Web Application Development (WAD)

Vth Sem – BBAITM(Unit-1)

By: Binit Patel

Introduction:

PHP (Hypertext Preprocessor) was invented by "Rasmus Lerdorf" in 1994. First it was known as Personal Home Page. Later PHP becomes Hypertext Preprocessor. Since the PHP script executes on the web server and produces pure HTML as its output the PHP script is completely opaque to the requesting browser. The web server always dispatches pure HTML code to the browser which is the output of the PHP script. Since PHP script is never dispatched to a browser, the script cannot be captured via a browser's **View Source** capabilities.

PHP is a parsed language. It requires a PHP environment to execute in. every time a client browser requests a page delivered by PHP code, the web server invokes the PHP program delivers this to a PHP parser which executes the PHP statements to produce the desired HTML output. Fortunately, this is a really fast process.

Features of PHP:

Some PHP attributes are:

- ✓ PHP stands for Hypertext Preprocessor
- ✓ PHP is a Web server-side scripting language
- ✓ PHP scripts are executed at web server. Their output, which is pure HTML is returned to the requesting browser.
- ✓ PHP can connect natively to many database engines such as (MySQL, Oracle, Postgre, SQLite, and so on)
- ✓ PHP is Open Source Software (OSS)
- ✓ PHP is totally free of cost to download and use.
- ✓ PHP runs perfectly on different operating systems such as Windows, Linux, Unix and so on

- ✓ PHP is compatible with almost all web servers (Apache, IIS and so on)

What is a PHP File?

- ✓ After PHP files are processed, their output which is pure HTML is returned to the browser
- ✓ PHP filenames usually have the extension of .php, .php3 or phtml

General Structure of PHP:

PHP code is always separated from HTML code with symbols **<?php** (LESS-THAN sign followed by question mark and the word php) and **?>** (a question mark followed by a GREATER-THAN sign).

<?php indicates the start of a PHP block or code and tells the web server that all statements that follow until **?>** are PHP statements.

For e.g.:

```
<?php
        //<some valid PHP syntax>
?>
```

Comments in PHP:

Comments help in understanding the code when maintaining it at a later date.

PHP provides the following ways to set comments:

- ✓ // at the beginning of each line (i.e. a single line comment)

```
<?php
        //<some valid PHP syntax>
Echo "Hello SEMCOM";
?>
```

- ✓ /* and */ to comment out several lines (i.e. multi line comment)

```
<?php
        /*<some valid PHP syntax>
                <Multi line comment > */
echo "Hello SEMCOM";
?>
```

- ✓ # at the beginning of each line (i.e. a single line comment)

```
<?php
        #<some valid PHP syntax>
        #<line comment >
Echo "Hello SEMCOM";
?>
```

The Semicolon:

The semicolon (;) signifies the end of a PHP statement and should never be forgotten. There should be semicolon after each line of PHP code.

White Space:

White spaces between lines of PHP code embedded in HTML are ignored by the PHP interpreter.

Displaying output:

To output (i.e. send) text in a PHP script to a browser the commands **print** or **echo** can be used.

The **print** statement is used in the following way:

Print("Hello SEMCOM");

print is the command which tells the PHP interpreter what to do. This is followed by the information to be printed, which is contained within brackets and inside double quotes.

The **echo** statement is used in the following way:

echo "Hello SEMCOM";

echo is the command which tells the PHP interpreter what to do. This is followed by the information to be printed, which is contained within double quotes.

Variables:

Variables are nothing but identifiers which point to a memory location (RAM) in which data is stored. Variables in PHP are quite different from compiled languages such as C and Java. This is because PHP variables are **weakly typed**. This means that when a PHP variable is being defined their data type is not declared prior their use.

A PHP variable must be named/declared starting with the **$** character followed by a letter. The rest of the variable name can consist of a mix of both alphabets and numbers.

The following are three examples of valid names for variables.

- ✓ $city
- ✓ $address2
- ✓ $age_30

The _ (underscore) character can also be used in variable names. **Space** is a **character that cannot be** used when naming a PHP variable.

The following characters **are not allowed** in a variable name and cause errors if used:

- ✓ * (asterix)
- ✓ + (plus)
- ✓ # (hash)

- ✓ @ (at the rate)
- ✓ - (minus)
- ✓ & (ampersand)

Once a variable is named, it is empty until assigned a value. To assign a value to variable

$city = "V V Nagar";

Everything between double quotes will be assigned to the variable named city. The named variable is on the **left** side of = (i.e. the assignment operator) and the **value** to be held by the variable is on its **right**.

Similarly, numeric values can be assigned.

$age = 24;

Here, **$age** is assigned a value of **24**. The only difference is that the value 24 is passed **without** quotation marks.

Data Types:

PHP has several types of variables. All hold a specific class of information, **except one**.

The PHP variable types are:

- ✓ **String** – Strings are a sequence of characters that are always internally NULL terminated. String variable hold characters. Usually a set of characters such as "S", "SEMCOM", "PHP is my favourite language" and so on. Strings can be as short or as long as desired. There is **no limit** to size.
- ✓ **Integer** – Integer variables hold whole numbers, either positive or negative such as 1, -50, 22937932 and so on. There is a maximum limit to the size of integers. Any number lower than -2147483647 and higher than 2147483647 are automatically converted to floats as this data type can hold a much larger range of values.
- ✓ **Float** – Float variables hold fractional numbers as well as very high integer numbers such as 3.5, 1.0000001, 21474836470000 and so on.

- ✓ **Boolean –** Boolean variables hold **TRUE** or **FALSE**. Behind the scenes these are just integers. PHP considers the number **0** to be **FALSE** and **everything else** to be **TRUE**.
- ✓ **Array –** Arrays are a special variable type, which hold multiple values. They are a great way of storing variables that are related such as colors, days of the week, and members of a team or items in a shopping cart.
- ✓ **Object –** Object variables are complex variables that have their own functions (i.e. methods) for accessing and/or manipulating their content.
- ✓ **Resource –** Resource variables are variables that hold anything that is **not** PHP data. This could be picture data loaded from a file, the result of an SQL query and so on.

PHP Superglobals:

Superglobals are variables that are automatically available throughout all program code, in all scopes. These variables require no declaration they can simply be accessed.

- ✓ Useful information about the environment
- ✓ Allow access to HTML form variables or parameters
- ✓ Access to cookies stored on a client
- ✓ Keeping track of sessions and file uploads

Major superglobals are as follows:

$_GET: $_GET is used for data passed using HTTP GET method, which includes variables passed as part of the URL

$_POST: the HTTP POST method is very similar to the $_GET. It is conventionally used when the content of an HTML form are going to change the values in a database.

$_REQUEST: $_REQUEST holds variable provided to the script via the GET, POST input mechanism.

Constants:

A constant, like variable, is a temporary placeholder in memory that holds a value. Unlike variables, the value of a constant never changes.

For e.g.: **PI** (3.14) or the values of midnight (12:00 am) are examples of constants applicable in a real life scenario.

When a constant is declared, the **define()** function is used and requires the name of the constant and the value that is required to be assigned to that constant. The names for constants have the same rules as variables except that they **do not** have the leading dollar sign.

Unlike variables, constants, once defined are globally accessible. Constants need not be declared again and again in each new function and PHP file.

By convention, the constant name is **always in uppercase**.

Syntax:

define("<constantName>", "<value or expression>")

for e.g.:

```
<?php
define("PI",3.14);
define("COLLEGE","SEMCOM");
?>
```

Setting & Testing Datatypes:

GET TYPE:

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

Syntax: gettype(var_name)

Return Value:

- Boolean
- Integer

- Double
- String
- Array
- Object
- Resource
- NULL

For e.g.:

```
<?php
$abc = 102;
echo gettype($abc);
echo "<br>";
$def = "SEMCOM";
echo gettype($def);
?>
```

OUTPUT:

Integer

String

SET TYPE:

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

Syntax: settype(var_name,var_type)

Return Value:

- **TRUE** on success or **FALSE** on failure

For e.g.:

```
<?php
$abc = "102";
```

settype($abc, “integer”);

$def = “100”;

settype($def, “integer”);

echo $abc + $def;

?>

OUTPUT:

202

is_int() Function:

The is_int() function is used to test whether the type of the specified variable is an integer or not.

Syntax: is_int(var_name)

Return Value:

- **TRUE** if var_name is an integer or **FALSE** otherwise

For e.g.:

<?php

$abc = 102;

If (**is_int**($abc))

{

echo “Number is Integer”;

}

?>

OUTPUT:

Number is Integer

is_numeric() Function:

The is_numeric() function is used to test whether the type of the specified variable is numeric or not.

Syntax: is_numeric(var_name)

Return Value:

- **TRUE** if var_name is a number or a numeric string, **FALSE** otherwise

For e.g.:

```
<?php
$abc = "102";
If (is_numeric($abc))
{
echo "Number is Numeric";
}
?>
```

OUTPUT:

Number is Numeric

is_float() Function:

The is_float() function is used to test whether the type of the specified variable is float or not.

Syntax: is_float(var_name)

Return Value:

- **TRUE** if var_name is float or **FALSE** otherwise

For e.g.:

```
<?php
```

```
$abc = 102.25;

If (is_float($abc))

{

echo "Number is Float";

}

?>
```

OUTPUT:

Number is Float

is_string() Function:

The is_string() function is used to test whether the type of the specified variable is a string or not.

Syntax: is_string(var_name)

Return Value:

- **TRUE** if var_name is a string or **FALSE** otherwise

For e.g.:

```
<?php

$abc = "SEMCOM";

If (is_string ($abc))

{

echo "Variable is String";

}

?>
```

OUTPUT:

Variable is String

Operators:

Arithmetic Operators:

If one or both of the operands are string, Booleans, Nulls or Resources, they are automatically converted to their numeric equivalents before the calculation is performed.

Operator	Description
+	**Addition Operator (Adds Two Values Together)**
-	**Subtraction Operator (Subtracts One Value From Another)**
*	**Multiplication Operator (Multiplies Two Values)**
/	**Division Operator (Divides One Values From Another)**
%	**Modulus Operator (Determines The Remainder of a Division)**

For e.g.:

```
<?php
$addition = 5 + 5;
echo $addition; // Result: 10

$subtraction = 5 - 5;
echo $subtraction; // Result: 0

$multiplication = 5 * 5;
echo $multiplication; // Result: 25

$division = 5 / 5;
echo $division; // Result: 1
```

```
$modulus = 7 % 5;
echo $modulus; // Result: 2


echo 3 + 5 / 2; // Result: 5.5
?>
```

Comparison Operators:

Comparison operator determines the relationship between two operands. When both operands are strings, the comparison is performed lexicographically.

Operator	Description	Example
==	Is Equal To	3==2 (Will Return "False")
!=	Is Not Equal To	3!=2 (Will Return "True")
<>	Is Not Equal To	3<>2 (Will Return "True")
>	Is Greater Than	3>2 (Will Return "True")
<	Is Less Than	3<2 (Will Return "False")
>=	Is Greater Than or Equal To	3>=2 (Will Return "True")
<=	Is Less Than or Equal To	3<=2 (Will Return "False")

Logical Operators:

Logical operators usually work with comparison operators to determine if more than one condition is true or false at the same time.

Operator	Meaning	Description
&&	And	True If Two Statements Are True, Otherwise False
||	Or	True If One Statement or Another Is True, Otherwise False
!	Not	True If Statement Is False, False If Statement Is True

Assignment Operators:

Very simply, assignment operators assign value. We have already used the assignment operator to assign values to variables.

Operator	Description
=	Assign a Value
+=	Adds Value to An Existing Variable Value
-=	Subtracts Value From Existing Variable Value
*=	Multiplies An Existing Variable Value
/=	Divides An Existing Variable Value
%=	Modulo of An Existing Variable Value And New Value

For e.g.:

```
<?php
$variable = 8;
$variable += 3;
echo $variable; // Result: 11
?>
```

Concatenation Operator:

The concatenation operator (.) concatenates two strings. This operator works only with strings. Thus, any non-string operand is first converted to string. The following example would deliver **The Month is 10** :

```
<?php
$month = 10;
echo "The Month is " . $month;
?>
```