SEMESTER – VI - III BSc IT

18BIT61C - SCRIPTING TOOLS

Prepared by Dr.N.Thenmozhi

UNIT I: PHP Introduction: History-unique features-basic development concepts. Using variables & operators: storing data in variables-understanding PHP'S data types-using constants. Controlling program flow: if, if-else, if-else if-else, switch case, while, do while, for loop.

UNIT II: Working with string & numeric functions: using string functions, using numeric functions. Working with arrays: creating an array – Modifying array element – using arrays with forms-working with array functions-working with date and time-creating user defined functions. Working with forms: creating a simple Input form –Accessing from - combining HTML and PHP code on single page –using hidden fields to save state –redirecting the user – working with file uploads.

UNIT III: Creating Classes: Introducing classes and objects-defining and using classes-using advanced OOPs concepts-using constructors and destructors-extending classes-adjusting visibility settings-working with files and directories: reading local file-remote file-specific segments of a file-writing files-processing directories-performing other file and directory operations.

UNIT IV: Reading & Writing Files – Testing File Attributes – Managing Sessions And Using Session Variables – Destroying A Session. Storing Data in Cookies – Selecting Cookies – Removing Cookies Data – Deleting Cookies – Dealing With Date & Time.

UNIT V: Database & MySQL – Installing MySQL – Integrating PHP & MySQL – Connecting to MySQL – MySQL Queries – Dataset – Multiple Connection – Error Checking – Creating MySQL Database with PHP – MySQL Data types – MySQL Functions.

TEXT BOOKS

1. Vikram Vaswani, "PHP: A beginners guide", TMH Hill, 1stedition, 2010 (Unit-I to IV).
2. Steve Suehring, Tim Converse, Joyce Park , "PHP 6 and MySQL 6 Bible", Wiley India pvt. Ltd., Edition, 2009 (Unit – V).

REFERENCE BOOKS

1. Matt Doyle, " Beginning PHP 5.3", Wiley India pvt. Ltd, First edition, 2010.
2. Luke welling and Laura Thomson, "PHP and MySQL Web Development", 5th Edition, 2016.

18BIT61C **:** SCRIPTING TOOLS

UNIT – I

PHP Introduction: History-unique features-basic development concepts. Using variables & operators: storing data in variables-understanding PHP'S data types-using constants. Controlling program flow: if, if-else, if-else if-else, switch case, while, do while, for loop.

1. Introduction of PHP

About PHP

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- Originally called "Personal Home page Tools"
- Popular server-side scripting technology
- PHP files extensions are ".php", ".php3", or ".html" or ".phtml"
- It is open source, ie anyone may view, modify and redistribute source code, supported freely community.

History of PHP

- First version of PHP created by Rasmus Lerdorf in 1994
- PHP/PHP 2 - Personal Home page/Forms Interpreter, server side embedded scripting language. Added database support, files upload, regular expreseeions, etc., 1997
- PHP 3 - Hypertext Preprocessor added support for ODBC data source, multiple platform support, email protocols by Andi Gutmans and Zeev suraski, API, 1998
- PHP 4 – Independent Component of web server, for added efficiently(adds Zend Engine) by Zend Developer Zone, 2000
- PHP 5, Included object oriented features(adds Zend Engine II), SQLite has been bundled with PHP, 2004

Why is PHP used?

- PHP runs on different platforms :
  - Web Servers: Apache, Microsoft IIS, Caudium, Netscape Enterprise Server.
  - Operating Systems: UNIX (HP-UX,OpenBSD,Solaris,Linux), Mac OSX, Windows NT/98/2000/XP/2003
  - Supported Databases: Adabas D, dBase,Empress, FilePro (read-only), Hyperwave,IBM DB2, Informix, Ingres, InterBase, FrontBase, mSQL, Direct MS-SQL, MySQL, ODBC, Oracle (OCI7 and OCI8), Ovrimos, PostgreSQL, SQLite, Solid, Sybase, Velocis,Unix dbm

- PHP is Free to download from official PHP web site : www.php.net

What is PHP Used For?

PHP is a general-purpose server-side scripting language originally designed for web development to produce dynamic web pages. PHP can interact with MySQL databases.

1.1 What is PHP?

PHP is a server side scripting language, that is **used** to develop Static websites or Dynamic websites or Web applications. **PHP** stands for Hypertext Pre-processor, that earlier stood for Personal Home Pages. PHP scripts reside between reserved PHP tags. This allows the programmer to embed PHP scripts within HTML pages. **PHP** scripts can only be interpreted on a server that has **PHP** installed.

1.2 Advantages or Features of PHP

It is most popular and frequently used worldwide scripting language, the main reason of popularity is; It is open source and very simple.

- Simple
- Faster
- Interpreted
- Open Source
- Case Sensitive
- Simplicity
- Efficiency
- Platform Independent
- Security
- Flexibility
- Familiarity
- Error Reporting
- Loosely Typed Language
- Real-Time Access Monitoring

1.3 Basic Development Concepts

How to Install PHP Server?

To install PHP, you need to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems. There are many AMP options available in the market that is given below:

- **WAMP** for Windows (W-Windows, A-Apache, M-MySQL, P-PHP)
- **LAMP** for Linux
- **MAMP** for Mac
- **SAMP** for Solaris
- **FAMP** for FreeBSD

XAMPP (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, OpenSSL, Mercury Mail etc.

Setup PHP on your Own PC

- Install a web server
- Install PHP (The official website for PHP : http://php.net/manual/en/install.php)

- Install a database, such as MySQL

The Windows server installation of PHP running IIS is much simpler than on Unix, since it involves a precompiled binary rather than a source build.

If you plan to install PHP over Windows, then here is the list of prerequisites –

- A working PHP-supported Web server. Under previous versions of PHP, IIS/PWS was the easiest choice because a module version of PHP was available for it; but PHP now has added a much wider selection of modules for Windows.
- A correctly installed PHP-supported database like MySQL or Oracle etc. (if you plan to use one)
- The PHP Windows binary distribution (download it at www.php.net/downloads.php)
- A utility to unzip files (search http://download.cnet.com for PC file compression utilities)

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible.

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

Canonical PHP tags

The most universally effective PHP tag style is

```
<?php                          ---→ Start tag
    ...
    ...  Write php code here
?>                             ---→ End tag
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags

Short or short-open tags look like this –

<?...?>

Short tags are, as one might expect, the shortest option You must do one of two things to enable PHP to recognize the tag.

- Choose the --enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

HTML script tags

HTML script tags look like this,

<script language = "PHP"> ... </script>

Commenting PHP Code

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP.

Single-line comments – they are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
  # This is a comment, and
  # This is the second line of the comment

  // This is a comment too. Each style comments only
  print "An example with single line comments";
?>
```

Multi-lines comments – They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
  /* This is a comment with multiline
     Author : Mohammad Mohtashim
     Purpose: Multiline Comments Demo
     Subject: PHP
  */

  print "An example with multi line comments";
?>
```

Multi-lines printing – Here are the examples to print multiple lines in a single print statement –

```
<?
  # First Example
  print <<<END
  This uses the "here document" syntax to output
  multiple lines with $variable interpolation. Note
  that the here document terminator must appear on a
  line with just a semicolon no extra whitespace!
  END;

  # Second Example
  print "This spans
  multiple lines. The newlines will be
  output as well";
?>
```

PHP is whitespace insensitive

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. One whitespace character is the same as many such characters.

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent.

```
$four = 2 + 2; // single spaces
$four <tab>=<tab>2<tab>+<tab>2 ; // spaces and tabs
$four =
2+
2; // multiple lines
```

PHP is case sensitive : The following example shows this :

```
<html>
  <body>

    <?php
      $capital = 67;
      print("Variable capital is $capital<br>");
      print("Variable CaPiTaL is $CaPiTaL<br>");
    ?>

  </body>
</html>
```

This will produce the following result :

Variable capital is 67
Variable CaPiTaL is

Statements are expressions terminated by semicolons

A *statement* in PHP is any expression that is followed by a semicolon (;).Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called $greeting.

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

Braces make blocks

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent :

```
if (3 == 2 + 1)
  print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1) {
  print("Good - I haven't totally");
  print("lost my mind.<br>");
}
```

Running PHP Script from Command Prompt

Run your PHP script on your command prompt. Assuming you have following content in test.php file

```
<?php
  echo "Hello PHP!!!!!";
?>
```

Now run this script as command prompt as follows :

```
$ php test.php
```

It will produce the following result :

```
Hello PHP!!!!!
```

1.4 PHP Language structure

1.4.1 The building blocks of PHP

- About variables
- How to define and access variables
- About data types
- About some of the more commonly used operators
- How to use operators to create expressions
- How to define and use constants

1.4.2 Structure of PHP

```
<?php                          ---→ Start tag
    ...
    ...  Write php code here
?>                             ---→ End tag
```

1.4.3 Variables

The main way to store information in the middle of a PHP program is by using a variable. The variable value can be changed during the program execution. Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

Variable Naming : Rules for naming a variable is

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc.
- There is no size limit for variables.

Syntax of declaring a variable is:

```
$variablename=value;
Example  $x = 5; $y = 10.6;
```

1.4.4 Data Types

PHP has a total of eight data types to construct 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 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).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

Integers : They are whole numbers, without a decimal point, like 4195. They are the simplest type, correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is . (2**31 . 1) (or .2,147,483,647).

Doubles :They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code

```
<?php
  $many = 2.2888800;
  $many_2 = 2.2111200;
  $few = $many + $many_2;
    print("$many + $many_2 = $few <br>");
?>
```

It produces the following browser output is

2.28888 + 2.21112 = 4.5

Boolean : They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so,

```
if (TRUE)
  print("This will always print<br>");
else
  print("This will never print<br>");
```

Here are the rules for determine the "truth" of any value not already of the Boolean type :

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

NULL : NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this,

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive;

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings : They are sequences of characters, like "PHP supports string operations". Following are valid examples of string.

```
$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
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?php
  $variable = "name";
  $literally = 'My $variable will not print!';
  print($literally);
  print "<br>";
  $literally = "My $variable will print!";
  print($literally);
?>
```

This will produce following result :

```
My $variable will not print!
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are

- \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 (\)

Here Document : You can assign multiple lines to a single string variable using here document.

```
<?php
  $channel =<<<_XML_
    <channel>
    <title>What's For Dinner</title>
    <link>http://menu.example.com/ </link>
    <description>Choose what to eat tonight.</description>
  </channel>
  _XML_;
    echo <<<END
  This uses the "here document" syntax to output multiple lines with variable
  interpolation. Note that the here document terminator must appear on a line with
  just a semicolon. no extra whitespace!
  END;
  print $channel;

?>
```

This will produce following result

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
What's For Dinner
http://menu.example.com/
Choose what to eat tonight.
```

1.4.5 Operator and Expressions

The combination of operands with an **operator** to produce a result is called an **expression**. Although **operators** and their operands form the basis of **expressions**, an **expression** need not contain an **operator**. In fact, an **expression** in **PHP** is defined as anything that can be used as a value.

What is Operator? Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators

- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators : 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 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

Comparison Operators

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.

Logical Operators

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	(A and B) is true.

or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero 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.

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

Conditional Operator

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.

Operator	Description	Example

? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories

All the operators we have discussed above can be categorised into following categories –

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators

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.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

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

1.4.6 Constant :

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By

convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

$x = 5;
$name = “Hello”;

define() and constant() function

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function : As indicated by the name, this function will return the value of the constant. This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. it is stored in a variable or returned by a function.

```
<?php
  define("MINSIZE", 50);

  echo MINSIZE;
  echo constant("MINSIZE"); // same thing as the previous line
?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.
Differences between constants and variables are

- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names

```
// Valid constant names
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing");

// Invalid constant names
define("2TWO",    "second thing");
define("__THREE__", "third value");
```

PHP 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 –

A few "magical" PHP constants are given below –

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).

1.4.7 Typecasting in PHP:

The meaning of type casting is to use the value of a variable with different data type. In other word typecasting is a way to utilize one data type variable into the different data type. Typecasting is the explicit conversion of data type because user explicitly defines the data type in which he wants to cast. PHP does not require or support type definition of the variable. In PHP we never define data type while declaring the variable. In PHP variables automatically decide the data type on the basis of the value assignment or context.
We can cast following data type variable in PHP

- (int), (integer) - cast to integer
- (bool), (boolean) - cast to boolean
- (float), (double), (real) - cast to float
- (string) - cast to string
- (array) - cast to array
- (object) - cast to object
- (unset) - cast to NULL (PHP 5)

Function	Purpose
`is_bool()`	Tests if a variable holds a Boolean value
`is_numeric()`	Tests if a variable holds a numeric value
`is_int()`	Tests if a variable holds an integer
`is_float()`	Tests if a variable holds a floating-point value
`is_string()`	Tests if a variable holds a string value
`is_null()`	Tests if a variable holds a NULL value
`is_array()`	Tests if a variable is an array
`is_object()`	Tests if a variable is an object

Testing the Type of a Variable

```
1: <?php
2: $testing; // declare without assigning
3: echo “is null? “.is_null($testing); // checks if null
4: echo “<br/>”;
5: $testing = 5;
6: echo “is an integer? “.is_int($testing); // checks if integer
7: echo “<br/>”;
8: $testing = “five”;
9: echo “is a string? “.is_string($testing); // checks if string
10: echo “<br/>”;
11: $testing = 5.024;
12: echo “is a double? “.is_double($testing); // checks if double
13: echo “<br/>”;
14: $testing = true;
15: echo “is boolean? “.is_bool($testing); // checks if boolean
16: echo “<br/>”;
17: $testing = array(‘apple’, ‘orange’, ‘pear’);
18: echo “is an array? “.is_array($testing); // checks if array
19: echo “<br/>”;
20: echo “is numeric? “.is_numeric($testing); // checks if is numeric
21: echo “<br/>”;
22: echo “is a resource? “.is_resource($testing); // checks if is a resource
23: echo “<br/>”;
24: echo “is an array? “.is_array($testing); // checks if is an array
25: echo “<br/>”;
26: ?>
```

Changing Type with settype()

PHP also provides the function settype(), which is used to change the type of a variable. To use settype(), you place the variable to change and the type to change it to between the parentheses and separate the elements with a comma, like this:

```
settype($variabletochange, ‘new type’);
```

Example for changing the Type of a Variable with settype()

```
1: <?php
2: $undecided = 3.14;
3: echo “is “.$undecided.” a double? “.is_double($undecided).”<br/>”; // double
4: settype($undecided, ‘string’);
5: echo “is “.$undecided.” a string? “.is_string($undecided).”<br/>”; // string
6: settype($undecided, ‘integer’);
7: echo “is “.$undecided.” an integer? “.is_integer($undecided).”<br/>”; //
integer
8: settype($undecided, ‘double’);
9: echo “is “.$undecided.” a double? “.is_double($undecided).”<br/>”; // double
10: settype($undecided, ‘bool’);
11: echo “is “.$undecided.” a boolean? “.is_bool($undecided).”<br/>”; // boolean
12: ?>
```

1.5 Controlling program flow

What is a control structure?

Code execution can be grouped into categories as shown below

- **Sequential** : this one involves executing all the codes in the order in which they have been written.
- **Decision** : this one involves making a choice given a number of options. The code executed depends on the value of the condition.

A control structure is a block of code that decides the execution path of a program depending on the value of the set condition.

1.5.1. Switching Flow

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements.

1. The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement. The general Syntax is

if (*condition*)
    code to be executed if condition is true;
else
    code to be executed if condition is false;

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

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

```
            echo "Have a nice day!";
    ?>
   </body>
</html>
```

It will produce the following result:

Have a nice weekend!

2.The ElseIf Statement

If you want to execute some code if one of the several conditions are true use the elseif statement. The Syntax is

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

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>
```

It will produce the following result –

Have a nice Weekend!

3. The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code. The syntax is

```
switch (expression){
        case label1:
                code to be executed if expression = label1;
                 break;
        case label2:
                code to be executed if expression = label2;
                break;
```

```
        default:
                code to be executed
                        if expression is different
                        from both label1 and label2;
        }
```

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the label matches then statement will execute any specified default code.

```
<html>
  <body>
        <?php
         $d = date("D");
            switch ($d){
                 case "Mon":
                                 echo "Today is Monday";
                                 break;
                            case "Tue":
                                  echo "Today is Tuesday";
                                   break;
                           case "Wed":
                                  echo "Today is Wednesday";
                                  break;
                              case "Thu":
                                  echo "Today is Thursday";
                                   break;
                              case "Fri":
                                   echo "Today is Friday";
                                  break;
                            case "Sat":
                                   echo "Today is Saturday";
                                  break;
                             case "Sun":
                                  echo "Today is Sunday";
                                  break;
                              default:
                                   echo "Wonder which day is this ?";
                }
        ?>
    </body>
</html>
```

It will produce the following result

Today is Monday

1.5.2 PHP - Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for –** loops through a block of code a specified number of times.
- **while –** loops through a block of code if and as long as a specified condition is true.
- **do...while –** loops through a block of code once, and then repeats the loop as long as a special condition is true.
- **foreach –** loops through a block of code for each element in an array.
- c**ontinue** and **break** keywords are used to control the loops execution.

1. The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

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

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop.

```
<html>
  <body>
    <?php
      $a = 0;
      $b = 0;
       for( $i = 0; $i<5; $i++ ) {
         $a += 10;
          $b += 5;
      }
      echo ("At the end of the loop a = $a and b = $b" );
    ?>
  </body>
</html>
```

This will produce the following result:

At the end of the loop a = 50 and b = 25

2. The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.
If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false. The general syntax is

```
while (condition) {
```

code to be executed;
}

These example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
  <body>
    <?php
      $i = 0;
      $num = 50;

      while( $i < 10) {
        $num--;
        $i++;
      }
        echo ("Loop stopped at i = $i and num = $num" );
    ?>
    </body>
</html>
```

This will produce the following result:

Loop stopped at i = 10 and num = 40

3. The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true. The general syntax is

```
do {
        code to be executed;
}
while (condition);
```

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 :

```
<html>
  <body>
    <?php
      $i = 0;
      $num = 0;
      do {
        $i++;
      }while( $i < 10 );
      echo("Loop stopped at i = $i" );
    ?>
   </body>
</html>
```

This will produce the following result :

Loop stopped at i = 10

4. The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed. The general syntax is

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

The following example to list out the values of an array.

```
<html>
  <body>
      <?php
      $array = array( 1, 2, 3, 4, 5);
       foreach( $array as $value ) {
          echo "Value is $value <br />";
       }
    ?>
   </body>
</html>
```

This will produce the following result :

Value is 1
Value is 2
Value is 3
Value is 4
Value is 5

5 .The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
  <body>
      <?php
          $i = 0;
         while( $i < 10) {
                  $i++;
                   if( $i == 3 )break;
          }
        echo ("Loop stopped at i = $i" );
       ?>
```

```
    </body>
</html>
```

This will produce the following result:

Loop stopped at i = 3

6. The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
  <body>
      <?php
        $array = array( 1, 2, 3, 4, 5);
          foreach( $array as $value ) {
                if( $value == 3 )continue;
                                echo "Value is $value <br />";
          }
    ?>
    </body>
</html>
```

This will produce the following result –

Value is 1
Value is 2
Value is 4
Value is 5

1.5.3 Code Blocks and Browser Output

You have discovered that you can present distinct output to the user according to a decision-making process you can control with if and switch statements. This section combines these two techniques. Imagine a script that outputs a table of values only when a variable is set to the Boolean value true. The following example shows a simplified HTML table constructed with the code block of if statement.

A Code Block containing multiple echo Statements

```
1: <?php
2: $display_prices = true;
3: if ($display_prices) {
4: echo “<table border=\”1\”>\n”;
5: echo “<tr><td colspan=\”3\”>”;
```

```
6: echo “today’s prices in dollars”;
7: echo “</td></tr>”;
8: echo “<tr><td>\$14.00</td><td>\$32.00</td><td>\$71.00</td></tr>\n”;
9: echo “</table>”;
10: }
11: ?>
```

In line 8, note the dollar sign, which when meant literally and not as part of a variable declaration, must be escaped with a backslash for it to be interpreted as the dollar sign character. The Output is

If the value of $display prices is set to true in line 2, the table is printed. For the sake of readability, we split the output into multiple echo() statements, and once again use the backslash to escape any quotation marks used in the HTML output. Put these lines into a text file called testmultiecho.php and place this file in your web server document root. When you access this script through your web browser, you will get the result of code block.

There’s nothing wrong with the way this is coded, but you can save yourself some typing by simply slipping back into HTML mode within the code block.

Returning to HTML Mode Within a Code Block

```
1: <?php
2: $display_prices = true;
3: if ($display_prices) {
4: ?>
5: <table border=”1”>
6: <tr><td colspan=”3”>today’s prices in dollars</td></tr>
7: <tr><td>$14.00</td><td>$32.00</td><td>$71.00</td></tr>
8: </table>
9: <?php
10: }
11: ?>
```

The important thing to note here is that the shift to HTML mode on line 4 occurs only if the condition of if statement is fulfilled. This can save you the bother of escaping quotation marks and wrapping our output in echo() statements. This approach might, however, affect the readability of the code in the long run, especially if the script grows larger.