UNIT – II

WORKING WITH BASIC BUILDING BLOCKS OF PHP

2.1 Introduction to PHP

- PHP stands for *PHP: Hypertext Preprocessor.*
- "PHP is an open-source, server-side, HTML-embedded Web-scripting language for building dynamic and interactive Web sites. "
- PHP programs run on a Web server. One of the key features of PHP is that you can embed PHP code within HTML Web pages, making it very easy for you to create dynamic content quickly.
- Its flexibility and relatively small learning curve make it one of the most popular scripting languages around.

2.2 A Brief History of PHP

- PHP was created by Rasmus Lerdorf in 1994.
- He released PHP to the general public in 1995, and called it PHP version 2.
- In 1997, Zeev Suraski and Andi Gutmans, rewrote most of PHP along with Rasmus, and released PHP version 3.0 in June 1998.
- PHP 4, was launched in May 2000. This version included session handling features, output buffering, a richer core language, and support for a wider variety of Web server platforms.
- PHP 5, released in July 2004 supports object - oriented programming (OOP) implementation, with private and protected class members; final, private, protected, and static methods; abstract classes; interfaces; and standardized constructor/destructor syntax.

2.3 How PHP Works?

• PHP file Structure

- The file must have the .php extension.
- If the file has an .html extension, the PHP code will not be executed.
- A PHP scripting block always starts with <?php and ends with ?>.
- A PHP scripting code can be placed anywhere in the document between <?php and ?>.

2.3 How PHP Works?

- **PHP start and end tags**

- There are **four** different pairs of opening and closing tags which can be used in php.

1. Standard PHP tags
2. Script Tags
3. Short Tags
4. ASP Style Tags

2.3 How PHP Works?

- **PHP start and end tags**

1. **Standard PHP tags**

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

2.3 How PHP Works?

- **PHP start and end tags**

  - **2. Script Tags**

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

2.3 How PHP Works?

- **PHP start and end tags**

3. Short Tags

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

- Short tags are only available when they are enabled via the short_open_tag in *php.ini* configuration file directive.

2.3 How PHP Works?

- **PHP start and end tags**

4. **ASP Style Tags**

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

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

2.3 How PHP Works?

- **PHP start and end tags**

- Two of those, <?php ?> and <script language="php"> </script>, are always available.

- The other two are short tags and ASP style tags, and can be turned on and off from the *php.ini* configuration file.

2.3 How PHP Works?

- **Commenting Codes**

- In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

- **Single Line comment:**

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

The single line comment starts with symbol // or #.

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

2.3 How PHP Works?

- **Commenting Codes**

- **Multiline Comment**

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

2.4 Creating and Saving a PHP file

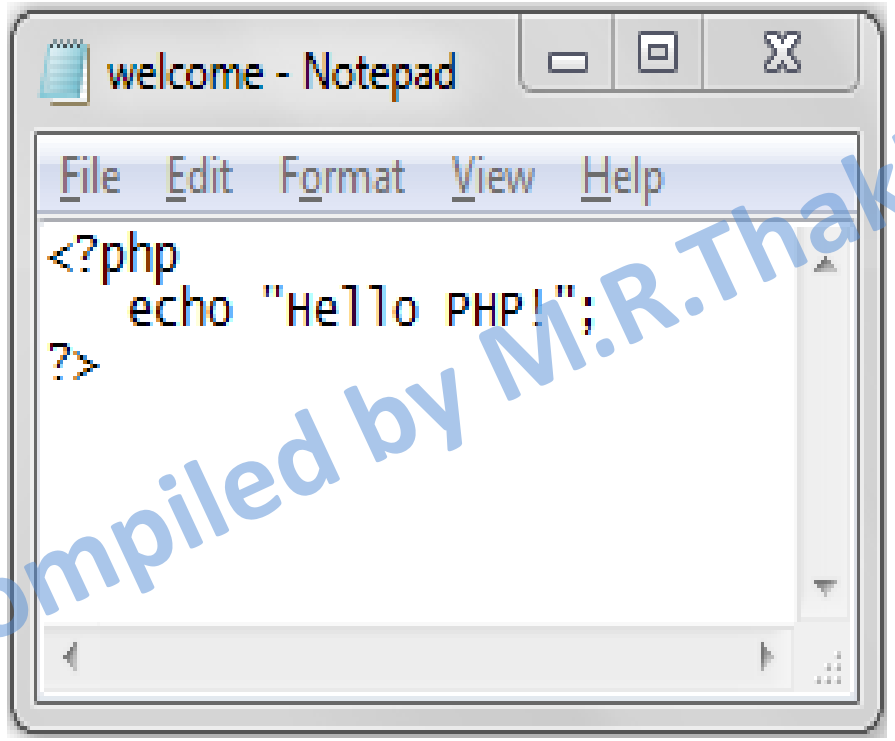
- To create PHP page you can use any simple editor like notepad or rich editor like Macromedia Dreamweaver.
- The page must be saved with the extension ".php".

- PHP is a web scripting language which means it was designed to work with HTML. You can easily embed PHP code into your HTML code.

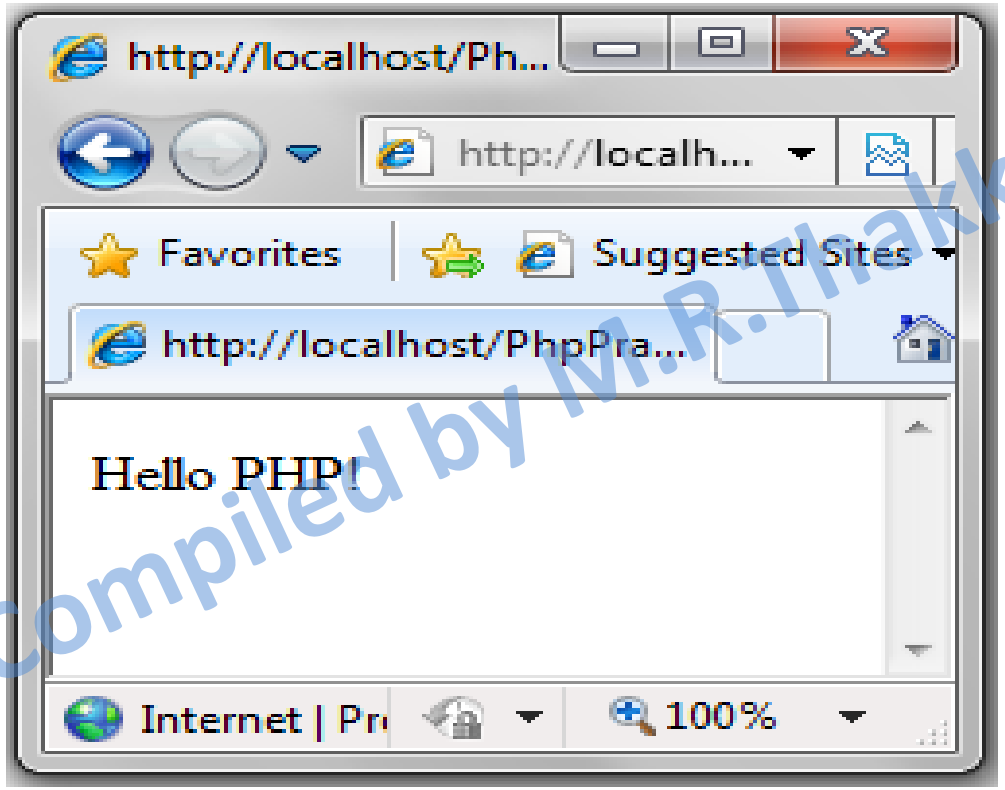
- If you have PHP inserted into your HTML and want the web browser to interpret it correctly, PHP code should be surrounded with <?php … ?> tags.

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

2.4 Creating and Saving a PHP file



2.4 Creating and Saving a PHP file



2.5 Output Statement : echo & Print

- There are two basic statements to output text with PHP:
  - echo
  - print

- echo

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

- print.

```
<?php
    print “Hello World”;
?>
```

2.6 Installing PHP

- **Covered in Laboratory**

2.7 PHP Variables & Value types

- A variable is a holder for a type of data. So, based on its type, a variable can hold numbers, strings, Booleans, objects, resources or it can be NULL.

- The correct way of setting a variable in PHP:

```
$var_name = value;
```

2.7 PHP Variables & Value types

- ***Variable Naming Rules***

- A variable name must start with a letter or an underscore "_" .

- A variable name can only contain alpha-numeric characters and underscores (a-Z, 0-9, and _) .

- A variable name should not contain spaces.

- If a variable name is more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString)

2.7 PHP Variables & Value types

- **Valid Variable names :**

- $var = 'Bob';
  $Var = 'Joe';
  $_4site = 'not yet'; // valid; starts with an underscore

- **Invalid Variable names:**
- $4site = 'not yet';

2.7 PHP Variables & Value types

- **Data types**
- PHP has a total of eight data types which we use to construct our variables:
  - **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 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).

2.8 PHP Operator

- PHP language supports following type of operators:
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators
  - Assignment Operators
  - Conditional (or ternary) Operators

2.8 PHP Operator

- Arithmetic Operators
- following arithmetic operators supported by PHP language:
- Assume variable A holds 10 and variable B holds 20 then:

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 denumerator	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

2.8 PHP Operator

- Comparison Operators
- There are following comparison operators supported by PHP language. Assume variable A holds 10 and variable B holds 20 then:

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.

2.8 PHP Operator

- Logical Operators
- There are following logical operators supported by PHP language. Assume variable A holds true and variable B holds true then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

2.8 PHP Operator

- Assignment Operators
- There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne 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

2.8 PHP Operator

- Conditional (or ternary) Operators

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

```
$result = ($a > $b ) ? $a :$b;
```

2.9 Operator Precedence, Constants, Predefine Constants

- **Operator Precedence**
- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.
- Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:
- For example: x = 7 + 3 * 2;
- Here x is assigned 13, not 20 because operator * has higher precedence than +, so it first get multiplied with 3*2 and then adds into 7.

2.9 Operator Precedence, Constants, Predefine Constants

- Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

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

2.9 Operator Precedence, Constants, Predefine Constants

- **Constants**

- A constant is an identifier (name) for a Literal value.
- The value of the constant cannot be changed during the script execution.
- A valid constant name starts with a letter or underscore, no $ sign before the constant name.

- **Creating Constant :**
- To create a constant, use the define() function.
- **Syntax:** define (constant_*name*, constant_*value)*
- **Example:** define("CONSTANT", "Hello world.");

2.9 Operator Precedence, Constants, Predefine Constants

- **Predefine Constants**

- PHP provides a large number of predefined constants to any script which it runs.

- There are **five magical constants** that change depending on where they are used.

- For example, the value of __LINE__ depends on the line that it's used on in your script.

2.9 Operator Precedence, Constants, Predefine Constants

- **Predefine Constants**

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances.
__FUNCTION__	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

2.10 Flow Control Statements

- PHP supports two Flow Control statements: **if and switch.**

- These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

2.10 Flow Control Statements

- **Simple If**

```
if (Expression )
{
  true-block ;
}
```

- **if-then-else**

```
if (Expression )
{
  true-block ;
}
else
{
  false-block ;
}
```

2.10 Flow Control Statements

- **nested-if**

```
if (Expression )
{
        if (Expression )
        {
                true-block ;
        }
        else
        {
                false-block ;
        }
}
else
{
  false-block ;
}
```

2.10 Flow Control Statements

- **if-else-if Ladder**

```
if(condition)
        statement;
else if(condition)
    statement;
else if(condition)
    statement;
...
else
statement;
```

2.10 Flow Control Statements

- **Switch**

```
switch (expression)
{
        case value1:
                // statement sequence
                break;
        case value2:
                // statement sequence
                break;
        ...
        case valueN:
                // statement sequence
                break;
        default:
                // default statement sequence
}
```

2.11 Loops

- In programming it is often necessary to repeat the same block of code a given number of times, or until a certain condition is met.
- This can be accomplished using looping statements.
- PHP supports following looping statements :
  - while
  - do-while
  - for
  - foreach

2.11 Loops

- **Loops (while)**

```
while (conditional expression)
{
    statements block
}
```

2.11 Loops

- **Loops (do-while)**

```
do
{
    Statement bolck;

} while (condition);
```

2.11 Loops

- **Loops (for)**

```
for ( initialization; termination; increment)
{
      statements;
}
```

2.11 Loops

- **Loops (foreach)**

- The foreach loop is a variation of the for loop and allows you to iterate over elements in an array.


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

2.11 Loops

- **Loops (foreach)**

```
<?php

$a  = array(10,20,30);

foreach  ( $a  as  $value)
{
        echo $value . "<br />";
}

?>
```


- **Output:**

```
10
20
30
```

2.12 Jump Statements

- PHP supports following jump statements :
  - break
  - continue

2.12 Jump Statements

- **Break statement**

- Break statement ends a loop immediately even if the condition being tested is still true.

- **Example:**

```
for($i=1 ; $i<= 5 ; $i++ )
{
      if( $i == 3 )
      {
        break;
      }
      echo $i , “<BR />”;
}
```

2.12 Jump Statements

- **continue statement**

- The continue statement causes the loop to exit its current iteration through the loop and start over at the first statement of the loop.

- **Example:**

```
for($i=1 ; $i<= 5 ; $i++ )
{
        if( $i == 3 )
        {
          continue;
        }
        echo $i , “<BR />”;
}
```

****