Unit – II
Working with Basic Building Blocks of PHP

PHP Introduction

PHP is a recursive acronym for “PHP: Hypertext Preprocessor” -- It is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

PHP Introduction

- PHP is a server-side scripting language
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is open source software
- PHP is free to download and use

PHP Introduction

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side.

PHP Introduction

Some info on MySQL which we will cover in the next workshop.

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

PHP Introduction

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something" (like in the next slide, it outputs "Hi, I'm a PHP script!").

The PHP code is enclosed in special start and end processing instructions **<?php** and **?>** that allow you to jump into and out of "PHP mode."

PHP Start & END Tags

1. **Standard PHP Tags**

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

2. **Script Tags**

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

PHP Start & END Tags

3. **Short Tags**

```
<?
        echo “Hello World”;
?>
```

Requires short_open_tag.ini in configuration file directive or if php is configured with the enable-short-tags option.

4. **ASP Style Tags**

```
<%
        echo “Hello World”;
%>
```

PHP Introduction

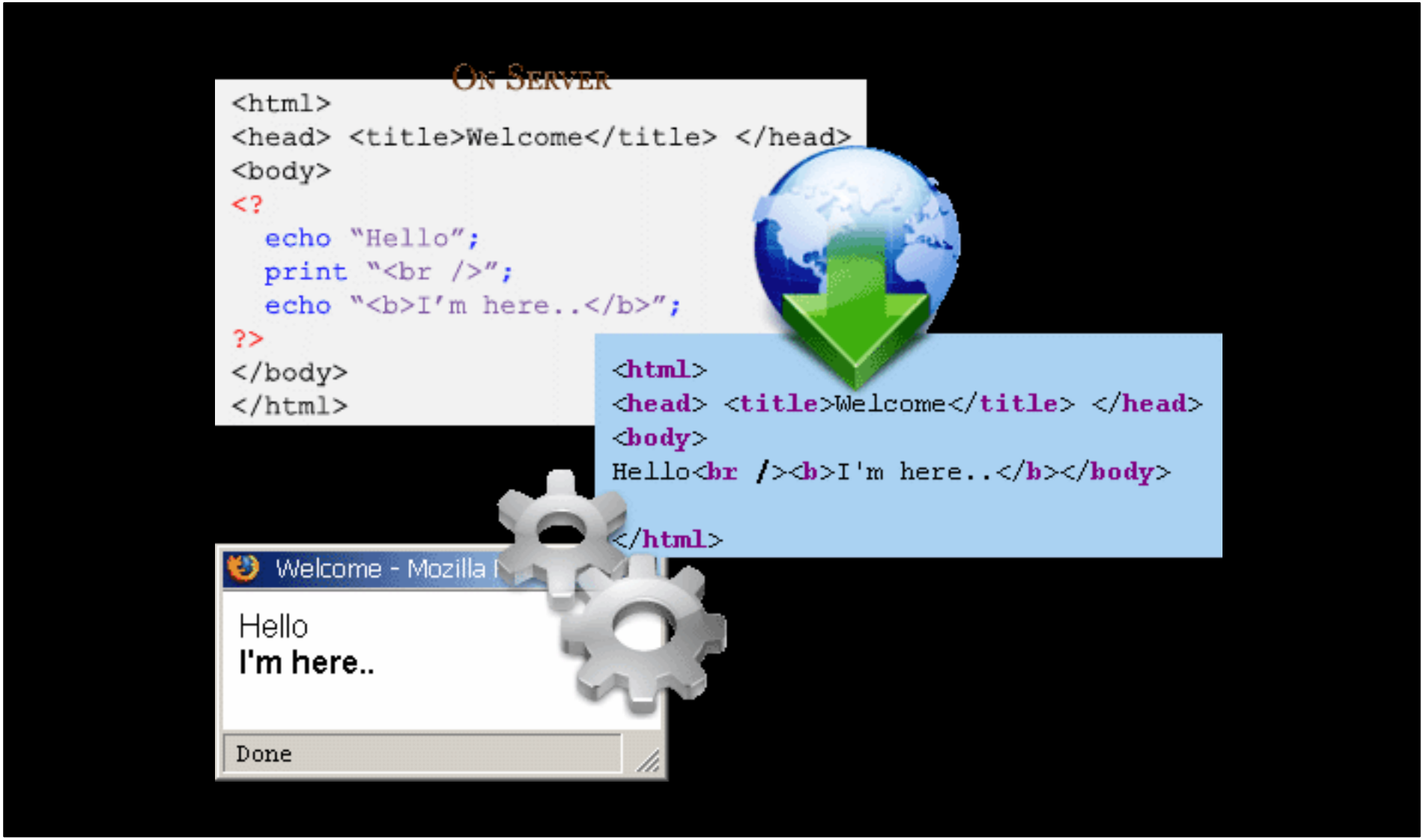
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

PHP Introduction

PHP code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was.

A visual, if you please...

PHP Introduction



Brief History of PHP

- PHP (PHP: Hypertext Preprocessor) was created by Rasmus Lerdorf in 1994. It was initially developed for HTTP usage logging and server-side form generation in Unix.
- PHP 2 (1995) transformed the language into a Server-side embedded scripting language. Added database support, file uploads, variables, arrays, recursive functions, conditionals, iteration, regular expressions, etc.
- PHP 3 (1998) added support for ODBC data sources, multiple platform support, email protocols (SNMP,IMAP), and new parser written by Zeev Suraski and Andi Gutmans .

Brief History of PHP

- PHP 4 (2000) became an independent component of the web server for added efficiency. The parser was renamed the Zend Engine. Many security features were added.
- PHP 5 (2004) adds Zend Engine II with object oriented programming, robust XML support using the libxml2 library, SOAP extension for interoperability with Web Services, SQLite has been bundled with PHP

PHP Hello World

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

Above is the PHP source code.

PHP Hello World

It renders as HTML that looks like this:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
 </body>
</html>
```

PHP Hello World

This program is extremely simple and you really did not need to use PHP to create a page like this. All it does is display: Hello World using the PHP **echo()** statement.

Think of this as a normal HTML file which happens to have a set of special tags available to you that do a lot of interesting things.

Displaying Data

There are two language constructs available to display data: print() and echo().

They can be used with or without brackets.

Note that the data ‘displayed’ by PHP is actually parsed by your browser as HTML. View source to see actual output.

Displaying data

```
<?php
echo 'Hello World!<br />';
echo('Hello World!<br />');
print 'Hello World!<br />';
print('Hello World!<br />');
?>
```

Escaping Characters

Some characters are considered ‘special’

Escape these with a backslash \

Special characters will be flagged when they arise, for example a double or single quote belong in this group…

Escaping Characters

```
<?php
// Claire O'Reilly said "Hello".
echo 'Claire O\'Reilly ';
echo "said \"Hello\".";
?>
```

PHP Data Types

- Variables can store data of different types, and different data types can do different things.
- PHP supports the following data types:

    - String
    - Integer
    - Float (floating point numbers - also called double)
    - Boolean
    - Array
    - Object
    - NULL
    - Resource

PHP String

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

A string can be any text inside quotes. You can use single or double quotes:

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>
```

PHP Integer

- An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
- Rules for integers:
    - An integer must have at least one digit
    - An integer must not have a decimal point
    - An integer can be either positive or negative
    - Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
    - In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

Example

```
<?php
    $x = 5985;
    var_dump($x);
?>
```

PHP Float

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

- In the following example $x is a float. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

PHP Boolean

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

- $x = true;
- $y = false;
- Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

- An array stores multiple values in one single variable.

- In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

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

PHP Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
    function Car() {
        $this->model = "VW";      }    }
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
```

PHP NULL Value

- 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.
- Tip: 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:
  - Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

PHP Resource

- The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.
- A common example of using the resource data type is a database call.
- We will not talk about the resource type here, since it is an advanced topic.

PHP Comments

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

Do not nest multi-line comments

// recommended over #

```
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>

</body>
</html>
```

Comments

```
<?php
// this is a comment
echo 'Hello World!';
/* another
   multi-line comment */
?>
```

Constants

Constants (unchangeable variables) can also be defined.

Each constant is given a name (note no preceding dollar is applied here).

By convention, constant names are usually in UPPERCASE.

Constants

```
<?php
define('NAME','Phil');
define('AGE',23);
echo NAME;
echo ' is ';
echo AGE;
// Phil is 23
?>
```

Predefined Constants

- These constants are defined by the PHP core. This includes PHP, the Zend engine, and SAPI modules.
- **PHP_VERSION (string)** :The current PHP version as a string in "major.minor.release[extra]" notation.
- **PHP_MAJOR_VERSION (integer) :** The current PHP "major" version as an integer (e.g., int(5) from version "5.2.7-extra"). Available since PHP 5.2.7.
- **PHP_MINOR_VERSION (integer) :** The current PHP "minor" version as an integer (e.g., int(2) from version "5.2.7-extra"). Available since PHP 5.2.7.
- **PHP_RELEASE_VERSION (integer) :** The current PHP "release" version as an integer (e.g., int(7) from version "5.2.7-extra"). Available since PHP 5.2.7.
- **PHP_VERSION_ID (integer) :** The current PHP version as an integer, useful for version comparisons (e.g., int(50207) from version "5.2.7-extra"). Available since PHP 5.2.7.

Magic Constants

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

Magic Constants

Sr. No	Name	Description
1	__LINE__	The current line number of the file.
2	__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.
3	__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.
4	__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.
5	__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP is a Loosely Typed Language

- PHP automatically converts the variable to the correct data type, depending on its value.
- In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.
- But in PHP we need not to tell datatype while declaring the variable and we can change the variable value as well like can store integer value in string as well.

Scope of Variable

- Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types as below
- **Local variables** : A variable declared in a function is considered local; that is, it can be referenced solely 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.
- **Function parameters** : Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be called in function.

Scope of Variable

- **Global variables** :global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL** in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name.
- **Static variables** : The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again. You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

PHP Variables

- Variables are used for storing values, like text strings, numbers or arrays.
- When a variable is declared, it can be used over and over again in your script.
- All variables in PHP start with a $ sign symbol.
- The correct way of declaring a variable in PHP:

```
$var_name = value;
```

PHP Variables

```
<?php
$txt="Hello World!";
$x=16;
?>
```

- In PHP, a variable does not need to be declared before adding a value to it.
- In the example above, you see that you do not have to tell PHP which data type the variable is.
- PHP automatically converts the variable to the correct data type, depending on its value.

PHP Variables

- A variable name must start with a letter or an underscore "_" -- not a number
- A variable name can only contain alpha-numeric characters, underscores (a-z, 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 an underscore ($my_string) or with capitalization ($myString)

Variables: example

```php
<?php
$name = 'Phil';
$age  = 23;
echo $name;
echo ' is ';
echo $age;
// Phil is 23
?>
```

settype() function

- The settype() function is used to set the type of a variable.

Syntax

bool settype(var_name, var_type)

- **var_name** : The variable being converted and mandatory.
- **var_type** : Type of the variable. Possible values are : boolean, integer, float, string, array, object, null.
- This functions Boolean value true or false.

```
<?php
$var1='98';
$var2='01';
settype($var1, "integer");
settype($var2, "integer");
echo ($var1.'<br>');
echo ($var2.'<br>');
echo ($var1+$var2.'<br>');
?>
```

```
98
1
99
```

gettype()

- The gettype() function is used to get the type of a variable.

Syntax

datatypes gettype(var_name)

- **var_name** : The name of variable and manda

```
<?php
    echo gettype(102).'<br>';
    echo gettype(true).'<br>';
    echo gettype(' ').'<br>';
    echo gettype(null).'<br>';
    echo gettype(array()).'<br>';
    echo gettype(new stdclass());
?>
```

integer
boolean
string
NULL
array
object

PHP Concatenation

- The concatenation operator (.) is used to put two string values together.
- To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

PHP Concatenation

The output of the code on the last slide will be:

```
Hello World! What a nice day!
```

If we look at the code you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

PHP Operators

Operators are used to operate on values. There are four classifications of operators:

- Arithmetic
- Assignment
- Comparison
- Logical
- ? Ternary

PHP Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

PHP Operators

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

PHP Operators

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

PHP Operators

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

? Ternary Operator

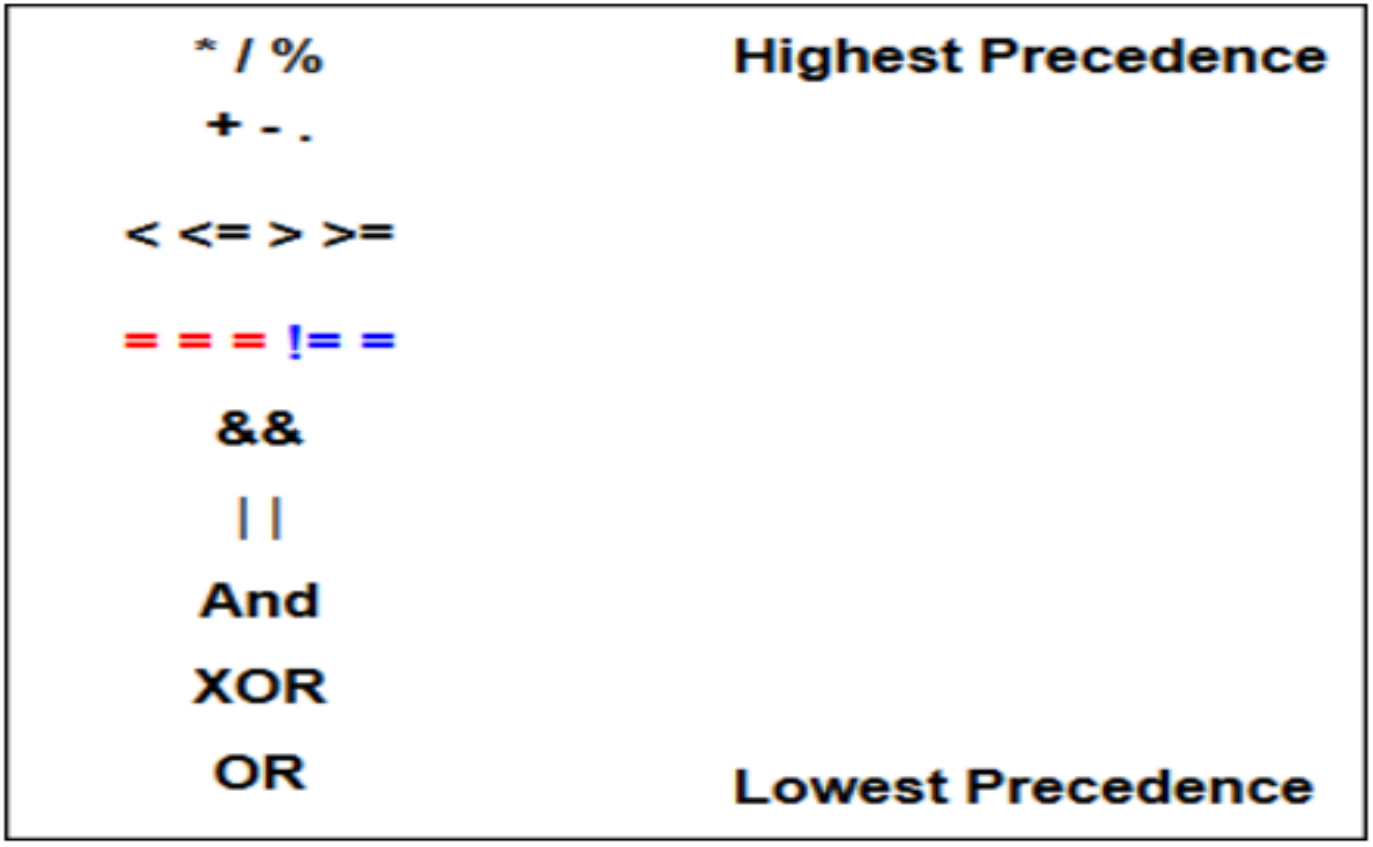
- Ternary operator logic is the process of using "(condition) ? (true return value) : (false return value)" statements to shorten your if/else structures.

```
$var = 5;

$greater_than_two = ($var > 2 ? true : false); // returns true
```

Advantages of ? Operator

- Makes coding simple if/else logic quicker
- You can do your if/else logic inline with output instead of breaking your output building for if/else statements
- Makes code shorter
- Makes maintaining code quicker, easier

Operator Precedence



PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
- In PHP we have the following conditional statements...

PHP Conditional Statements

- **if** statement - use this statement to execute some code only if a specified condition is true
- **if...else** statement - use this statement to execute some code if a condition is true and another code if the condition is false
- **if...elseif....else** statement - use this statement to select one of several blocks of code to be executed
- **switch** statement - use this statement to select one of many blocks of code to be executed

PHP Conditional Statements

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

PHP Conditional Statements

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

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Conditional Statements

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces **{ }**

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

PHP Conditional Statements

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Conditional Statements

Use the switch statement to select one of many blocks of code to be executed.

```
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 both label1 and label2;
}
```

PHP Conditional Statements

For switches, first we have a single expression n (most often a variable), that is evaluated once.

The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed.

Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

PHP Conditional Statements

```
<html>
<body>

<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

PHP Loops

- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
- In PHP, we have the following looping statements:

PHP Loops

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a 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

PHP Loops - While

The while loop executes a block of code while a condition is true. The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

PHP Loops - While

Output:

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

PHP Loops – Do ... While

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

The next example defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

PHP Loops – Do ... While

```
<html>
<body>

<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<=5);
?>

</body>
</html>
```

PHP Loops – Do ... While

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

PHP Loops - For

The for loop is used when you know in advance how many times the script should run.

Syntax

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

PHP Loops - For

Parameters:

- **init**: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- **condition**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **increment**: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

PHP Loops - For

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>

</body>
</html>
```

PHP Loops - For

Output:

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

PHP Loops - Foreach

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

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

PHP Loops - Foreach

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>

</body>
</html>
```

PHP Loops - Foreach

Winner of the most impressive slide award

Output:

```
one
two
three
```

continue Statement

- Sometimes a situation arises where we want to take the control to the beginning of the loop (for example for, while, do while etc.) skipping the rest statements inside the loop which have not yet been executed.

- The keyword continue allow us to do this. When the keyword continue executed inside a loop the control automatically passes to the beginning of loop. Continue is usually associated with the if.

Continue Statement Example

```
<?php
$x=1;
echo 'List of odd numbers between 1 to 10 <br />';
while ($x<=10)
{
if (($x % 2)==0)
{
$x++;
continue;
}
else
{
echo $x.'<br />';
$x++;
}
}
```

Output

List of odd numbers between 1 to 10
1
3
5
7
9

Break Statement

- Sometimes a situation arises where we want to exit from a loop immediately without waiting to get back to the conditional statement.

- The keyword break ends execution of the current for, foreach, while, do while or switch structure. When the keyword break executed inside a loop the control automatically passes to the first statement outside the loop. A break is usually associated with the if.

Break Statement Example.

```
<?php
$array1=array(100, 1100, 200, 400, 900);
$x1=0;
$sum=0
while ($x1<=4)
{
if ($sum>1500)
{
break;
}
$sum = $sum+$array1[$x1];
$x1=$x1+1;
}
echo $sum;
?>
```

Output :

1800

Thank You