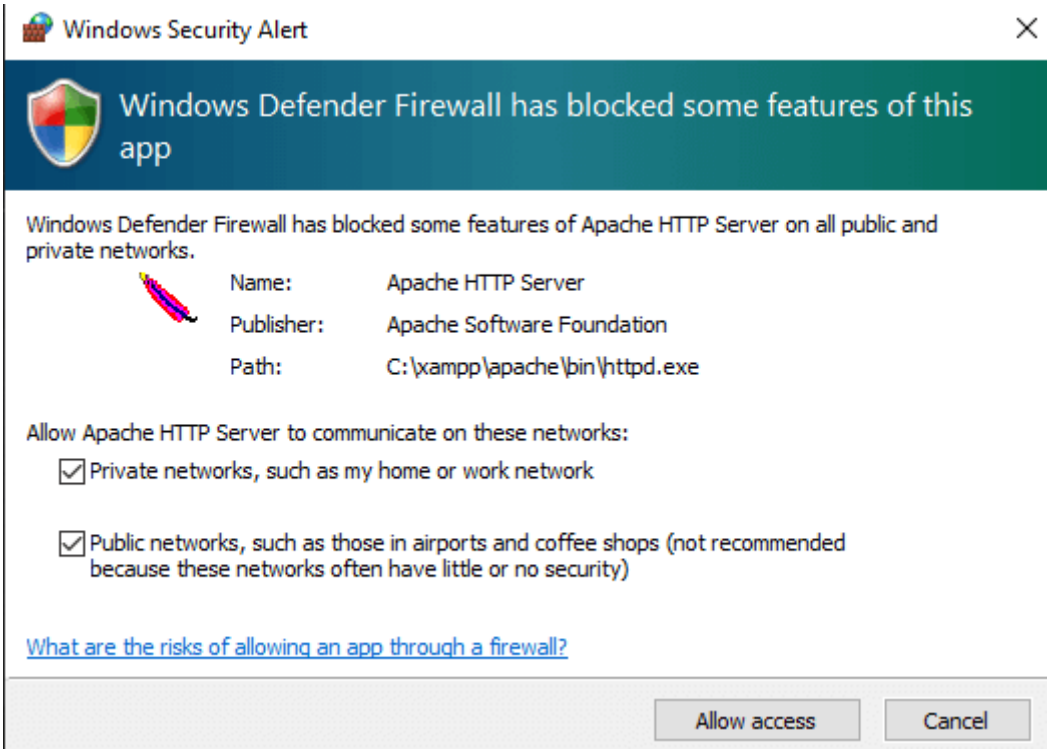
As soon as you click on it, a message displaying the automatic start of download appears on the screen.



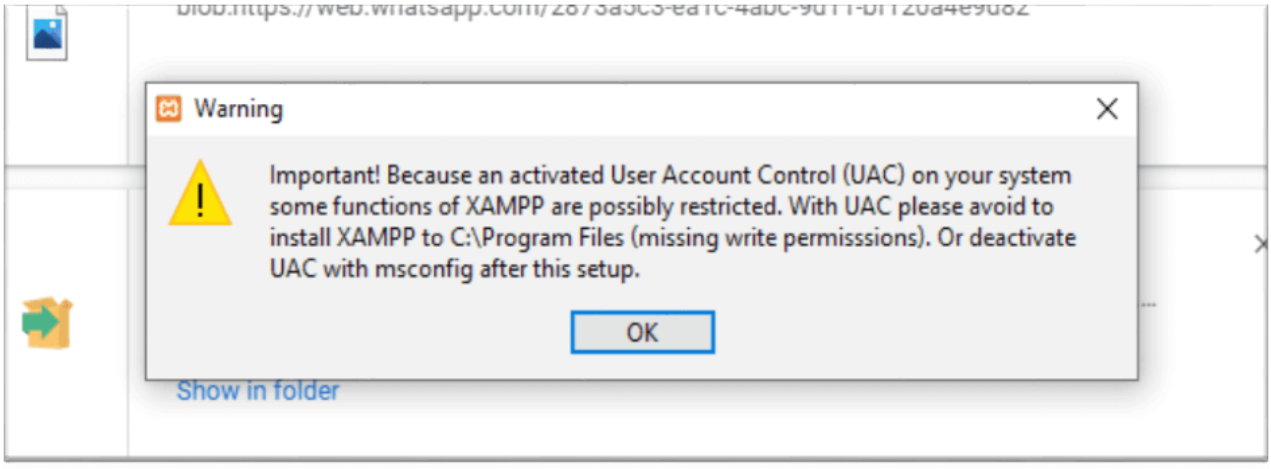
STEP 2- After the download is completed, double click the .exe extension file to start the process of installation.

STEP 3- A pop-up screen with the message asking you to allow to make changes on your desktop appears. Click "YES" to continue the process.

STEP 4- Click to Allow access or deactivate the firewall and any other antivirus software because it can hamper the process of installation. Thus, it is required to temporarily disable any antivirus software or security firewall till the time all the XAMPP components have been installed completely.

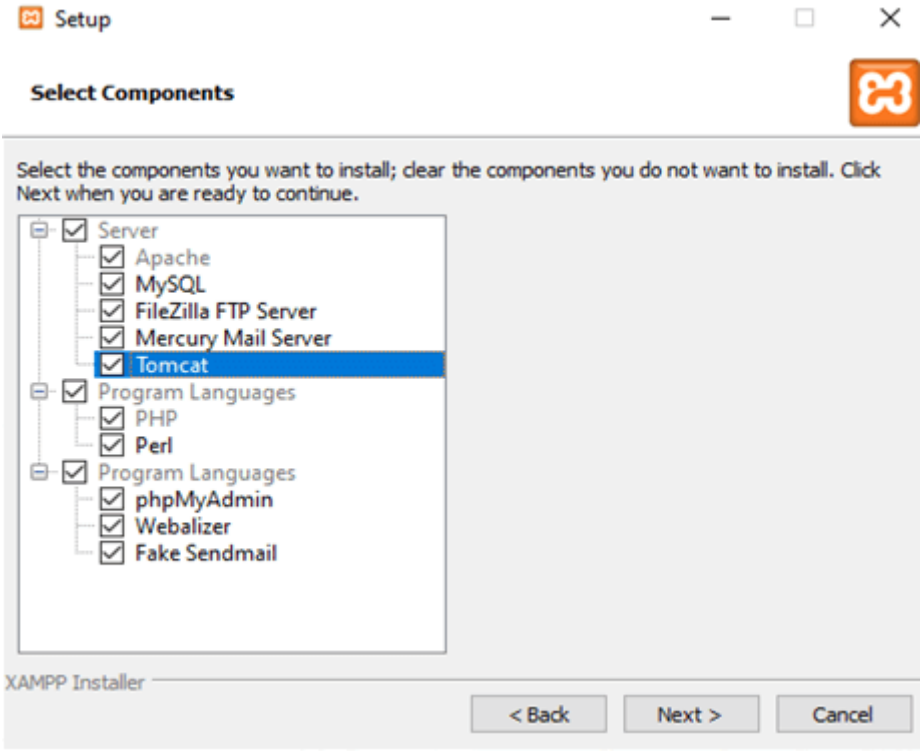


STEP 5- Just before the installation, a pop-up window appears with a warning to disable UAC. User Account Control (UAC) interrupts the XAMPP installation because it restricts the access to write to the C: drive. Therefore, it is suggested to disable it for the period of installation.

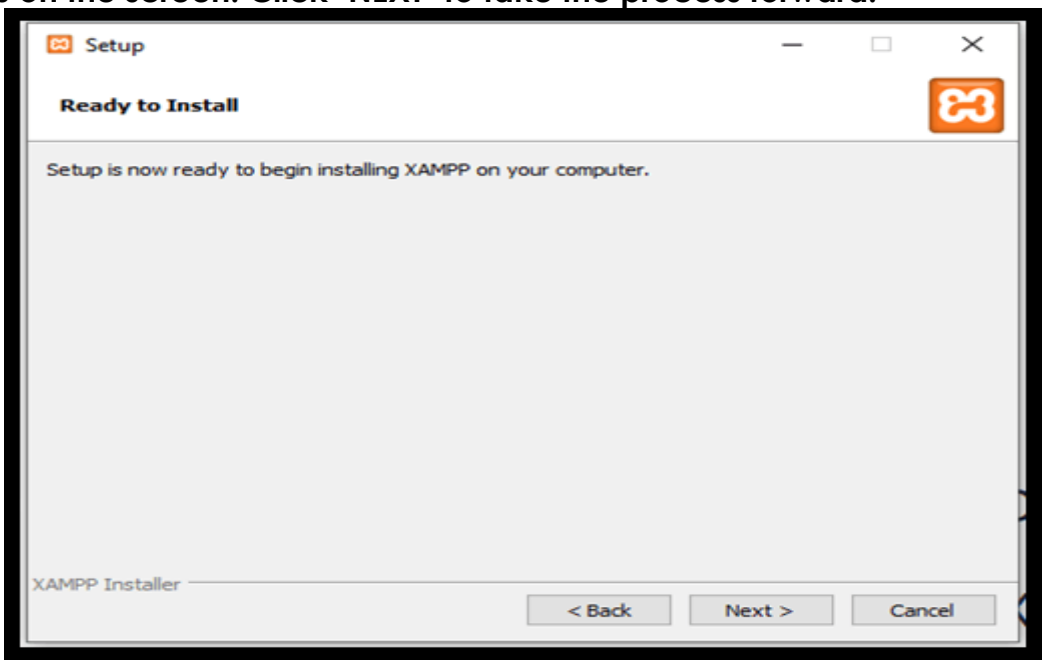


STEP 6- After clicking the .exe extension file, the XAMPP setup wizard opens spontaneously. Click on "NEXT" to start the configuration of the settings.

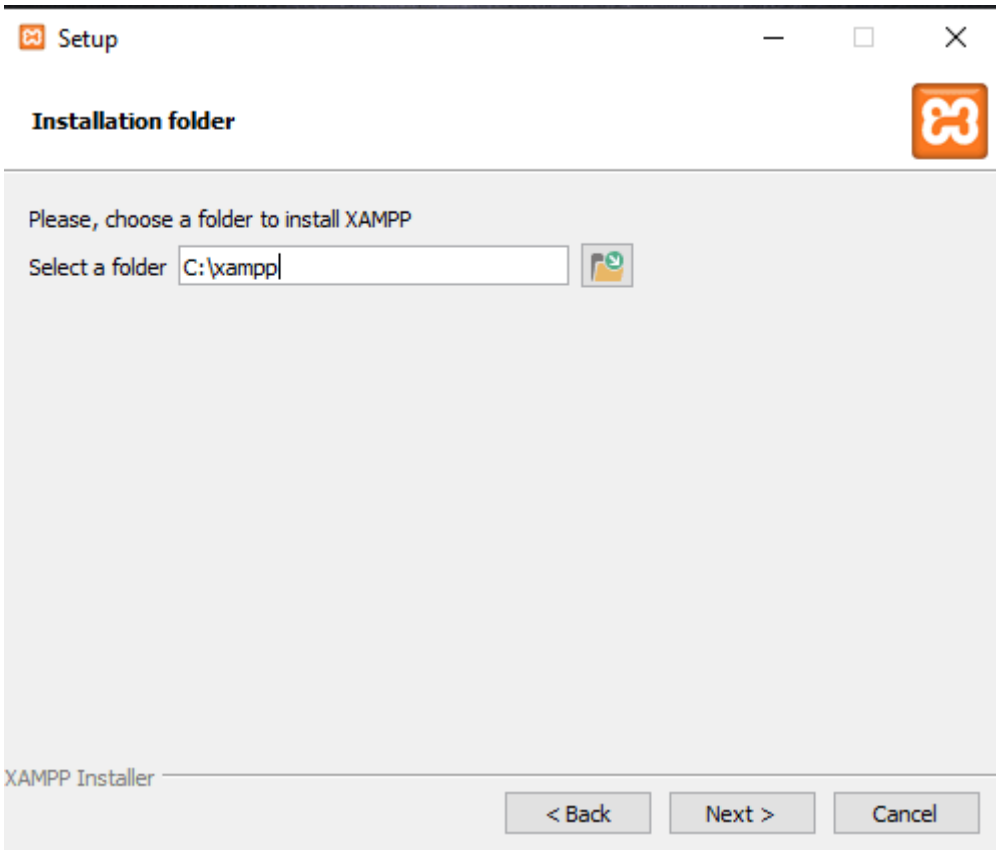
STEP 7- After that, a 'Select Components' panel appears, which gives you the liberty to choose amongst the separate components of the XAMPP software stack for the installation. To get a complete localhost server, it is recommended to install using the default options of containing all available components. Click "NEXT" to proceed further.



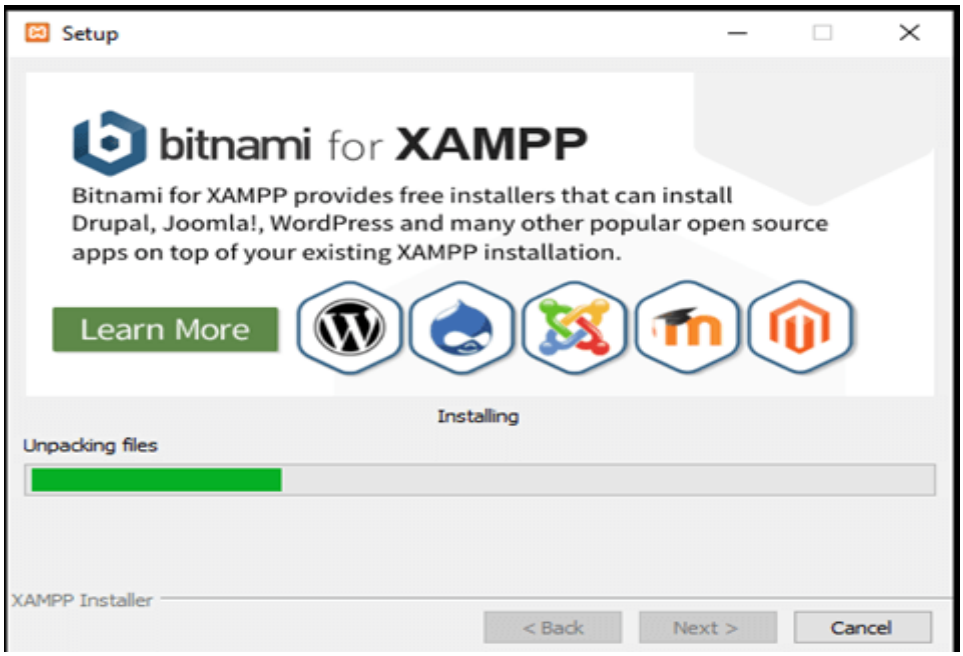
STEP 8- The setup is now ready to install, and a pop-up window showing the same appears on the screen. Click "NEXT" to take the process forward.



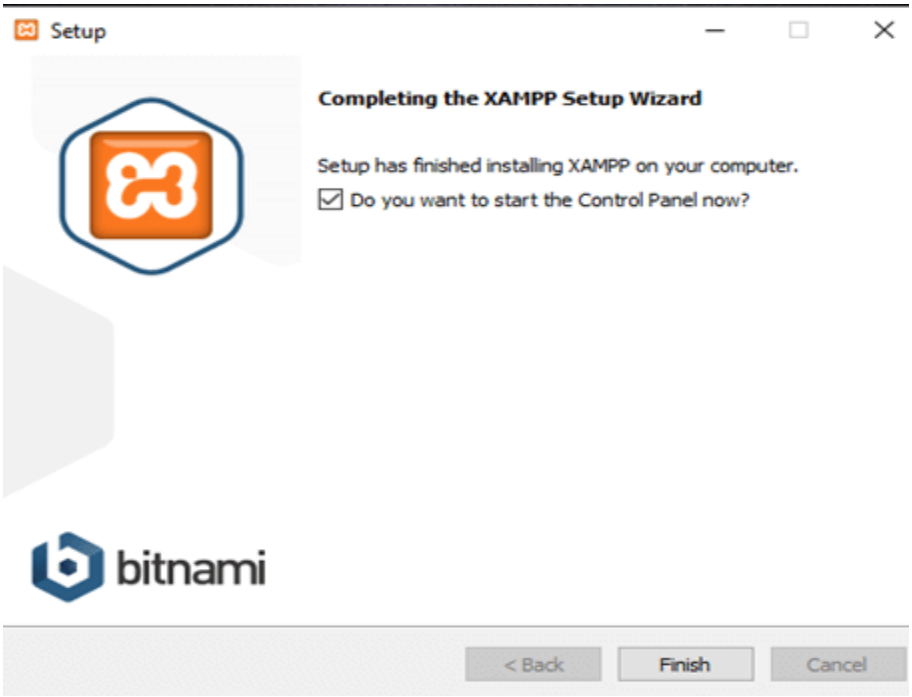
STEP 9- Select the location where the XAMPP software packet needs to be installed. The original setup creates a folder titled XAMPP under C:\ for you. After choosing a location, click "NEXT".



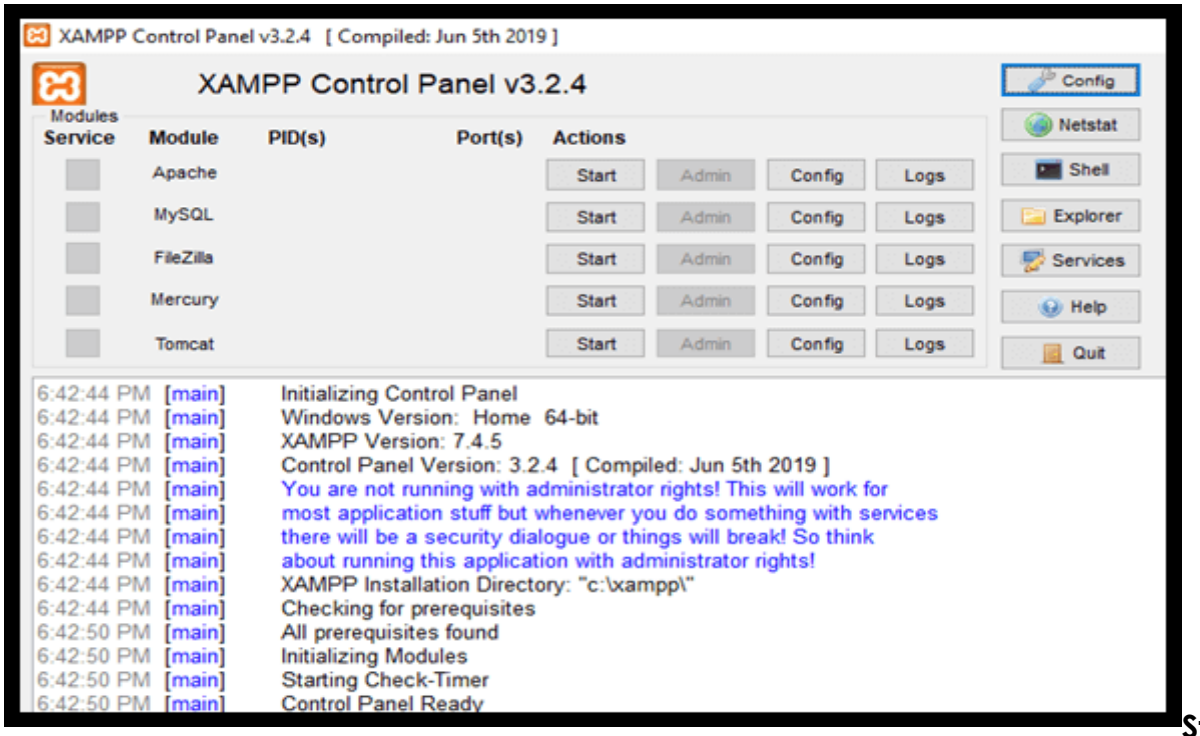
STEP 10- After choosing from all the previously mentioned preferences (like language and learn more bitnami dialogue box) click to start the installation. The setup wizard will unpack and install the components to your system. The components are saved to the assigned directory. This process may takes a few minutes to complete. The progress of the installation in terms of percentage is visible on the screen.



STEP 11- After the successful installation of the XAMPP setup on your desktop, press the "FINISH" button.



On clicking the FINISH button, the software automatically launches, and the CONTROL PANEL is visible. The image below shows the appearance of the final result.



Steps to create a PHP page:-

1. Open the XAMPP directory present in C Drive and choose the htdocs folder (C:\xampp\htdocs for standard installations). This directory contains all the data required to run a web page.
2. Please create a new folder **ROOLNUMBER** for the test page in htdocs.
3. Open notepad and type the following code and save the file in the Test folder.
4. Make sure you have saved the file with .php extension.

```
<!doctype html>

<html>
    <head>
        <title> Test Page</title>
    </head>
    <body>
        <p> XAMPP Server runs successfully</p>
    </body>
```

</html>

1. Open the XAMPP control panel and start the apache module.
2. Open your browser and type localhost/ROLLNUMBER/test.php in the URL tab. If your browser prints 'XAMPP Server runs successfully', it means XAMPP is successfully installed and correctly configured.

How to run PHP code in XAMPP

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

Note: PHP statements ends with semicolon (;).

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

```
<?php
        //your code here
?>
```

A simple PHP example where we are writing some text using PHP echo command.

File: first.php

<!DOCTYPE>

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

Output:

Hello First PHP (* html heading h2 will be applied with text)

PHP Case Sensitivity

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

Example, where all three echo statements are equal and valid:

<!DOCTYPE>

```
<html>
   <body>
      <?php
         echo "Hello world using echo </br>";
         ECHO "Hello world using ECHO </br>";
         EcHo "Hello world using EcHo </br>";
      ?>
   </body>
</html>
```

Output:

Hello world using echo
Hello world using ECHO
Hello world using EcHo

Look at the below example that the variable names are case sensitive. You can see the example below that only the second statement will display the value of the $color variable. Because it treats $color, $ColoR, and $COLOR as three different variables:

```
<html>

<body>

<?php

$color = "blue";

echo "My car is ". $ColoR ."</br>";

echo "My pen is ". $color ."</br>";

echo "My Phone is ". $COLOR ."</br>";

?>

</body>

</html>
```

Output:

Notice: Undefined variable: ColoR in p2.php on line 5
My Car is
My Pen is blue
Notice: Undefined variable: COLOR in **p2.php** on line **7**
My Phone is

PHP Echo

PHP echo is a language construct, not a function. Therefore, you don't need to use parenthesis with it. But if you want to use more than one parameter, it is required to use parenthesis.

The syntax of PHP echo is given below: **void echo (string $arg1 [, string $...])**

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

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

PHP echo: printing string

File: echo1.php

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

PHP echo: printing multi line string

File: echo2.php

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

```
?>
```

PHP echo: printing variable value

File: echo3.php

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

PHP Print

- Like PHP echo, PHP print is a language construct, so you don't need to use parenthesis with the argument list.
- Print statement can be used with or without parentheses: print and print().
- Unlike echo, it always returns 1.
- The syntax of PHP print is given below: **int print(string $arg)**
- PHP print statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc.
- Some important points that you must know about the echo statement are:
    1. print is a statement, used as an alternative to echo at many times to display the output.
    2. print can be used with or without parentheses.
    3. print always returns an integer value, which is 1.
    4. Using print, we cannot pass multiple arguments.
    5. print is slower than the echo statement.

PHP print: printing string

File: print1.php

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

Output: Hello by PHP print Hello by PHP print()

PHP print: printing variable value
File: print2.php

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

Output:

Message is: Hello print() in PHP

PHP echo and print Statements

We frequently use the echo statement to display the output. There are two basic ways to get the output in PHP:

- echo
- print

echo and print are language constructs, and they never behave like a function. Therefore, there is no requirement for parentheses. However, both the statements can be used with or without parentheses. We can use these statements to output variables or strings.

Difference between echo and print

echo

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

print

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

You can see the difference between echo and print statements with the help of

the following programs.

For Example (Check multiple arguments)

- You can pass multiple arguments separated by a comma (,) in echo. It will not generate any syntax error.

```
<?php
        $fname = "RUSMUS";
        $lname = "LERDORF";
        echo "My name is: ".$fname,$lname;
 ?>
```

Output: My name is: RUSMUS LERDORF

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

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

Output:

Parse error: syntax error, unexpected ',' in **/opt/lampp/htdocs/Lab/testecho.php** on line **4**

For Example (Check Return Value)

- echo statement does not return any value. It will generate an error if you try to display its return value.

```
<?php
    $lang = "PHP";
    $ret = echo $lang." is a web development language.";
    echo "</br>";
    echo "Value return by print statement: ".$ret;
?>
```

Output: Parse error: syntax error, unexpected 'echo' (T_ECHO) in

/opt/lampp/htdocs/Lab/testecho.php on line 3

As we already discussed that print returns a value, which is always 1.

```
<?php
    $lang = "PHP";
    $ret = print $lang." is a web development language.";
    print "</br>";
print "Value return by print statement: ".$ret;
?>
```

Output:

PHP is a web development language.
Value return by print statement: 1

PHP Variables

In PHP, a variable is declared using a **$ sign** followed by the variable name. Here, some important points to know about variables:

- As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
- After declaring a variable, it can be reused throughout the code.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

$variablename=value;

Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

PHP Variable: Declaring string, integer, and float

- Example to store string, integer, and float values in PHP variables.

File: variable1.php

```
<?php
$str="RADHA ";
$id=201;
$percentage=96.5;
echo "string type is: $str <br/>";
echo "integer type is: $id <br/>";
echo "float type is: $percentage <br/>";
?>
```

Output:

```
string type is: RADHA
integer type is: 201
float type is: 96.5
```

PHP Variable: Sum of two variables

File: variable2.php

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

Output:

```
11
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

PHP Variable: Rules

1. PHP variables must start with letter or underscore only.
2. PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)
echo "$a <br/> $_b";
?>
```

Output:

```
hello
hello
```

File: variableinvalid.php

```
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)
echo "$4c <br/> $*d";
?>
```

Output:

Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable (T_VARIABLE) or '$' in **/opt/lampp/htdocs/Lab/testecho.php** on line **2**

PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

Local variable

- The variables that are declared within a function are called local variables for that function.
- These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.
- A variable declaration outside the function with the same name is completely different from the variable declared inside the function.

File: local_variable1.php

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

Output:

Local variable declared inside the function is: 45

File: local_variable2.php

```
<?php
   function mytest()
   {
      $lang = "PHP";
      echo "Web development language: " .$lang;
   }
   mytest();
   //using $lang (local variable) outside the function will generate an error
   echo $lang;
?>
```

Output:

Web development language: PHP
Notice: Undefined variable: lang in
/opt/lampp/htdocs/Lab/local_variable2 .php on line **9**

Global variable

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

Example:

File: global_variable1.php

```
<?php
$name = " RADHA ";        //Global Variable
function global_var()
{
global $name;
echo "Variable inside the function: ". $name;
echo "</br>";
}
global_var();
echo "Variable outside the function: ". $name;
?>
```

Output:

Variable inside the function: RADHA
Variable outside the function: RADHA

Note: Without using the global keyword, if you try to access a global variable inside the function, it will generate an error that the variable is undefined.

Using $GLOBALS instead of global

Another way to use the global variable inside the function is predefined $GLOBALS array.

Example:

File: global_variable2.php

```
<?php
$num1 = 5;      //global variable
$num2 = 13;     //global variable
function global_var()
{
$sum = $GLOBALS['num1'] + $GLOBALS['num2'];
echo "Sum of global variables is: " .$sum;
}
global_var();
?>
```

Output:

Sum of global variables is: 18

Note: If two variables, local and global, have the same name, then the local variable has higher priority than the global variable inside the function.

Example: File: global_variable2.php

```
<?php
   $x = 5;
   function mytest()
   {
      $x = 7;
      echo "value of x: " .$x;
   }
   mytest();
?>
```

Output: value of x: 7

Static variable

- It is a feature of PHP to delete the variable, once it completes its execution and memory is freed.
- Sometimes we need to store a variable even after completion of function execution.
- Therefore, another important feature of variable scoping is static variable.
- We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.
- Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope.

Example: static_variable.php

```
<?php
function static_var()
{
static $num1 = 3;       //static variable
$num2 = 6;          //Non-static variable
//increment in non-static variable
$num1++;
//increment in static variable
$num2++;
echo "Static: " .$num1 ."</br>";
echo "Non-static: " .$num2 ."</br>";
}
//first function call
static_var();
//second function call
static_var();
?>
```

Output:

```
Static: 4
Non-static: 7
Static: 5
Non-static: 7
```

Understanding Data Types

All data stored in PHP variables fall into one of **eight** basic categories, known as data types . A variable's data type determines what operations can be carried out on the variable's data, as well as the amount of memory needed to hold the data

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

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

Changing a Variable ' s Data Type

Here ' s some example code that converts a variable to various different types using settype() :

$test_var = 8.23; echo $test_var . " < br / > "; // Displays "8.23" settype($test_var, "string"); echo $test_var . " < br / > "; // Displays "8.23" settype($test_var, "integer"); echo $test_var . " < br / > "; // Displays "8" settype($test_var, "float"); echo $test_var . " < br / > "; // Displays "8" settype($test_var, "boolean"); echo $test_var . " < br / > "; // Displays "1"

Changing Type by Casting

In the following example, a variable ' s value is cast to various different types at the time that the value is displayed:

$test_var = 8.23; echo $test_var . " < br / > "; // Displays "8.23"

echo (string) $test_var . " < br / > "; // Displays "8.23"

echo (int) $test_var . " < br / > "; // Displays "8"

echo (float) $test_var . " < br / > "; // Displays "8.23"

echo (boolean) $test_var . " < br / > "; // Displays "1"

Operators and Expressions

So an operator is a symbol that manipulates one or more values, usually producing a new value in the process. Meanwhile, an expression in PHP is anything that evaluates to a value; this can be any combination of values, variables, operators, and functions.

Operator Types

- PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

- PHP 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 - $y	Difference$Xand$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 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 gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets 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

PHP 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
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y.

PHP 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

PHP 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
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true

!	Not	!$x	True if $x is not true

PHP 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

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to

			$y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

---

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

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

Decisions and Loops

Making Decisions Like most programming languages, PHP lets you write code that can make decisions based on the result of an expression. This allows you to do things like test if a variable matches a particular value, or if a string of text is of a certain length. In essence, if you can create a test in the form of an expression that evaluates to either true or false , you can use that test to

make decisions in your code.

PHP gives you a number of statements that you can use to make decisions:

- The if statement
- The else and else if statements
- The switch statement

Simple Decisions with the if Statement The easiest decision - making statement to understand is the if statement. The basic form of an if construct is as follows:

```
if ( expression )
{
// Run this code
}
// More code here
```

Here ’ s a simple real - world example:

```
$widgets = 23;
if ( $widgets == 23 )
{
echo “We have exactly 23 widgets in stock!”;
}
```

Providing an Alternative Choice with the else Statement

As you ’ ve seen, the if statement allows you to run a block of code if an expression evaluates to true . If the expression evaluates to false , the code is skipped.

You can enhance this decision - making process by adding an else statement to an if construction. This lets you run one block of code if an expression is true , and a different block of code if the expression is false .

For example:

```
if ( $widgets > = 10 )
{
echo “We have plenty of widgets in stock.”;
}
else
```

```
{
 echo "Less than 10 widgets left. Time to order some more!";
}
```

Sometimes you want to test an expression against a range of different values, carrying out a different task depending on the value that is matched. Here ' s an example, using the if , elseif , and else statements:

```
if ( $userAction == "open" )
{
// Open the file
}
else if ( $userAction == "save" )
{ // Save the file
 }
else if ( $userAction == "close" )
{ // Close the file
}
else if ( $userAction == "logout" )
{
// Log the user out
}
 else
 {
print "Please choose an option";
}
```

PHP provides a more elegant way to run these types of tests: the switch statement. With this statement, you include the expression to test only once, then provide a range of values to test it against, with corresponding code blocks to run if the values match. Here ' s the preceding example rewritten using switch :

```
 switch ( $userAction )
{
 case "open":
 // Open the file
```

```
break;
 case "save":
 // Save the file
 break;
case "close":
 // Close the file
 break;
 case "logout":
// Log the user out
 break;
default: print "Please choose an option";
 }
```

Doing Repetitive Tasks with Looping

The basic idea of a loop is to run the same block of code again and again, until a certain condition is met. As with decisions, that condition must take the form of an expression. If the expression evaluates to true , the loop continues running. If the expression evaluates to false , the loop exits, and execution continues on the first line following the loop ' s code block.

You look at three main types of loops in this chapter:

- while loops
- do...while loops
- for loops

Mixing Decisions and Looping with HTML

You also learned that you can switch between displaying HTML markup and executing PHP code by using the < ?php ... ? > tags.

Strings

you can use PHP ' s string functions to:

- Search for text within a string
- Replace some text within a string with another string of text
- Format strings so that they ' re easier to read or work with
- Encode and decode strings using various popular encoding formats

$myString = `hello`; In this example, the string literal (hello) is enclosed in single quotation marks (`).

You can also use double quotation marks ("),

as follows: $myString = "hello";

Single and double quotation marks work in different ways. If you enclose a string in single quotation marks, PHP uses the string exactly as typed. However, double quotation marks give you a couple of extra features:

- Any variable names within the string are parsed and replaced with the variable ' s value
- You can include special characters in the string by escaping them

PHP String Functions

The PHP string functions are part of the PHP core. No installation is required to use these functions.

Function	Description
addcslashes()	Returns a string with backslashes in front of the specified characters
addslashes()	Returns a string with backslashes in front of predefined characters
bin2hex()	Converts a string of ASCII characters to hexadecimal values
chop()	Removes whitespace or other characters from the right end of a string
chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts
convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a

	string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string hashing
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
get_html_translation_table()	Returns the translation table used by htmlspecialchars() and htmlentities()
hebrev()	Converts Hebrew text to visual text
hebrevc()	Converts Hebrew text to visual text and new lines (\n) into
hex2bin()	Converts a string of hexadecimal values to ASCII characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
htmlspecialchars_decode()	Converts some predefined HTML entities to characters
htmlspecialchars()	Converts some predefined characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the

	left side of a string
md5()	Calculates the MD5 hash of a string
md5_file()	Calculates the MD5 hash of a file
metaphone()	Calculates the metaphone key of a string
money_format()	Returns a string formatted as a currency string
nl_langinfo()	Returns specific local information
nl2br()	Inserts HTML line breaks in front of each newline in a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
quoted_printable_decode()	Converts a quoted-printable string to an 8-bit string
quoted_printable_encode()	Converts an 8-bit string to a quoted printable string
quotemeta()	Quotes meta characters
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string
sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings
soundex()	Calculates the soundex key of a string
sprintf()	Writes a formatted string to a variable

sscanf()	Parses input from a string according to a format
str_getcsv()	Parses a CSV string into an array
str_ireplace()	Replaces some characters in a string (case-insensitive)
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times
str_replace()	Replaces some characters in a string (case-sensitive)
str_rot13()	Performs the ROT13 encoding on a string
str_shuffle()	Randomly shuffles all characters in a string
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
strcoll()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found
strip_tags()	Strips HTML and PHP tags from a string
stripcslashes()	Unquotes a string quoted with addcslashes()
stripslashes()	Unquotes a string quoted with addslashes()
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)

strlen()	Returns the length of a string
strnatcasecmp()	Compares two strings using a "natural order" algorithm (case-insensitive)
strnatcmp()	Compares two strings using a "natural order" algorithm (case-sensitive)
strncasecmp()	String comparison of the first n characters (case-insensitive)
strncmp()	String comparison of the first n characters (case-sensitive)
strpbrk()	Searches a string for any of a set of characters
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)
strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start

	position (binary safe and optionally case-sensitive)
substr_count()	Counts the number of times a substring occurs in a string
substr_replace()	Replaces a part of a string with another string
trim()	Removes whitespace or other characters from both sides of a string
ucfirst()	Converts the first character of a string to uppercase
ucwords()	Converts the first character of each word in a string to uppercase
vfprintf()	Writes a formatted string to a specified output stream
vprintf()	Outputs a formatted string
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

Arrays

Create an Array in PHP

In PHP, the `array()` function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

$authors = array(“Steinbeck”, “Kafka”, “Tolkien”, “Dickens”);

If you want to create an associative array, where each element is identified by a string index rather than a number, you need to use the => operator, as follows:

$myBook = array(“title” = > “The Grapes of Wrath”,

"author" = > "John Steinbeck",

"pubYear" = >1939);

PHP Indexed Arrays

❮ PreviousNext ❯

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a `for` loop, like this:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
```

?> Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a `foreach` loop, like this:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
```

```
  echo "<br>";
}
?>
```

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
```

```
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

We can also put a `for` loop inside another `for` loop to get the elements of the $cars array (we still have to point to the two indices):

Example

```
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

Accessing Array Elements

Once you ’ ve created your array, how do you access the individual values inside it? In fact, you do this in much the same way as you access the individual characters within a string:

$authors = array(“Steinbeck”, “Kafka”, “Tolkien”, “Dickens”);

$myAuthor = $authors[0]; // $myAuthor contains “Steinbeck”

$anotherAuthor = $authors[1]; // $anotherAuthor contains “Kafka”

In other words, you write the variable name, followed by the index of the element in square brackets. If you want to access the elements of an associative array, simply use string indices rather than numbers:

```
$myBook = array( "title" = > "The Grapes of Wrath",
                "author" = > "John Steinbeck", "pubYear" = >1939 );
$myTitle = $myBook["title"]; // $myTitle contains "The Grapes of Wrath"
$myAuthor = $myBook["author"]; // $myAuthor contains "Steinbeck"
```

You don ' t have to use literal values within the square brackets;

you can use any expression, as long as it evaluates to an integer or string as appropriate:

```
$authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );
$pos = 2;
echo $authors[$pos + 1]; // Displays "Dickens"
```

PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

Advantage of PHP Functions

Code Reusability: PHP functions are defined only once and can be invoked many times, like in other programming languages.

Less Code: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

Easy to understand: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare user-defined functions.

Syntax

```
1. function functionname(){
2. //code to be executed
3. }
```

Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.

Handling HTML Forms with PHP

An HTML form, or Web form, is simply a collection of HTML elements embedded within a standard Web page. By adding different types of elements, you can create different form fields, such as text fields,pull - down menus, checkboxes, and so on.
All Web forms start with an opening < form > tag, and end with a closing < /form > tag:
< form action=”myscript.php” method=”post” >
< !-- Contents of the form go here -- >
< /form >
By the way, the second line of code in this example is an HTML comment – – everything between the
<!-- and --> is ignored by the Web browser.
Notice that there are two attributes within the opening <form> tag:

action tells the Web browser where to send the form data when the user fills out and
submits the form. This should either be an absolute URL (such as http://www.example.com/myscript.php) or a relative URL (such as myscript.php , /myscript.php , or ../scripts/myscript.php).
The script at the specified URL should be capable of accepting and processing the form data; more on this in a moment.
method tells the browser how to send the form data. You can use two methods: get is useful for sending small amounts of data and makes it easy for the user to resubmit the form, and post can send much larger amounts of form data.

The created form fields include:
A text input field –– This allows the user to enter a single line of text. You

can optionally prefill the field with an initial value using the value attribute (if you don't want to do this, specify an empty string for the value attribute, or leave the attribute out altogether):

```
<label for="textField">A text input field</label>
<input type="text" name="textField" id="textField" value="" />
```

A password field — This works like a text input field, except that the entered text is not
displayed. This is, of course, intended for entering sensitive information such as passwords. Again, you can prefill the field using the value attribute, though it's not a good idea to do this because the password can then be revealed by viewing the page source in the Web browser:

```
<label for="passwordField">A password field</label>
<input type="password" name="passwordField" id="passwordField" value="" />
```

A checkbox field — This is a simple toggle; it can be either on or off. The value attribute should contain the value that will be sent to the server when the checkbox is selected (if the checkbox isn't selected, nothing is sent):

```
<label for="checkboxField">A checkbox field</label>
<input type="checkbox" name="checkboxField" id="checkboxField"
value="yes" />
```

You can preselect a checkbox by adding the attribute checked="checked" to the input tag –– for example:

```
 <input type="checkbox" checked="checked" ... />.
```

By creating multiple checkbox fields with the same name attribute, you can allow the user to select multiple values for the same field.

Two radio button fields — Radio buttons tend to be placed into groups of at least two buttons. All buttons in a group have the same name attribute. Only one button can be selected per group. As with checkboxes, use the value attribute to store the value that is sent to the server if the button is selected.

Note that the value attribute is mandatory for checkboxes and radio buttons,
and optional for other field types:

```
<label for="radioButtonField1">A radio button field</label>
<input type="radio" name="radioButtonField" id="radioButtonField1"
value="radio1" />
<label for="radioButtonField2">Another radio button</label>
<input type="radio" name="radioButtonField" id="radioButtonField2"
value="radio2" />
```

You can preselect a radio button using the same technique as for preselecting checkboxes.

A submit button — Clicking this type of button sends the filled-in form to the server-side script for processing. The value attribute stores the text label that is displayed inside the button (this value is also sent to the server when the button is clicked):

```
<label for="submitButton">A submit button</label>
```

<input type=”submit” name=”submitButton” id=”submitButton”
value=”Submit Form” />
A reset button — This type of button resets all form fields back to their initial values (often empty). The value attribute contains the button label text:
<label for=”resetButton”>A reset button</label>
<input type=”reset” name=”resetButton” id=”resetButton”
value=”Reset Form” />
A file select field — This allows the users to choose a file on their hard drive for uploading to the server (see “Creating File Upload Forms” later in the chapter). The value attribute is usually ignored by the browser:
<label for=”fileSelectField”>A file select field</label>
<input type=”file” name=”fileSelectField” id=”fileSelectField” value=”” />
A hidden field — This type of field is not displayed on the page; it simply stores the text value specified in the value attribute. Hidden fields are great for passing additional information from the form to the server, as you see later in the chapter:
<label for=”hiddenField”>A hidden field</label>
<input type=”hidden” name=”hiddenField” id=”hiddenField” value=”” />
An image field — This works like a submit button, but allows you to use your own button graphic instead of the standard gray button. You specify the URL of the button graphic using the src attribute, and the graphic’s width and height (in pixels) with the width and height attributes. As with the submit button, the value attribute contains the value that is sent to the server when the button is clicked:
<label for=”imageField”>An image field</label>
<input type=”image” name=”imageField” id=”imageField”
value=””src=”asterisk.gif” width=”23” height=”23” />
A push button — This type of button doesn’t do anything by default when it’s clicked, but you can make such buttons trigger various events in the browser using JavaScript. The value attribute specifies the text label to display in the button:
<label for=”pushButton”>A push button</label>
<input type=”button” name=”pushButton” id=”pushButton” value=”Click Me” />
A pull-down menu — This allows a user to pick a single item from a predefined list of options. The size attribute’s value of 1 tells the browser that you want the list to be in a pull-down menu format. Within the select element, you create an option element for each of your options. Place the option label between the <option> ... </option> tags. Each option element can have an optional value attribute, which is the value sent to the server if that option is selected. If

you don’t include a value attribute, the text between the <option> ... </option> tags is sent instead:
<label for=”pullDownMenu”>A pull-down menu</label>
<select name=”pullDownMenu” id=”pullDownMenu” size=”1”>

<option value=”option1”>Option 1</option>
<option value=”option2”>Option 2</option>
<option value=”option3”>Option 3</option>
</select>

A list box — This works just like a pull-down menu, except that it displays several options at once. To turn a pull-down menu into a list box, change the size attribute from 1 to the number of options to display at once:

<label for=”listBox”>A list box</label>
<select name=”listBox” id=”listBox” size=”3”>
<option value=”option1”>Option 1</option>
<option value=”option2”>Option 2</option>
<option value=”option3”>Option 3</option>
</select>

A multi-select list box — This works like a list box, but it also allows the user to select multiple items at once by holding down Ctrl (on Windows and Linux browsers) or Command (on Mac browsers). To turn a normal list box into a multi-select box, add the attribute multiple (with a value of “multiple“) to the select element. If the user selects more than one option, all the selected values are sent to the server (you learn how to handle multiple field values later in the chapter):

<label for=”multiListBox”>A multi-select list box</label>
<select name=”multiListBox” id=”multiListBox” size=”3” multiple=”multiple”>
<option value=”option1”>Option 1</option>
<option value=”option2”>Option 2</option>
<option value=”option3”>Option 3</option>
</select>

You can preselect an option in any type of select element by adding the attribute
selected=”selected” to the relevant <option> tag — for example: <option value=”option1” selected=”selected”>.

A text area field — This is similar to a text input field, but it allows the user to enter multiple lines of text. Unlike most other controls, you specify an initial value (if any) by placing the text
between the <textarea> ... </textarea> tags, rather than in a value attribute. A
textarea element must include attributes for the height of the control in rows (rows) and the width of the control in columns (cols):

<label for=”textAreaField”>A text area field</label>
<textarea name=”textAreaField” id=”textAreaField” rows=”4” cols=”50”></textarea>

Once the controls have been added to the form, it’s simply a case of closing the form element:

</form>

Capturing Form Data with PHP

A super global is a built - in PHP variable that is available in any scope: at the top level of your script, within a function, or within a class method.

Superglobal Array	Description
$_GET	Contains a list of all the field names and values sent by a form using the get method
$_POST	Contains a list of all the field names and values sent by a form using the post method
$_REQUEST	Contains the values of both the $_GET and $_POST arrays combined, along with the values of the $_COOKIE superglobal array

Each of these three superglobal arrays contains the field names from the sent form as array keys, with the field values themselves as array values. For example, say you created a form using the get method, and that form contained the following control:
< input type="text " name="emailAddress" value="" / >

You could then access the value that the user entered into that form field using either the $_GET or the $_REQUEST superglobal:

$email = $_GET["emailAddress"];
$email = $_REQUEST["emailAddress"];

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

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

PHP Get Form

Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.

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

File: form1.html

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name"/>
<input type="submit" value="visit"/>
</form>
```

File: welcome.php

```
<?php
```

```
2. $name=$_GET["name"];//receiving name field value in $name variable
3. echo "Welcome, $name";
4. ?>
```

PHP Post Form

Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.

The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

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

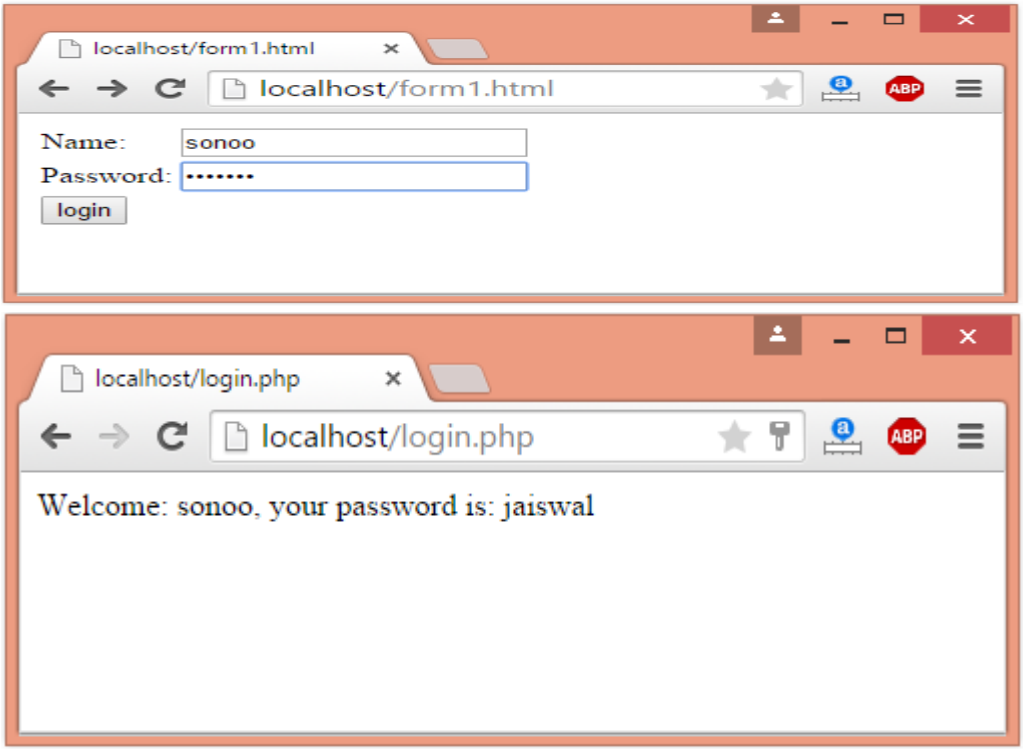
File: form1.html

```
1. <form action="login.php" method="post">
2. <table>
3. <tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
4. <tr><td>Password:</td><td> <input type="password" name="password"/>
   </td></tr>
5. <tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>
6. </table>
7. </form>
```

File: login.php

```
1. <?php
2. $name=$_POST["name"];//receiving name field value in $name variable
3. $password=$_POST["password"];//receiving password field value in $passwor
   d variable
4.
5. echo "Welcome: $name, your password is: $password";
6. ?>
```

Output:



Handling an HTML form with PHP is one of the most important process in a dynamic Web site. Two steps are involved: first you create the HTML form, and then you create the corresponding PHP script that will receive and process the form data.

1) An HTML form is created using the form tags and various elements for taking input.
- Here is an example of an HTML form with one input text box, two radio buttons and a submit button:

```
<!DOCTYPE html>
<html>
<head>
<body>

<form action="script.php" method="post">
 Name: <input type="text" name="user" /><br />
 <input type="radio" name="gen" value="man" />Man
 <input type="radio" name="gen" value="woman" />Woman<br />
 <input type="submit" name="submit" value="Send" />
</form>

</body>
</html>
```

- The **action** attribute dictates to which page the form data will be sent.
- The **method** attribute dictates how the data is sent to the PHP script. It has

two options: post and get (post is the value used most frequently).
- The **name** attribute gives the name of that form field. With this name, the php script can retrieve data from that field.
- The **value** attribute define a value for that form element.
If multiple radio buttons have the same name, only one can be chosen by the user.

The HTML code above will display the following form:

Name:
Man Woman
Send

When a user fills out the form above and click on the submit button (Send), the form data is sent to the "script.php" file.

2) In the "script.php" file we write the PHP script that will receive and process the form data.
- *All data received from an HTML form with method="post" are stored in the superglobal $_POST array.*

```
<?php
if (isset($_POST['submit'])) {
  $user = $_POST['user'];
  $gen = $_POST['gen'];

  echo 'User: '. $user. ' - gender: '. $gen;
}
?>
```

First, the script use the expresion *if (isset($_POST['submit']))* to determine if the form has been submitted (checks if the form element with name="submit" has been received).
The data entered into the form input which has *name="user"*, will be accessible through $_POST['user']. The same applies to the other form elements, their name attribute will automatically be the keys in the $_POST array.
In the PHP script above, the values received from that form are added in *$user* and *$gen* variables, then they are displayed with the "echo" instruction.
The result will be something like this:

User: MarPlo - gender: man

*If you use **method="get"** in the form tag, then, in the PHP script you must use **$_GET** ($_GET['field_name'])*

*or **$_REQUEST** ($_REQUEST['field_name']).*
- With $_REQUEST you can get data from both "post" and "get" methods.

Form Validation

Data validation is necessary to avoid errors and for security.
Validating form data requires the use of conditionals, operators, and specific functions. One standard function to be used is **isset()**, which tests if a variable has a value (but not NULL).
One issue with the isset() function is that an empty string results as TRUE.
Along with the isset() function we can use a construction with the **strlen()** function to check if the string has at least one character.

Let's see a more complete example, with the PHP code and the HTML form in the same php file.
To test yourself this example, add the fallowing code in a php file, than access it in your browser.

```
<html>
<head>
<title>PHP form example</title>
</head>
<body>

<?php
if (isset($_POST['user']) && isset($_POST['age'])) {

    $user = trim($_POST['user']);
    $age = trim($_POST['age']);


  if (strlen($user)>0 && is_numeric($age)) {
    echo $user. ' - '. $age. ' years old.';
  }
  else {
    echo '<h4>Fill the form with valid data</h4>';
  }
}
else {
  echo '<h4>Fill the form</h4>';
}
?>

<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>" method="post">
 User: <input type="text" name="user" /><br />
 Age: <input type="text" name="age" /><br />
 <input type="submit" value="Send" />
```

```
</form>

</body>
</html>
```

- First, the PHP script uses the isset() function to check if the necessary data from the form are sent. Then, if that data are received, adds them in variable, using the *trim()* function to remove white spaces from the beginning and the end of the values.
- If there isn't data received from the form, will be executed the else statement, wich outputs: "*<h4>Fill the form</h4>*".
- To not use empty value, we use "*strlen($user)>0*" to check if the data from 'user' has at least one character.
- Becouse the age must be a number, we use "*is_numeric($age)*" to check if the data from 'age' is a numeric value.
- If the $user has an empty value or the $age isn't a number, will be executed the else statement, wich outputs: "*<h4>Fill the form with valid data</h4>*".

The code: ***<?php echo $_SERVER['SCRIPT_NAME']; ?>*** *outputs the path (the filename) of the current script, often used for the "action" attribute of the form that must send data to the same php file where the form is created.*
- You can also have a form submit back to this same page by using no value for the action attribute:
<form action="" method="post">

PHP File Upload

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

PHP file upload features allow you to upload binary and text files both. Moreover, you can have the full control over the file to be uploaded through PHP authentication and file operation functions.

PHP $_FILES

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

Here, we are assuming that file name is *filename*.

$_FILES['filename']['name']

returns file name.

$_FILES['filename']['type']

returns MIME type of the file.

$_FILES['filename']['size']

returns size of the file (in bytes).

$_FILES['filename']['tmp_name']

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

$_FILES['filename']['error']

returns error code associated with this file.

---

move_uploaded_file() function

The move_uploaded_file() function moves the uploaded file to a new location. The move_uploaded_file() function checks internally if the file is uploaded thorough the POST request. It moves the file if it is uploaded through the POST request.

Syntax

1. bool move_uploaded_file (string $filename , string $destination)

PHP File Upload Example

File: uploadform.html

```
1. <form action="uploader.php" method="post" enctype="multipart/form-data">
2.     Select File:
3.     <input type="file" name="fileToUpload"/>
4.     <input type="submit" value="Upload Image" name="submit"/>
5. </form>
```

File: uploader.php

```
1. <?php
2. $target_path = "e:/";
3. $target_path = $target_path.basename( $_FILES['fileToUpload']['name']);
4.
5. if(move_uploaded_file($_FILES['fileToUpload']['tmp_name'], $target_path)) {
6.     echo "File uploaded successfully!";
7. } else{
8.     echo "Sorry, file not uploaded, please try again!";
9. }
10.       ?>
```

PHP get_browser() Function

‹ PHP Misc Reference

Example

Look up the browscap.ini file and return the capabilities of the browser:

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
$browser = get_browser();
print_r($browser);
?>
```

Definition and Usage

The get_browser() function looks up the user's browscap.ini file and returns the capabilities of the user's browser.

Syntax

get_browser(*user_agent,return_array*)

Parameter Values

Parameter	Description
user_agent	Optional. Specifies the name of an HTTP user agent. Default is the value of $HTTP_USER_AGENT. You can bypass this parameter with NULL
return_array	Optional. If this parameter is set to TRUE, the function will return an array instead of an object

Technical Details

Return Value:	Returns an object or an array with information about the user's browser on success, or FALSE on failure
PHP Version:	4+
Changelog:	The *return_array* parameter was added in PHP 4.3.2