

*PHI*

*Prof. Jayakumar Sadhasivam,  
School of Information Technology and Engineering,  
Vellore Institute of Technology, Vellore.*

# Table of Content

- AMP Package
- Overview of PHP
- PHP - Environment Setup
- PHP with HTML
- PHP Syntax
- Output/Printing in PHP
- PHP Variables
- PHP Data Types
- PHP Strings
- PHP - Constants
- Type Casting
- PHP Operators
- Conditional Statements
- PHP Loops
- PHP Arrays
- PHP Functions
- Date and Time Function
- Global Variables - Superglobals
- File Handling

# AMP Package

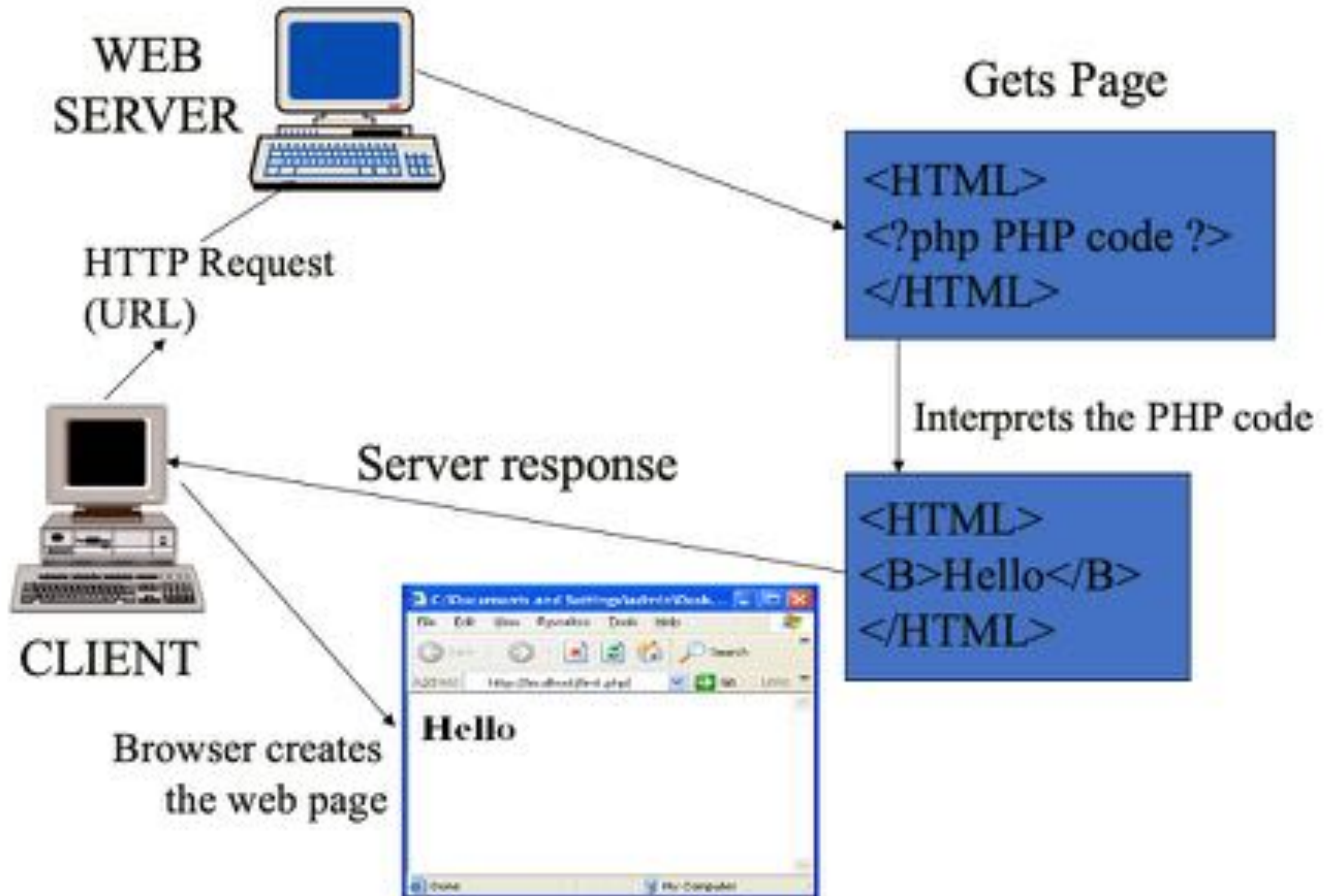
- ◉ AMP is an acronym formed from the initials of Apache, MySQL and PHP or Python or Perl.
  - ✓ Apache: It acts as Web server. Its main job is to parse any file requested by a browser and display the correct results according to the code within that file.
  - ✓ PHP: PHP is a server-side scripting language that allows your Web site to be truly dynamic.
  - ✓ MySQL: It enables PHP and Apache to work together to access and display data in a readable format to a browser.

# Apache Introduction

- ◉ **Apache** is a Web server Responds to client requests by providing resources
- ◉ The first version of Apache, based on the **NCSA**(National Center for Supercomputing Applications) https Web server, was developed in 1995.
- ◉ Because it was developed from existing NCSA code plus various patches, It was called a **patchy server** - hence the name **Apache Server**.
- ◉ As a result of its sophisticated features, excellent performance, and low price - free, Apache has become the world's most popular Web server.

# HTTP Request Types

- ◉ **Get** - The HTTP GET request method is designed to retrieve information from the server.
- ◉ **Post** - The POST request method is designed to request that a web server accepts the data enclosed in the request message's body for storage. It is often used when uploading a file or submitting a completed web form.



# Overview of PHP

◎ PHP - Hypertext PreProcessor.

✓ Other Names :

- Personal Home Page
- Professional Home Page

◎ It was originally created by Danish-Canadian programmer Rasmus Lerdorf in 1994.





# What is PHP?

- ◉ PHP is a widely-used, open source scripting language
- ◉ PHP is free to download and use
- ◉ PHP scripts are executed on the server
  - ✓ Capable of generating the HTML pages
- ◉ HTML generates the web page with the static text and images.



# What Can PHP Do?

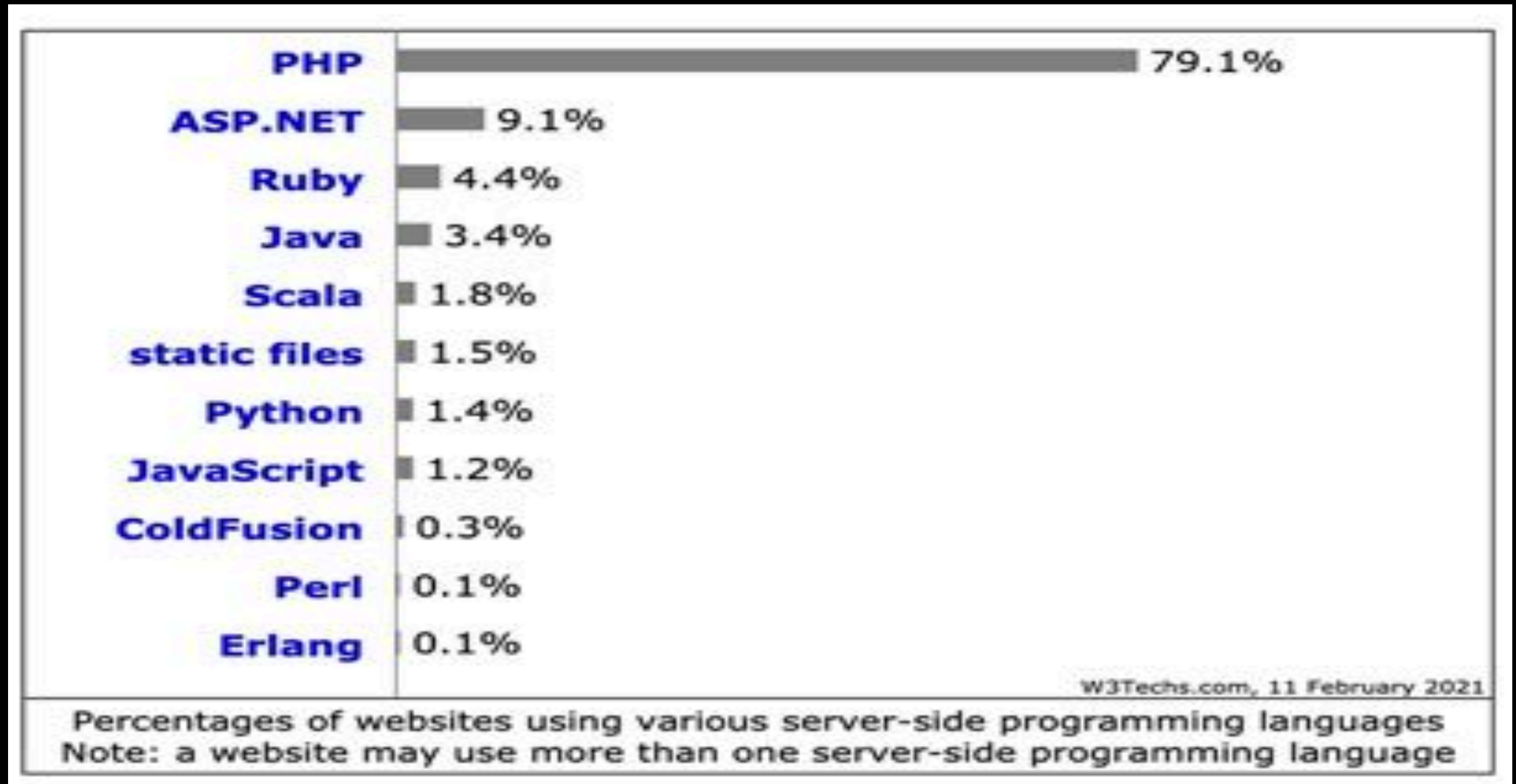
- ◉ PHP can generate dynamic page content
- ◉ PHP can create, open, read, write, delete, and close files on the server
- ◉ PHP can collect data from user
- ◉ PHP can send and receive cookies
- ◉ PHP can add, delete, modify data in your database
- ◉ PHP can be used to control user-access
- ◉ PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

# PHP is an amazing and popular language

- ◉ It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
- ◉ It is deep enough to run the largest social network (Facebook)!
- ◉ Companies using PHP
  - ✓ Wikipedia
  - ✓ Yahoo
  - ✓ Mailchimp
  - ✓ Tumblr(Now switched to PHP7)
  - ✓ Slack (PHP in the backend)
  - ✓ Dailymotion
  - ✓ Etsy

# Usage statistics of server-side programming languages - Feb 2021



[https://w3techs.com/technologies/overview/programming\\_language](https://w3techs.com/technologies/overview/programming_language)

# Why PHP?

- ◉ PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- ◉ PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- ◉ PHP supports a wide range of databases
- ◉ PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net)

# PHP - Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- ◉ **Web Server** – PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server.
- ◉ **Database** – PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- ◉ **PHP Parser** – In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser.

# Setup Web Server on Your Own PC

- ◉ Install a web server
- ◉ Install PHP
- ◉ Install a database, such as MySQL
- ◉ XAMPP (**X** ("cross"-platform), **A**pache HTTP Server, **M**ySQL, **P**HP and **P**erl)
  - ✓ <https://www.apachefriends.org/download.html>
  - ✓ <https://www.wikihow.com/Install-XAMPP-for-Windows>

# Setup Web Server on Your Own PC

- ◉ WAMP for Windows

- <https://www.wampserver.com>

- ◉ LAMP for Linux

- <https://bitnami.com/stack/lamp>

- ◉ MAMP for MacOS

- <https://www.mamp.info>

- ◉ Wamp vs Lamp vs Mamp vs Xampp

- <https://www.websoftwo.com/difference-wamp-lamp-mamp-xampp/>



# PHP with HTML

- ◉ PHP programs are written using a text editor, such as Notepad or WordPad, just like HTML pages.
- ◉ PHP pages end with **.PHP** file extension. This extension signifies to the server that it needs to parse the PHP code before sending the resulting HTML code to the viewer's Web browser.
- ◉ What makes PHP so different is that it not only allows HTML pages to be created.
- ◉ It is invisible to your Web site visitors.
- ◉ The only thing they see the resulting HTML output.
- ◉ This gives you more security for your PHP code and more flexibility in writing it.
- ◉ HTML can also be written inside the PHP section of your page.
- ◉ PHP can also be written as a standalone program, without HTML

# PHP Syntax

- PHP is denoted in the page with opening and closing tags as follows:

`<?php`                       $\longrightarrow$  Opening Tag

`//php code;`

`?>`                       $\longleftarrow$  Closing Tag

- Short-open (SGML-style) tags
  - ✓ Short or short-open tags look like this

`<?...?>`

# Output/Printing in PHP

- ◉ **echo** & **print()** is the common method in outputting data.
- ◉ Since it is a language construct, echo doesn't require parenthesis like print().

```
<?php  
    echo "Hello World";  
    print("Hello World");  
?>
```

Output:

```
Hello World  
Hello World
```

# Comments in PHP

```
<?php
```

```
// This is a single-line comment
```

```
# This is also a single-line comment
```

```
?>
```

```
<?php
```

```
/*
```

```
This is a multiple-lines comment block  
that spans over multiple  
lines
```

```
*/
```

```
?>
```

# Example HTML and PHP

## HTML

```
<html>
<head>
<title>HTML Page View
  </title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
```

## PHP

```
<html>
<head>
<title>Hello World PHP
  Script</title>
</head>
<body>

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

</body>
</html>
```

Output for both HTML and PHP is "Hello World!"

# Web Browser Source Page View of HTML and PHP

```
1 <html>
2 <head>
3 <title>HTML Page View</title>
4 </head>
5 <body>
6 <p>Hello World!</p>
7 </body>
8 </html>
```

```
1 <html>
2 <head>
3 <title>PHP Page View</title>
4 </head>
5 <body>
6
7 Hello World!
8 </body>
9 </html>
10
```

# Printing Statements

<?

```
echo 123;
```

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

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

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

```
echo 'Hello World!';
```

```
echo Hello World!;
```

?>

## Output

123

Hello World!

HelloWorld!

Hello World!

Hello World!

Not valid



# PHP Variables

- ◉ Variables are "containers" for storing information.
- ◉ 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 characters and underscores (A-z, 0-9, and \_)**
  - ✓ Variable names are **case-sensitive**
    - **\$age** and **\$AGE** are two different variables)

# Variables

## VALID

\$prod\_desc

\$Intvar

\$\_Salesamt

## INVALID

\$90ctSales

Sales123

\$\*asgs

# PHP Facts

- PHP is a Loosely Typed Language
  - ✓ PHP automatically associates a data type to the variable, depending on its value.
  - ✓ Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

# PHP Data Types

# PHP Data Types

- ◉ Variables can store data of different types, and different data types can do different things.
- ◉ PHP supports the following data types:
  - ✓ **Integers** – are whole numbers, without a decimal point, like 4195
  - ✓ **Doubles** – are floating-point numbers, like 3.14159 or 49.1.
  - ✓ **Booleans** – have only two possible values either true or false.
  - ✓ **NULL** – is a special type that only has one value: NULL.
  - ✓ **Strings** – are sequences of characters, like 'PHP supports string operations.'
  - ✓ **Arrays** – are named and indexed collections of other values.
  - ✓ **Objects** – are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
  - ✓ **Resources** – are special variables that hold references to resources external to PHP (such as database connections).

# PHP Integer

- ◉ An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
- ◉ An integer can be either positive or negative.

```
<?php
$int_var = 12345;
$another_int = -123456 + 12345;
echo $int_var;
echo $another_int;
?>
```

12345  
-111111

# PHP Float

- ◉ A float (floating point number) is a number with a decimal point or a number in exponential form.

```
<?php
$many = 2.28888002;
$many_2 = 2.2111200;
$few = $many + $many_2;
echo "$many + $many_2 = $few";
?>
```

2.28888002 + 2.21112 = 4.500000002



# PHP Boolean

- ◉ A Boolean represents two possible states: TRUE or FALSE.

```
<?php  
$x = true;  
$y = false;  
?>
```

# PHP NULL

- ◉ Null is a special data type which can have only one value: NULL.
- ◉ A variable of data type NULL is a variable that has no value assigned to it.
- ◉ If a variable is created without a value, it is automatically assigned a value of NULL.
- ◉ Variables can also be emptied by setting the value to NULL:

```
<?
```

```
$my_var = NULL;  
$mmy_var = null;
```

```
//NULL is case insensitive;
```

```
?>
```

# PHP Array

- ◉ An array stores multiple values in one single variable.

<?

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

```
echo $cars[0];
```

```
echo $cars[1];
```

```
echo $cars[2];
```

?>

# PHP Strings

# PHP Strings

- ◉ A string is a sequence of characters, like "Hello world!".

```
<?php
```

```
$string_1 = "This is a string in double quotes";
```

```
$string_2 = 'This is a somewhat longer, singly  
quoted string';
```

```
$string_39 = "This string has thirty-nine characters";
```

```
$string_0 = ""; // a string with zero characters
```

```
echo $string_1;
```

```
echo $string_2;
```

```
echo $string_39;
```

```
echo $string_0;
```

```
?>
```

Output:

This is a string in double quotes

This is a somewhat longer, singly  
quoted string

This string has thirty-nine characters

# Quotes (" " Vs. ' ')

- ◉ In a double-quoted string any variable names are expanded to their values.
- ◉ In a single-quoted string, no variable expansion takes place.

## Single Quote

```
<?php
$name = "Jay";
$age = 23;
echo '$name is $age';
?>
```

---

Output:

\$name is \$age

## Double Quote

```
<?php
$name = "Jay";
$age = 23;
echo "$name is $age";
?>
```

---

Output:

Jay is 23

# String Escape Sequence

- `\n` is replaced by the newline character
- `\r` is replaced by the carriage-return character
- `\t` is replaced by the tab character
- `\$` is replaced by the dollar sign itself (\$)
- `\"` is replaced by a single double-quote (")
- `\\` is replaced by a single backslash (\)



# String Functions

- `strlen()` - Return the Length of a String

```
<?php
echo strlen("Hello world!");
// outputs 12
?>
```

- `str_word_count()` - Count Words in a String

```
<?php
echo str_word_count("Hello world!");
// outputs 2
?>
```

# String Functions

## ● strpos() - Search For a Text Within a String

- ✓ The PHP strpos() function searches for a specific text within a string. If a **match** is found, the function returns the **Character Position** of the **first match**. If **no match** is found, it will return **FALSE**.

```
<?php
echo strpos("Hello world!", "world");
           // outputs 6
?>
```

# String Functions

## ● `str_replace()` - Replace Text Within a String

- ✓ `str_replace()` function replaces some characters with some other characters in a string.

```
<?php
```

```
$para = "Its PHP world!!!, welcome to Hello world !!";
```

```
$newPara = str_replace("world", "Server", $para);
```

```
echo $newPara;
```

```
?>
```

O/P - Its PHP **Server!!!**, welcome to Hello **Server !!**

# PHP - Constants

# PHP - Constants Types

- ◉ 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**)

## Parameters:

- ◉ **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**



Optional

# Constant - Without using Case-insensitive

```
1 <?php
2 define("GREETING", "Welcome PHP C0ursse \n", false);
3 echo GREETING;
4 //using case sensitive while calling the function
5 ?>
6
7 <?php
8 define("GREETING", "Welcome PHP C0ursse \n");
9 echo greeting;
10 //NOT using case sensitive while calling the function
11 ?>
```

Welcome PHP C0ursse

Warning: Use of undefined constant greeting – assumed 'greeting' (that will be an Error in a future version of PHP) in /private/var/folders/bw/46g1\_xf961bgfrfbb6z5x5rr0000gn/T/CodeRunner/Untitled.php on line 9  
greeting

# Constant - Using Case-insensitive

```
1 <?php
2 define("GREETING", "Welcome PHP C0ursse \n",true);
3 echo GREETING; //using case sensitive
4 ?>
5
6 <?php
7 define("GREETING", "Welcome PHP C0ursse \n",true);
8 echo greeting; //using Case-insensitive
9 ?>
```

Welcome PHP C0ursse

Welcome PHP C0ursse





# PHP Constants

- Another way of calling the PHP Constants

```
1
2 <?php
3 //define a constant
4 define("HELLO", "Welcome to PHP");
5 echo constant("HELLO");
6 ?>
```

Welcome to PHP



# Type Casting

# Type Casting

- ◉ Forcing a variable to behave as a type other than the one originally intended for it is known as type casting.
- ◉ A variable can be evaluated once as a different type by casting it to another.
- ◉ Types
  - ✓ **settype()**
  - ✓ **gettype()**

# Type Casting - settype()

- ◉ The `settype()` function converts a variable to a specific type.

- ◉ Syntax:

```
settype(VariableName, "newDataType");
```

```
<?php
    $pi = 3.14;
    settype($pi, "string");
    echo $pi;
    settype($pi, "integer");
    echo $pi;
?>
```

Output:

3.14

3

# Type Casting - gettype()

- The `gettype()` function returns the type of a variable.

- Syntax:

`gettype(VariableName);`

```
<?php
```

```
$a = 3;          echo gettype($a);  
$b = 3.2;        echo gettype($b);  
$c = "Hello";    echo gettype($c);  
$d = array();    echo gettype($d);  
$e = array("red", "green", "blue");  
echo gettype($e);  
$f = NULL;       echo gettype($f);  
$g = false;      echo gettype($g);  
?>
```

Output:  
integer  
double  
string  
array  
array  
NULL  
boolean

# PHP Operators

# PHP Operators

- ◉ Operators are used to perform operations on variables and values.
- ◉ PHP language supports following type of operators.
  - ✓ 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.
- ◉ Assume variable A holds 10 and variable B holds 20.

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

# Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into $C$
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C \% = A$ is equivalent to $C = C \% A$



# Comparison Operators

- ◉ Assume variable A holds 10 and variable B holds 20.

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

# Increment/Decrement operators

- ◉ Increment operators are used to increment a variable's value.
- ◉ Decrement operators are used to decrement a variable's value.

If  $x = 10$

Operator	Name	Description	Result $x$
$++x$	Pre-increment	Increments $x$ by one, then returns $x$	<b>11</b>
$x++$	Post-increment	Returns $x$ , then increments $x$ by one	<b>10</b>
$--x$	Pre-decrement	Decrements $x$ by one, then returns $x$	<b>9</b>
$x--$	Post-decrement	Returns $x$ , then decrements $x$ by one	<b>10</b>

# Logical Operators

Operator	Name	Description	Result
<b>and</b>	<b>And</b>	\$x and \$y	True if both \$x and \$y are true
<b>or</b>	<b>Or</b>	\$x or \$y	True if either \$x or \$y is true
<b>xor</b>	<b>Xor</b>	\$x xor \$y	True if either \$x or \$y is true, but not both
<b>&amp;&amp;</b>	<b>And</b>	\$x && \$y	True if both \$x and \$y are true
<b>  </b>	<b>Or</b>	\$x    \$y	True if either \$x or \$y is true
<b>!</b>	<b>Not</b>	!\$x	True if \$x is not true

# String Operators

Operator	Name	Description	Result
<b>.</b> ( <b>dot</b> )	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
<b>.=</b>	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```

Output:  
Hello world!

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
$txt1 .= $txt2;
echo $txt1;
?>
```

Output:  
Hello world!

# Array Operators

Operator	Name	Example	Result
+	Union	<code>\$x + \$y</code>	Union of <code>\$x</code> and <code>\$y</code>
<code>==</code>	Equality	<code>\$x == \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs
<code>===</code>	Identity	<code>\$x === \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs in the same order and of the same types
<code>!=</code>	Inequality	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>&lt;&gt;</code>	Inequality	<code>\$x &lt;&gt; \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>!==</code>	Non-identity	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not identical to <code>\$y</code>

# Conditional Assignment Operators

Operator	Name	Description	Result
<b>?:</b>	Ternary	Conditional Expression	<p>If Condition is true ? Then value X : Otherwise value Y</p> <p><b>\$x = expr1 ? expr2 : expr3</b> Returns the value of \$x. The value of \$x is expr2 if expr1 = TRUE. The value of \$x is expr3 if expr1 = FALSE</p>

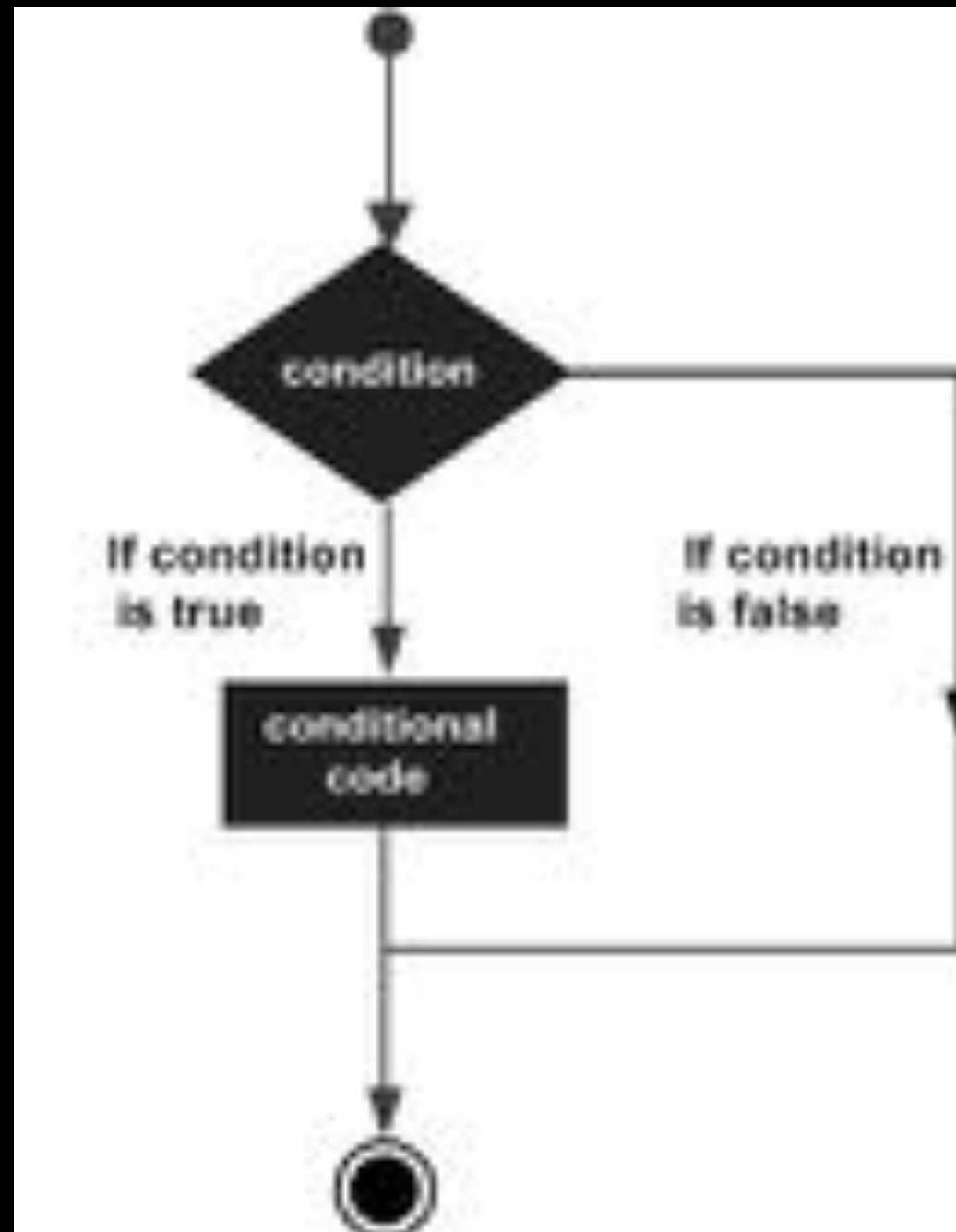
# Conditional Statements

# Conditional Statements

- ◉ Conditional statements are used to perform different actions based on different conditions.
  - ✓ **if statement** - executes some code if one condition is true
  - ✓ **if...else statement** - executes some code if a condition is true and another code if that condition is false
  - ✓ **if...elseif...else statement** - executes different codes for more than two conditions
  - ✓ **switch statement** - selects one of many blocks of code to be executed



# Conditional Statements



# The if Statement

- The if statement executes some code if one condition is true.

## Syntax

```
<?
if (condition) {
    code to be executed if condition is true;
}
?>
```

```
<?php
$a=30;  $b=20;
if ($a > $b)
{
    echo "a is bigger than b";
}
?>
```

Output: a is bigger than b

# The if...else Statement

- ◉ The if...else statement executes some code if a condition is true and another code if that condition is false.
- ◉ Syntax

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

# The if...else Statement

```
<?php
$a=10;
if ($a > 100) {
    echo "A is greater";
} else {
    echo "A is smaller";
}
?>
```

Output: A is smaller

# The if...elseif...else Statement

- ◉ The if...elseif statement to select one of several blocks of code to be executed.

- ◉ Syntax

```
if (condition) {  
    code to be executed if this condition is true;  
} elseif (condition) {  
    code to be executed if first condition is false  
    and this condition is true;  
} else {  
    code to be executed if all conditions are  
    false;  
}
```

# The if...elseif...else Statement

```
<?php
$Weather="SUN";
if($Weather=="Sunny"){
    echo "Weather is pleasant";
} elseif ($Weather=="Rainy") {
    echo "its raining";
} elseif($Weather=="Cloudy") {
    echo "Cloud pleasant";
} else{
    echo "Its hard to forecast weather";
} ?>
```

Output: Its hard to forecast weather

# Switch Statement

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

## SYNTAX:

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

# Switch Statement

```
<?php
$Weather="Cloudyblurr";
switch($Weather){
    case "Sunny":
        echo "Weather is quite pleasant outside";
        break;
    case "Rainy":
        echo "Its Raining outside";
        break;
    case "Cloudy":
        echo "It is expected to Rain";
        break;
    default:
        echo "Weather Can not be forecast";
        break;
}
?>
```

Output:

Weather Can not be forecast



# Switch Statement

```
<?php
$favcolor = "red";
switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red,
            blue, nor green!";
}
?>
```

Output:

Your favorite color is red!

# PHP Loops

# PHP Loops

- ◉ Loops are used to execute the same block of code again and again, as long as a certain condition is true.
- ◉ In PHP, we have the following loop types:
  - ✓ **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;  
}
```

# while Loop

```
<?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*

# while Loop

```
<?php
$x = 0;
while($x <= 100) {
    echo "The number is: $x <br>";
    $x+=10;
}
?>
```

## Output:

```
The number is: 0
The number is: 10
The number is: 20
The number is: 30
The number is: 40
The number is: 50
The number is: 60
The number is: 70
The number is: 80
The number is: 90
The number is: 100
```

# 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);
```

# do...while Loop

```
<?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*



# do...while Loop

```
<?php
$x = 6;

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

*Output:*  
*The number is: 6*

# for Loop

- ◉ The for 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 for each iteration;
}
```

## 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

# for Loop

```
<?php
for ($x = 0; $x <= 5; $x++) {
    echo "The number is: $x <br>";
}
?>
```

*Output:*

*The number is: 0*

*The number is: 1*

*The number is: 2*

*The number is: 3*

*The number is: 4*

*The number is: 5*

# for Loop

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

*Output:*

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

# for Loop

```
<?php
for ($x = 0; $x <= 50; $x+=10) {
    echo "The number is: $x <br>";
}
?>
```

Output:

```
The number is: 0
The number is: 10
The number is: 20
The number is: 30
The number is: 40
The number is: 50
```

# 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;  
}
```

# foreach Loop

```
<?php
$colors = array("red", "green", "blue",
               "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

Output:

red  
green  
blue  
yellow

# foreach Loop

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

foreach($age as $x => $val) {
    echo "$x = $val<br>";
}
?>
```

Output:

Peter = 35

Ben = 37

Joe = 43



# Break and Continue

- ◉ The **break** statement can also be used to jump out of a loop.
- ◉ The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

# Break Statement

```
<?php
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        //Break the statement, when it reaches X=4.
        break;
    }
    echo "The number is: $x <br>";
}
?>
```

*Output:*

*The number is: 0*  
*The number is: 1*  
*The number is: 2*  
*The number is: 3*

# Continue Statement

```
<?php
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        continue; //when it reaches the value x=4,
                  //it skip that value and continue to next iteration
    }
    echo "The number is: $x <br>";
}
?>
```

*Output:*

*The number is: 0  
The number is: 1  
The number is: 2  
The number is: 3  
The number is: 5  
The number is: 6  
The number is: 7  
The number is: 8  
The number is: 9*

# Break in While Loop

```
<?php
$x = 0;

while($x < 10) {
    if ($x == 4) {
        break;
    }
    echo "The number is: $x <br>";
    $x++;
}
?>
```

*Output:*

*The number is: 0*  
*The number is: 1*  
*The number is: 2*  
*The number is: 3*

# Continue in While Loop

```
<?php
$x = 0;
while($x < 10) {
    if ($x == 4) {
        $x++;
        continue;
    }
    echo "The number is: $x";
    $x++;
}
?>
```

## Output:

The number is: 0  
The number is: 1  
The number is: 2  
The number is: 3  
The number is: 5  
The number is: 6  
The number is: 7  
The number is: 8  
The number is: 9

# Arrays

# Arrays

- ◉ An array stores multiple values in one single variable.
- ◉ In PHP, there are three types of arrays:
  - ✓ **Indexed/Numeric arrays** - Arrays with a numeric index
  - ✓ **Associative arrays** - Arrays with named keys
  - ✓ **Multidimensional arrays** - Arrays containing one or more arrays

# Length of an Array - The count() Function

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

Output: 3



# Indexed/Numeric Arrays

- ◉ A numeric array stores each array element with a numeric index.
- ◉ There are two ways to create indexed arrays:
  - ✓ The index can be assigned automatically (index always starts at 0)

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

- ✓ The index can be assigned manually

```
$cars[10] = "Volvo";  
$cars[11] = "BMW";  
$cars[12] = "Toyota";
```

# Indexed/Numeric Arrays

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

echo "I like " . $cars[0] . ", " .
      $cars[1] . " and " . $cars[2] . ".";

?>
```

Output:

I like Volvo, BMW and Toyota.

# Loop Through an Numeric Array

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

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

Output:

```
Volvo
BMW
Toyota
```

# Associative Arrays

- ◉ An associative array, each ID key is associated with a value.
- ◉ When storing data about specific named values, a numerical array is not always the best way to do it.
- ◉ With associative arrays we can use the values as keys and assign values to them

# Associative Arrays

- 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";
```

# Associative Arrays

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

Output:

Peter is 35 years old.

# Associative Arrays

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

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

?>
```

Output:

```
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43
```

# Multidimensional Arrays

- ◉ In a multidimensional array, each element in the main array can also be an array.
- ◉ Each element in the sub-array can be an array, and so on.



# Multidimensional Arrays

```
<?php
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2]."\n";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2]."\n";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2]."\n";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2]."\n";
?>
```

```
Volvo: In stock: 22, sold: 18
BMW: In stock: 15, sold: 13
Saab: In stock: 5, sold: 2
Land Rover: In stock: 17, sold: 15
```

# PHP Sorting Arrays

- ◉ The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.
- `sort()` - sort arrays in ascending order
- `rsort()` - sort arrays in descending order
- `asort()` - sort associative arrays in ascending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `arsort()` - sort associative arrays in descending order, according to the value
- `krsort()` - sort associative arrays in descending order, according to the key

# PHP Arrays Functions

- ◉ `is_array($arr)` - To check whether the provided data is in form of an array, we can use the `is_array()` function. It returns `True` if the variable is an array and returns `False` otherwise.
- ◉ `in_array($var, $arr)` - When using an array, we may often want to check whether a certain value is present in the array or not.
- ◉ `sizeof($arr)` - This function returns the size of the array or the number of data elements stored in the array. It is just like `count($arr)` method
- ◉ `array_slice($arr, $offset, $length)` - This function is used to create a subset of any array. Using this function, we define the starting point(`$offset`, which is the array index from where the subset starts) length(or, the number of elements required in the subset, starting from the offset).

# PHP Arrays Functions

- ◉ `array_pop($arr)` - This function removes the last element of the array.
- ◉ `array_shift($arr)` - used to remove/shift the first element out of the array.
- ◉ `array_push($arr, $val)` - This can be used to add a new element at the end of the array.
- ◉ `array_reverse($arr)` - This function is used to reverse the order of elements
- ◉ `array_merge($arr1, $arr2)` - If you want to combine two different arrays into a single array.
- ◉ `array_rand($arr)` - If you want to pick random data element from an array

# PHP Functions

# PHP Functions

## ● PHP Built-in Functions

- ✓ PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.
  - \* Array functions
  - \* Calendar functions
  - \* Date functions
  - \* Directory functions
  - \* Error functions
  - \* Mail functions
  - \* Math functions
  - \* Misc functions
  - \* MySQL functions

# PHP Functions

## ● PHP User Defined Functions

- Besides the built-in PHP functions, it is possible to create your own functions.
  - ✓ A function is a **block of statements** that can be used **repeatedly** in a program.
  - ✓ A function will **not execute automatically** when a **page loads**.
  - ✓ A function will be **executed by a call** to the function.

# User Defined Function in PHP

- ◉ A user-defined function declaration starts with the word **function**
- ◉ A function name must start with a letter or an underscore.  
Function names are NOT case-sensitive.

- ◉ Syntax:

```
function functionName()  
{  
    code to be executed;  
}
```



# User Defined Function in PHP

```
<?php  
function apache() {  
    echo "PHP and MySQL";  
}  
apache();  
?>
```

output:  
PHP and MySQL

# User Defined Function in PHP

```
<?php
function apache() {
    echo "PHP and MySQL";
}
echo "Calling Apache First Time : ";
apache();           //we can call the function "N" number of times
echo "\n\nCalling Apache Second Time : ";
apache();
?>
```

output:

```
Calling Apache First Time : PHP and MySQL
Calling Apache Second Time : PHP and MySQL
```

# User Defined Function in PHP

```
<?php
function addnum() {           //creating the function
    $a=10;
    $b=20;
    echo $c=$a+$b;
}
addnum();                     //calling the function
?>
```

output:  
30

# PHP Function Arguments

- ◉ PHP function – Adding Parameters
- ◉ Information can be passed to functions through arguments.
- ◉ An argument is just like a variable.
- ◉ Arguments are specified after the function name, inside the parentheses.
- ◉ You can add as many arguments as you want, just separate them with a comma.

`$fname` is an Argument

```
function familyName($fname) {  
    code to be executed;  
}
```

# PHP Function Arguments

```
<?php
function MarvelMovies($namelist) {
    echo "The Best Marvel Movies – $namelist \n";
}
MarvelMovies("Iron Man");
MarvelMovies("The Incredible Hulk");
MarvelMovies("The Avengers");
MarvelMovies("Doctor Strange");
MarvelMovies("Avengers: Infinity War");
?>
```

output:

```
The Best Marvel Movies – Iron Man
The Best Marvel Movies – The Incredible Hulk
The Best Marvel Movies – The Avengers
The Best Marvel Movies – Doctor Strange
The Best Marvel Movies – Avengers: Infinity War
```

# PHP Function Arguments

```
<?php
function DCMovies($namelist,$year)
{
    echo "$namelist Movie released in $year\n";
}
DCMovies("Batman Begins","2005");
DCMovies("The Dark Knight","2008");
DCMovies("The Dark Knight Rises","2012");
?>
```

output:

```
Batman Begins Movie released in 2005
The Dark Knight Movie released in 2008
The Dark Knight Rises Movie released in 2012
```

# PHP Default Argument Value

```
<?php
function setHeight($minheight = 50)
{
    echo "The height is : $minheight";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

output:

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

# PHP Functions - Returning values

```
<?php
function sum(int $x, int $y)
{
    $z = $x + $y;
    return $z;
}
echo "5 + 10 = " . sum(5,10);
echo "7 + 13 = " . sum(7,13);
echo "2 + 4 = " . sum(2,4);
?>
```

output:

```
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6
```



# PHP Date and Time Functions

# PHP DATE and Time Functions

- ◉ With PHP's date function you format timestamps, so they are more human readable.
- ◉ A timestamp is the number of seconds from January 1, 1970 at 00:00.
- ◉ Otherwise known as the Unix Timestamp, this measurement is a widely used standard that PHP has chosen to utilize.
- ◉ The date function uses letters of the alphabet to represent various parts of a typical date and time format.

# TIME Function

- ◉ a (lowercase A): am or pm depending on the time.
- ◉ A (uppercase A): AM or PM depending on the time.
- ◉ g (lowercase G): Hour without leading zeroes. Values are 1 through 12.
- ◉ G (Uppercase G): Hour in 24-hour format without leading zeroes. Values are 0 through 23.
- ◉ h (lowercase H): Hour with leading zeroes. Values 01 through 12.
- ◉ H (uppercase H): Hour in 24-hour format with leading zeroes. Values 00 through 23.
- ◉ i (lowercase I): Minute with leading zeroes. Values 00 through 59.
- ◉ s (lowecase S): Seconds with leading zeroes. Values 00 through 59.

# DATE Functions

- ◉ **d (lowercase D)**: Day of the month with leading zeroes. Values are 01 through 31.
- ◉ **j (lowercase J)**: Day of the month without leading zeroes. Values 1 through 31
- ◉ **D (uppercase D)**: Day of the week abbreviations. Sun through Sat
- ◉ **l (lowercase L)**: Day of the week. Values Sunday through Saturday
- ◉ **w (lowercase W)**: Day of the week without leading zeroes. Values 0 through 6.
- ◉ **z (lowercase Z)**: Day of the year without leading zeroes. Values 0 through 365.

# MONTH Function

- ◉ **m (lowercase M):** Month number with leading zeroes. Values 01 through 12
- ◉ **n (lowercase N):** Month number without leading zeroes. Values 1 through 12
- ◉ **M (uppercase M):** Abbreviation for the month. Values Jan through Dec
- ◉ **F (uppercase F):** Normal month representation. Values January through December.
- ◉ **t (lowercase T):** The number of days in the month. Values 28 through 31.

# YEAR Function

- ◉ L (uppercase L): 1 if it's a leap year and 0 if it isn't.
- ◉ Y (uppercase Y): A four digit year format. Values 2020
- ◉ y (lowercase Y): A two digit year format. Values 00 through 99.

# Other Functions of Date and Time

- ◉ **e (lowercase E)**: Timezone identifier Examples: UTC, GMT, Atlantic/Azores
- ◉ **S (uppercase S)**: English ordinal suffix for the day of the month, 2 character. Values st, nd, rd or th. Works well with j

# Full Date and Time Function

- ◉ **r (lowercase R)**: Displays the full date, time and timezone offset. It is equivalent to manually entering `date("D, d M Y H:i:s O")`. Values: Example: Thu, 21 Dec 2000 16:01:07 +0200
- ◉ **c (lowercase C)**: ISO 8601 date. Values 2011-02-12T15:19:21+00:00



# Example

```
<?php  
echo date("m/d/y");  
?>
```

Output: 07/15/20

```
<?php  
echo date('l \t\h\e jS');  
?>
```

Output: Wednesday the 15th

```
<?php
```

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

```
echo "Today is " . date("Y.m.d") . "<br>;
```

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

```
echo "Today is " . date("l") . "<br>;
```

```
echo date("l jS \of F Y h:i:s A") . "<br>;
```

```
?>
```

• Today is 2021/03/17

• Today is 2021.03.17

• Today is 2021-03-17

• Today is Wednesday

• Wednesday 17th of March 2021 07:08:30 AM

# Example

```
<?php
date_default_timezone_set("Asia/Kolkata");
echo "The time is " . date("h:i:sa");

echo "<br><br>";

date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

The time is 12:44:07pm

The time is 03:14:07am

# Check Date

● Syntax:

```
checkdate(month, day, year)
```

```
<?php  
var_dump(checkdate(07,15,2020));  
?>
```

Output: bool(true)

# Example

```
<?php  
echo date("m/d/y");  
echo "<br>";  
echo "Today is ";  
echo date("j F Y, \a\\t G.ia");  
?>
```

Output:

07/15/20

Today is 15 July 2020, at 8.45am

# DISPLAYING NEXT WEEK

```
<?php
$nextWeek = time() + (7 * 24 * 60 * 60);
// 7 days; 24 hours; 60 mins; 60secs
echo 'Now: ' . date('Y-m-d') . "\n";
echo 'Next Week: ' . date('Y-m-d', $nextWeek) . "\n";
?>
```

Output:

Now: 2020-07-15

Next Week: 2020-07-22

# mktime() Function

- ◉ Return the Unix timestamp for a date. Then use it to find the day of that date:
- ◉ Syntax
  - `mktime(hour, minute, second, month, day, year, is_dst)`
  - All the functions are optional
  - DST - Daylight Savings Time

```
<?php
```

```
// Prints: March 17th, 2021 was on a Wednesday
```

```
echo "March 17th, 2021 was on a " . date("l",  
    mktime(0, 0, 0, 3, 17, 2021)) .  
    "<br><br>";
```

```
//The mktime() function is useful for doing date arithmetic and validation.  
//It will automatically calculate the correct value for out-of-range input:
```

```
echo date("M-d-Y", mktime(0, 0, 0, 10, 36, 2022)) . "<br>";  
echo date("M-d-Y", mktime(0, 0, 0, 14, 1, 2021)) . "<br>";  
echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 2021)) . "<br>";  
echo date("M-d-Y", mktime(0, 0, 0, 1, 1, 2099)) . "<br>";  
?>
```

March 17th, 2021 was on a Wednesday

Nov-05-2022

Feb-01-2022

Jan-01-2021

Jan-01-2099



# Predefined Date Functions

- ◉ **DATE\_ATOM** - Atom (example: 2021-04-12T15:52:01+00:00)
- ◉ **DATE\_COOKIE** - HTTP Cookies (example: Friday, 12-Apr-20 15:52:01 UTC)
- ◉ **DATE\_ISO8601** - ISO-8601 (example: 2020-04-12T15:52:01+0000)
- ◉ **DATE\_RFC822** - RFC 822 (example: Fri, 12 Apr 20 15:52:01 +0000)
- ◉ **DATE\_RFC850** - RFC 850 (example: Friday, 12-Apr-20 15:52:01 UTC)
- ◉ **DATE\_RFC1036** - RFC 1036 (example: Fri, 12 Apr 20 15:52:01 +0000)
- ◉ **DATE\_RFC1123** - RFC 1123 (example: Fri, 12 Apr 2020 15:52:01 +0000)
- ◉ **DATE\_RFC2822** - RFC 2822 (Fri, 12 Apr 2020 15:52:01 +0000)
- ◉ **DATE\_RFC3339** - Same as DATE\_ATOM
- ◉ **DATE\_RSS** - RSS (Fri, 12 Aug 2020 15:52:01 +0000)
- ◉ **DATE\_W3C** - World Wide Web Consortium (example: 2020-04-12T15:52:01+00:00)

# Predefined Date Functions

```
<?php
    echo date(DATE_RFC822) . "<br>";
    echo date(DATE_ATOM);
?>
```

## Output:

Wed, 17 Mar 21 06:43:40 +0000

2021-03-17T06:43:40+00:00

# PHP Global Variables - Superglobals

# PHP Global Variables - Superglobals

- ◉ PHP offers a number of useful predefined variables.
- ◉ Accessible from anywhere within the executing script and provide you with a substantial amount of environment-specific information
- ◉ The PHP superglobal variables are:
  - ✓ `$GLOBALS`
  - ✓ `$_SERVER`
  - ✓ `$_REQUEST`
  - ✓ `$_POST`
  - ✓ `$_GET`
  - ✓ `$_FILES`
  - ✓ `$_ENV`
  - ✓ `$_COOKIE`
  - ✓ `$_SESSION`

# PHP \$GLOBALS

- ◉ \$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script.
- ◉ PHP stores all global variables in an array called \$GLOBALS[index]. The index holds the name of the variable.

```
<?php
```

```
$x = 75;
```

```
$y = 25;
```

```
function addition() {
```

```
$GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
```

```
}
```

```
addition();
```

```
echo $z;
```

```
?>
```

Output: 100

```

<?php
$MyNumber=456456; // Local Scope
echo "Number is : $MyNumber \n";

function Addition(){
    global $MyNumber; //Assigning $MyNumber as Global
    Variable
    $a=50;
    $b=7;
    $c=$a * $b;
    echo "New Global Addition : " . ($MyNumber * $c);
    echo "\nAddition is {$c}";
}
Addition();
?>

```

Output:

```

Number is : 456456
New Global Addition : 159759600
Addition is 350

```

# PHP \$\_SERVER

- ◉ \$\_SERVER is a PHP super global variable which holds information about headers, paths, and script locations created by the Web server.
- ◉ Information regarding the server and client configuration and the current request environment.
- ◉ `$_SERVER['PHP_SELF']`: Returns the filename of the currently executing script
- ◉ `$_SERVER['HTTP_REFERER']`: The URL of the page that referred the user to the current location.
- ◉ `$_SERVER['REQUEST_URI']`: The path component of the URL. For example, if the URL is `http://www.example.com/blog/apache/index.html`, then the URI is `/blog/apache/index.html`.

# PHP \$\_SERVER

- ◉ `$_SERVER['REMOTE_ADDR']`: The client's IP address.
- ◉ `$_SERVER['HTTP_USER_AGENT']`: The client's user agent, which typically offers information about both the operating system and browser.
- ◉ `$_SERVER['SERVER_SOFTWARE']` : Returns the server identification string (such as Apache/2.2.24)
- ◉ `$_SERVER['SERVER_ADDR']` : Returns the IP address of the host server
- ◉ `$_SERVER['SERVER_NAME']` : Returns the name of the host server (such as jayakumars.in)
- ◉ `$_SERVER['GATEWAY_INTERFACE']`: Returns the version of the Common Gateway Interface (CGI) the server is using



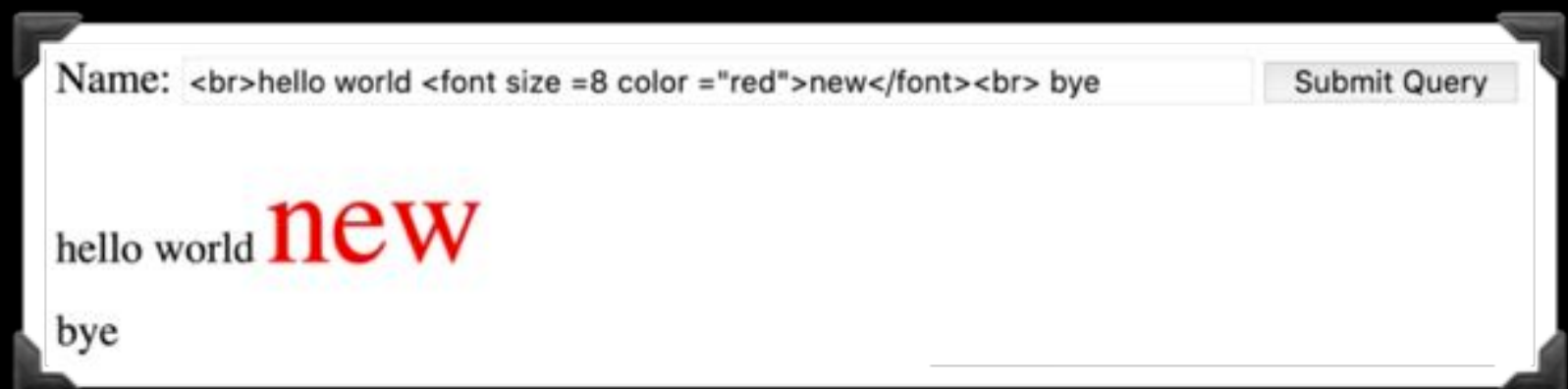
# PHP \$\_REQUEST

- ◉ PHP \$\_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.
- ◉ The form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable \$\_REQUEST to collect the value of the input field.

```

<body>
<form method="post" action="<?php echo
    $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit"> </form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
} ?>
</body>

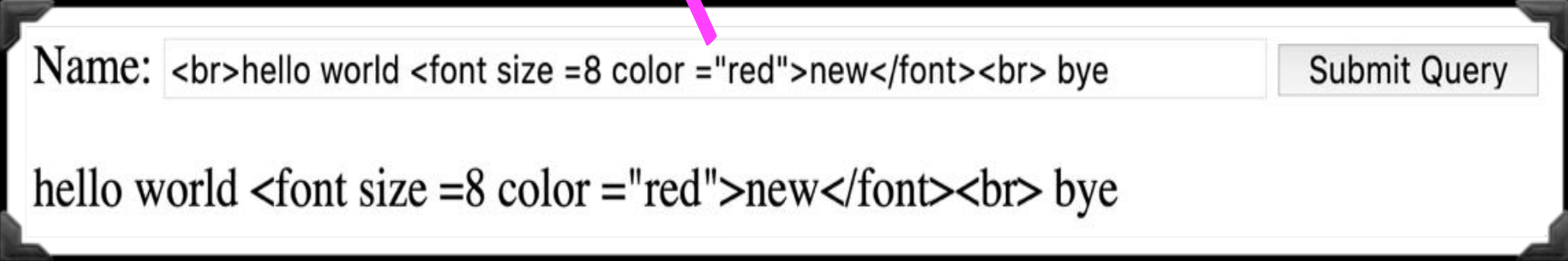
```



```

<body>
<form method="post" action="<?php echo
    $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = htmlspecialchars($_REQUEST['fname']);
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
} ?>
</body>

```



# \$\_GET

- ◉ PHP \$\_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".
- ◉ \$\_GET can also collect data sent in the URL.
- ◉ <http://www.example.com/index.html?cat=apache&id=157> was requested, you could access the following variables by using the \$\_GET superglobal:

```
✓ $_GET['cat'] = "apache"  
✓ $_GET['id'] = "157"
```

# \$\_POST

- ◉ The \$\_POST superglobal contains information pertinent to any parameters passed using the POST method.
- ◉ The following POST variables will be made available via the target **subscribe.php** script:

- ✓ `$_POST['email'] = "jayakumars@vit.ac.in";`
- ✓ `$_POST['pswd'] = "rainyday";`
- ✓ `$_POST['subscribe'] = "subscribe!";`

- ◉ **URL:** `http://www.example.com/index.html`

# Get vs Post Method

GET	POST
Data is submitted as a part of url	Data is submitted as a part of http request
Data is visible to the user	Data is not visible in the url
It is not secure but fast and quick	It is more secure but slower as compared to GET
We can pass 255 char as query string	We can pass unlimited data by this method as query string

# \$\_REQUEST – GET Method

```
<form action="myphp.php" method="GET">
Firstname: <input type="text" name="fname" />
Lastname: <input type="text" name="lname" />
<input type="submit" />
</form>
<?php
echo "Firstname: " , $_REQUEST["fname"];
echo "Lastname: " , $_REQUEST["lname"];
?>
```

# \$\_REQUEST – POST Method

```
<form action="myphp.php" method="POST">
Firstname: <input type="text" name="fname" />
Lastname: <input type="text" name="lname" />
<input type="submit" />
</form>
<?php
echo "Firstname: " , $_REQUEST["fname"];
echo "Lastname: " , $_REQUEST["lname"];
?>
```



# \$\_COOKIE

- ◉ The \$\_COOKIE superglobal stores information passed into the script through HTTP cookies.
- ◉ Such cookies are typically set by a previously executed PHP script through the PHP function setcookie().
- ◉ For example, suppose that you use setcookie() to store a cookie named example.com with the value ab2213.
- ◉ You could later retrieve that value by calling `$_COOKIE["example.com"]`

# \$\_SESSION

- ◉ The \$\_SESSION superglobal contains information regarding all session variables.
- ◉ Registering session information allows you the convenience of referring to it throughout your entire Web site, without the hassle of explicitly passing the data via GET or POST.
- ◉ Sessions are stored on the server, which means clients do not have access to the information you store about them.
- ◉ They work instead like a token allowing access and passing information while the user has their browser open.
- ◉ The problem with sessions is that when you close your browser you also lose the session.

# \$\_FILES

- ◉ The `$_FILES` superglobal contains information regarding data uploaded to the server via the POST method.
- ✓ `$_FILES["file"]["name"]` - the name of the uploaded file
- ✓ `$_FILES["file"]["type"]` - the type of the uploaded file
- ✓ `$_FILES["file"]["size"]` - the size in bytes of the uploaded file
- ✓ `$_FILES["file"]["tmp_name"]` - the name of the temporary copy of the file stored on the server
- ✓ `$_FILES["file"]["error"]` - the error code resulting from the file upload

# \$\_FILES

- ◉ **UPLOAD\_ERR\_OK**: The file was successfully uploaded.
- ◉ **UPLOAD\_ERR\_INI\_SIZE**: The file size exceeds the maximum size imposed by the `upload_max_filesize` directive.
- ◉ **UPLOAD\_ERR\_FORM\_SIZE**: The file size exceeds the maximum size imposed by an optional `MAX_FILE_SIZE` hidden form-field parameter.
- ◉ **UPLOAD\_ERR\_PARTIAL**: The file was only partially uploaded.
- ◉ **UPLOAD\_ERR\_NO\_FILE**: A file was not specified in the upload form prompt.

# \$\_FILES - Example

```
<?php
if ($_FILES["file"]["error"] > 0)
{
    echo "Error: " . $_FILES["file"]["error"];
}
else {
    echo "Upload: " . $_FILES["file"]["name"];
    echo "Type: " . $_FILES["file"]["type"];
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
}
?>
```

# \$\_ENV

- ◉ The \$\_ENV superglobal offers information regarding the PHP parser's underlying server environment.
- ◉ Some of the variables found in this array include:

`$_ENV['HOSTNAME']`: The server host name

`$_ENV['SHELL']`: The system shell

# Local Variables

- ◉ A variable declared in a function is considered local.
- ◉ That is, it can be referenced only in that function.
- ◉ Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function.
- ◉ Note that when you exit the function in which a local variable has been declared, that variable and its corresponding value are destroyed.

# Local Variables

```
<?
$x = 4;
function assignx () {
$x = 0;
echo "\$x inside function is $x. <br>";
}
assignx();
echo "\$x outside of function is $x. <br>";
?>
```

Output:

\$x inside function is 0.

\$x outside of function is 4.



# Static Variables

- ◉ In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable does not lose its value when the function exits, and will still hold that value if the function is called again.
- ◉ You can declare a variable as static simply by placing the keyword `STATIC` in front of the variable name.

## Using STATIC Keyword

```
<?
function static_track()
{
    STATIC $count = 0;
    $count++;
    print $count;
    print "<br>";
}
static_track();
static_track();
static_track();
?>
```

Output:

1  
2  
3

## Without STATIC Keyword

```
<?
function static_track(
{
    $count = 0; //not using
                STATIC keyword
    $count++;
    print $count;
    print "<br>";
}
static_track();
static_track();
static_track();
?>
```

Output:

1  
1  
1

# File Handling

# File Handling

- ◉ What is a file?

- A computer file is a block of information, or resource, which is available to a computer program.
- How to create a File? [Filename.txt Example]
- File Permissions [Read, Write, Execute]
- File Types
  - ✓ Text File [ASCII, Unicode ]
  - ✓ Media File [Mp3, Video, Flash]
  - ✓ System Files [Dynamic Link Library – DLL]
  - ✓ Executable Files [Exe, Deb, Rpm]

# Basic File Operations - for Text File

- ◉ `fopen(filename,mode)` = used for opening a file with specific mode
- ◉ `fclose(fp)` = used to close the file. (fp = filepointer)
- ◉ `feof(fp)` = used to find the End of File.
- ◉ `file exists(fp)` = Checks whether a file or directory exists
- ◉ `filesize(fp)` = Gets file size
- ◉ `filetype(fp)` = Gets file type
- ◉ `fstat(fp)` = Gets information about a file

# File Handling

Syntax:

```
$filename = "sample.txt";  
$fp = fopen($filename,"r") or  
    exit("Unable to open file");
```

# File Permissions or Modes:

- ◉ **r** - Read only Starts at the beginning of the file
- ◉ **r+** - Read/Write Starts at the beginning of the file
- ◉ **w** - Write only Opens and clears the contents of file or creates a new file if it does not exist
- ◉ **w+** - Read/Write Opens and clears the contents of file; or creates a new file if it does not exist

# File Permissions or Modes:

- ◉ **a** - Append Opens and writes to the end of the file or creates a new file if it does not exist
- ◉ **a+** - Read/Append Preserves file content by writing to the end of the file
- ◉ **x** - Write only Creates a new file. Returns FALSE and an error if file already exists
- ◉ **x+** - Read/Write Creates a new file. Returns FALSE and an error if file already exists



# Reading File Contents

- ◉ There are five methods to read the file contents
  - ✓ **fread(fp,size)** = File Reading Using Bytes
  - ✓ **fgetc(fp)** = File Reading Using Character
  - ✓ **fgets(fp)** = File Reading Using Lines
  - ✓ **file(fp)** = Reads entire file into an array
  - ✓ **file\_get\_contents(fp)** = Reads entire file into a string
  - ✓ **readfile(fp)** - function returns the number of bytes read on success

# Writing Contents into File

- ◉ **fwrite(fp,string)** = Binary-safe file write
- ◉ **file\_put\_contents(fp,string)** = Write a string to a file

# Formatted File Read and write

- ◉ **fscanf** = Parses input from a file according to a format
- ◉ **fprintf** = Write a formatted string to a stream

# PHP Check End-Of-File - feof()

- ◉ The `feof()` function checks if the "end-of-file" (EOF) has been reached.
- ◉ The `feof()` function is useful for looping through data of unknown length.

# Writing Contents into File

```
<?php
$filename = "welcome.txt";
$fp = fopen($filename,"w+") or exit("Unable to
    open file");
$content="welcome to php";
if(fwrite($fp,$content)==true)
    {    echo "sucess";    }
else
    {    echo "failed to written";    }

?>
```

*Thank You !!!*

