Web Application Development

Using

PHP – MYSQL

EVS Professional Training Institute
http://www.evslearning.com/
+92-42-111-685-822
+92-51-111-685-822

Compiled By :
Kashif Umar
kashif_umar@ymail.com

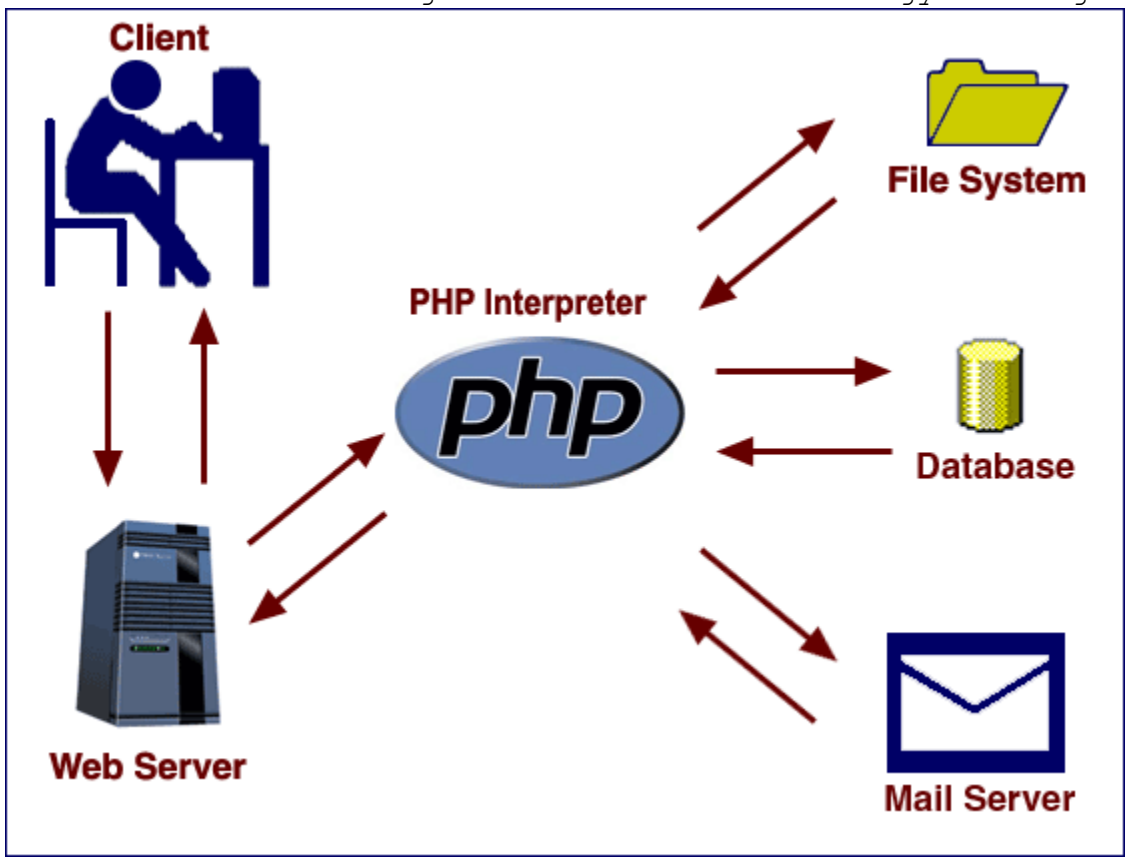
Week 1 – Lecture 1
PHP Basics & Introduction

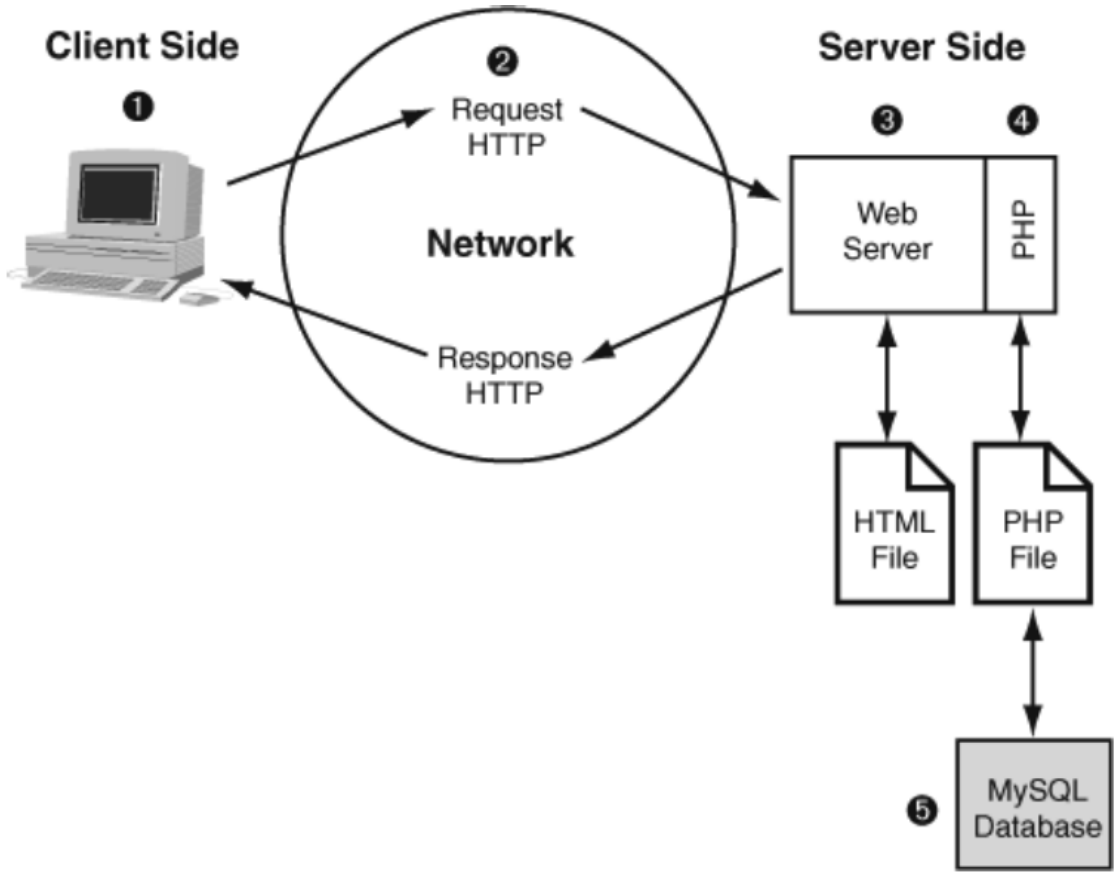
❖ Basics

- **Internet, WWW & HTTP**
- **Web Application development**
- **Static vs Dynamic Site – Dynamic Technology + Database**
- **Dynamic Technologies**
  - PHP, ASP.NET, JSP
- **Why Use PHP**
  - High Performance
  - Built-in libraries
  - Extensibility
  - Relative low cost
  - Portability
  - Developer community
  - Ease of learning
- **Why MySql**

❖ How PHP Works

- Describe the working of server-side technology working





❖ TOOLS

- **PHP**
- **Apache**
- **MySql**
- **XAMP / W.A.M.P**
- **Dreamweaver**
- **The php.ini file**
  - C:\Programs\xampp\php
  - Change tag styles

❖ Introduction to PHP

➢ PHP Tags

- ✦ <?php php-code ?> on by default
- ✦ <script language="php">php code </script> on by defult
- ✦ <? Php code ?> short_open_tag=off – php.ini
- ✦ <% php code %> asp_tags=off – php.ini

➢ PHP Statements and White space

- ✦ print "This is a PHP statement.";
- ✦ print "This is also a PHP statement.";
- ✦ print "This is also a PHP

statement.";

➢ Comments

- ✦ // single line comment
- ✦ # single line comment
- ✦ /* */ multi-line comment

➢ PHP Functions

- ✦ Simple introduction, PHP has many built-in functions
- ✦ Phpinfo()

➢ Hello World!

- ✦ echo("Hello World" . "
");
- ✦ echo('Hello World' . '
');
- ✦ echo "Hello World" . "
";
- ✦ echo 'Hello World' . '
';
- ✦ print("Hello World" . "
");
- ✦ print('Hello World' . '
');
- ✦ print "Hello World" . "
";
- ✦ print 'Hello World' . '
';

❖ Lets Start PHP

➢ Variables

- ✦ **Variable Types**
- ✦ **Variable Names (Identifiers)**
  - ⅄ PHP supports a number of different variable types—Booleans, integers, floating point numbers, strings, arrays, objects, resources, and NULL
  - ⅄ The language can automatically determine variable type by the context in which it is being used.

- Every variable has a name, which is preceded by a dollar ($) symbol, and it must begin with a letter or underscore character, optionally followed by more letters, numbers, and underscores.
- $popeye, $one_day, and $INCOME are all valid
- Variable names in PHP are case-sensitive; $count is different from $Count or $COUNT.
- Boolean, Integer, Float, String
- PHP variables must start with a letter or underscore "_".
- PHP variables may only be comprised of alpha-numeric characters and underscores. a-z, A-Z, 0-9, or _ .
- Variables with more than one word should be separated with underscores. $my_variable
- Variables with more than one word can also be distinguished with capitalization. $myVariable

✦ **Type Strength**

- PHP is weakly typed, meaning that variables do not need to be assigned a type (e.g, Integer) at the time they are declared. Rather, the type of a PHP variable is determined by the value the variable holds and the way in which it is used.

✦ **Variable Scope**

- Superglobal - Superglobal variables are predefined arrays, including $_POST and $_GET. They are accessible from anywhere on the page.
- Global - Global variables are visible throughout the script in which they are declared. However, they are not visible within functions in the script unless they are re-declared within the function as global variables.
- Function - Variables in the function scope are called local variables. Local variables are local to the function in which they are declared.

✦ **Constants**

- Constants are like variables except that, once assigned a value, they cannot be changed. Constants are created using the define() function and by convention (but not by rule) are in all uppercase letters. Constants can be accessed from anywhere on the page.

```
define('CONST_NAME',VALUE);
echo (CONST_NAME);
```

- **Variable-Testing and Manipulation Functions**
  - gettype($name)
  - settype($name, "type")
  - isset(variable);
  - unset(variable);
  - empty(variable);
  - is_bool() Checks if a variable or value is Boolean
  - is_string() Checks if a variable or value is a string
  - is_numeric() Checks if a variable or value is a numeric string
  - is_float() Checks if a variable or value is a floating point number
  - is_int() Checks if a variable or value is an integer
  - is_null() Checks if a variable or value is NULL
  - is_array() Checks if a variable is an array
  - is_object() Checks if a variable is an object

- **References**
  - Another way to assign a value to a variable is to create a reference (PHP 4). A reference is when one variable is an alias or pointer to another variable; that is, they point to the same underlying data. Changing one variable automatically changes the other. This might be useful in speeding things up when using large arrays and objects,

```
<?php
$age = 26;
$old = &$age; // This is a valid assignment.
$old = &(26 + 7); // Invalid; references an unnamed expression.
?>
```

- **Pre-Defined Variables**
  - DOCUMENT_ROOT : The full path of the Web's document root, normally where HTML pages are stored and defined in the server's configuration file.
  - HTTP_USER_AGENT : Identifies the type of Web browser to the server when it requests a file.
  - HTTP_REFERER : The full URL of the page that contained the link to this page. Of course if there isn't a referring page, this variable would not exist.
  - REMOTE ADDRESS : The remote IP address of the client machine that requested the page.

- **PHP Operators**
  - **Arithmetic Operators**
    - +, -, *, /, %, +=, -=, *=, /=, %=, ++, --
    - Post Increment/Decrement, Pre Increment/Decrement
  - **String Operators**
    - .
  - **Assignment Operators**
    - =
  - **Comparison Operators**
    - <, >, >=, <=, ==, !=, <>
  - **===, !=== Operator**
    - The === and !== operators test that their operands are not only of the same value, but also of the same data type. String "54" is equal to number 54, but not identical because one is a string and the other is a number, even though their values are equal.
  - **Logical Operators**
    - &&, and, ||, or, !
  - **Other Operators**
    - ?: ternary $foo = ($age >= 18) ? 'adult' : 'child';
    - @ Error suppression $a = @(1/0);
  - **Operator Precedence**
    -
  - **Single Quotes vs Double Quotes**
    - In PHP, for simple strings you can use single quotes and double quotes interchangeably. However, there is one important difference of which you need to be aware. Text within single quotes will not be parsed for variables and escape sequences

```
$person = 'George';
echo '\"Hello\"$person!!';

$person = 'George';
echo "\"Hello\"$person!!";
```

- **Flow Control**

  Conditional processing allows programmers to output different code based on specific conditions. There are two conditional structures in PHP - if-else if-else and switch/case.

✦ Conditional Processing

✦ If Conditions

- Simple if statement

  Syntax

```
if (conditions)
     Do this;
```

  In the above code, the Do this; statement will either run or not run depending on whether or not the conditions are true. This syntax can only be used when the condition affects a single line of code. For a block of code, use the following syntax.

  Syntax

```
if (conditions)
{
     Do this;
     Then do this;
     And this too;
}
```

  The lines of code affected by the if condition are put in a code block, which is surrounded by curly brackets to indicate that all of the code either should or should not be executed, depending on the result of the if condition.

- if-else statement

  Syntax

```
if (conditions)
{
     Do this;
}

else
{
     Do that;
}
```

- if-else if-else statement
  Syntax

```
if (conditions)
{
     Do this;
}

else if (other conditions)
{
     Do that;
}

else
{
     Do this other thing;
}
```

The two syntax blocks above show an if-else and an if-else if-else statement, which can have any number of else if blocks.

- Code Sample: FlowControl/Demos/If.php

```
<html>
<head>
<title>if-elseif-else</title>
</head>
<body>
<?php
   $Age = 21;
   if ($Age >= 21)
   {
        echo 'You can vote and drink!';
   }
   else if ($Age >= 18)
   {
        echo 'You can vote, but can\'t drink.';
   }
   else
   {
        echo 'You cannot vote or drink.';
   }
?>
</body>
</html>
```

- **Compound If Statements**
  More complex if statements often require that several conditions be checked. Logical operators are used to make compound conditions.

  - Code Sample: FlowControl/Demos/If2.php

```
<html>
<head>
<title>if-elseif-else</title>
</head>
<body>
<?php
     $Age = 21;
     $Citizen = false;
     if ($Age >= 21 && !$Citizen)
     {
      echo 'You can drink, but can\'t vote.';
     }
     elseif ($Age >= 21)
     {
      echo 'You can vote and drink!';
     }
     elseif ($Age >= 18 && $Citizen)
     {
      echo 'You can vote, but can\'t drink.';
     }
     else
     {
      echo 'You cannot vote or drink.';
     }
?>
</body>
</html>
```

- **Switch/case**
  A switch/case statement is similar to an if statement, except that it can only check for an equality comparison of a single expression. It cannot, for example, be used to check if one value is higher than another.

  - Syntax

```
switch (expression)
{
   case 'a' :
        echo 'expression is a';
   break;
```

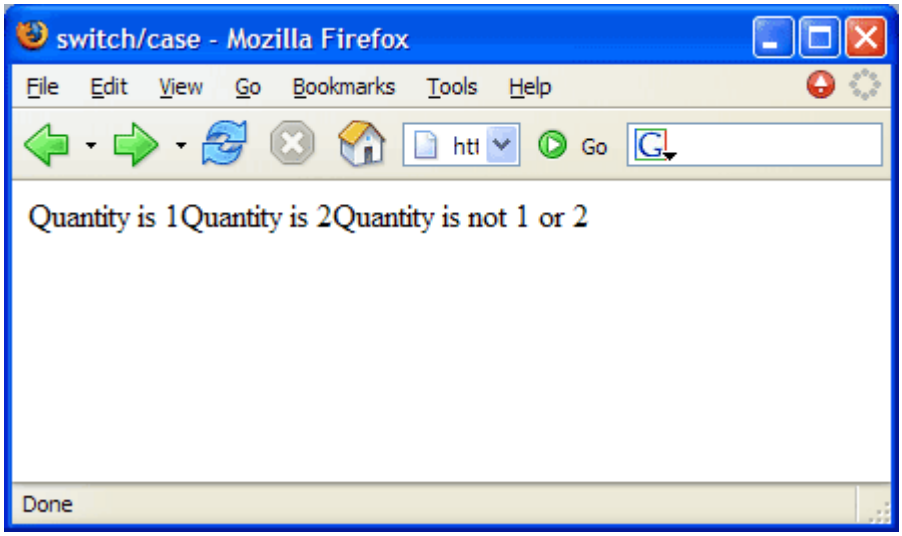
```
     case 'b' :
          echo 'expression is b';
     break;

     case 'c' :
          echo 'expression is c';
     break;

     default :
          echo 'expression is unknown';
     break;
   }
```

The break statement is important. Without it, after a single match is found, all following statements will execute.

The following example demonstrates a switch/case statement without break statements.

➢ Code Sample: FlowControl/Demos/Switch.php

```
<html>
<head>
<title>switch/case</title>
</head>
<body>
<?php
$Quantity = 1;
switch ($Quantity)
{
   case 1 :
        echo 'Quantity is 1';
   case 2 :
        echo 'Quantity is 2';
   default :
        echo 'Quantity is not 1 or 2';
}
?>
</body>
</html>
```

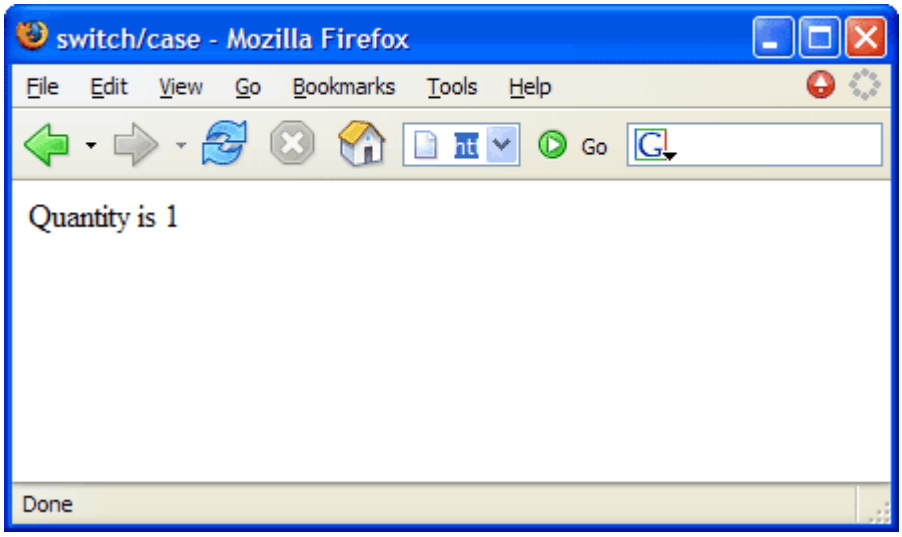


When you run this script, you would notice that, once a match is found, all remaining echo statements are output. The following example shows how this can be fixed by adding break statements.

➢ Code Sample: FlowControl/Demos/SwitchWithBreak.php

```
<html>
<head>
<title>switch/case</title>
</head>
<body>
<?php
$Quantity = 1;
switch ($Quantity)
{
 case 1 :
  echo 'Quantity is 1';
  break;
 case 2 :
  echo 'Quantity is 2';
  break;
 default :
  echo 'Quantity is not 1 or 2';
}
?>
</body>
</html>
```

This time, only the first statement is output:



- **Loops**

  As the name implies, loops are used to loop (or iterate) over code blocks. The following section shows the syntax for different types of loops. Each loop will return "12345".

  There are several types of loops in PHP.

  - while
  - do...while
  - for
  - foreach

  - **while**

    while loops are used to execute a block of code repeatedly while one or more conditions is true.

    Syntax

```
$a=1;
while ($a < 6)
{
 echo $a;
 $a++;
}
```

  - **do...while**

    do...while loops are used to execute a block of code repeatedly until one or more conditions is found to be false. The difference between while loops and do...while loops is that the condition is checked after the code block is executed. This means that, in

a do...while loop, the code block will always be executed at least once.

Syntax

```
$a=1;
do
{
 echo $a;
 $a++;
}
while ($a < 6);
```

- **for**

A for loop takes three expressions separated by semi-colons and grouped in parentheses before the block to be iterated through.

1. The first expression is executed once before the loop starts. It is usually used to initialize the conditions.

2. The second expression is evaluated before each iteration through the loop. If it evaluates to false, the loop ends.

3. The third expression is executed at the end of each iteration through the loop. It is usually used to make changes that can affect the second expression.

Syntax

```
for ($a=1; $a < 6; $a++)
{
 echo $a;
}
```

- **break and continue**

To break out of a loop, insert a break statement.

Syntax

```
for ($a=1; $a < 6; $a++)
{
 echo $a;
```

```
 if ($a > 3)
 {
  break;
 }
}
```

To jump to the next iteration of a loop without executing the remaining statements in the block, insert a continue statement.

Syntax

```
for ($a=1; $a < 6; $a++)
{
 if ($a == 3)
 {
  continue;
 }
 echo $a;
}
```

- **foreach**

  The foreach loop is designed to work with arrays and works only with arrays. Arrays are covered later, but because we are talking about looping constructs, the foreach loop should be mentioned here.

  An array is a list of items, like an array of numbers or strings. The loop expression consists of the array name, followed by the as keyword, and a user-defined variable that will hold each successive value of the array as the loop iterates. The foreach loop, as the name implies, works on each element of the array, in turn, moving from left to right, until all of the elements of the array have been processed. The loop expression is followed by a block of statements that will be executed for each item in the expression.

Syntax

```
$array_name=array( item1, item2, item3, ...);

foreach ($array_name as $value)
{
     do-something with the element's value;
}
```

Example:

```
$suit=array("diamond", "spade", "club", "heart");
// An array
foreach ( $suit as $card_type)
{
     echo $card_type . "<br />";
     // displays: diamond heart
}
```