Unit-4

Open Source Programming Languages: PHP- Introduction - programming in web environment, variables, constants, data types, operators, statements, functions, arrays and OOP - string manipulation and regular expression.

Perl: Perl backgrounder, Perl overview, Perl parsing rules, variables and data -statements and control structures, subroutines, packages, and modules- working with files and data manipulation.

PHP- Introduction

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

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

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

```
<html>

  <head>
    <title>Hello World</title>
  </head>

  <body>
    <?php echo "Hello, World!";?>
  </body>

</html>
```

It will produce following result −

Hello, World!

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ATE are recognised by the PHP Parser.

```
<?php PHP code goes here ?>

<?    PHP code goes here ?>

<script language = "php"> PHP code goes here </script>
```

A most common tag is the <?php...?> and we will also use the same tag in our tutorial.

From the next chapter we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

PHP - Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** − PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here − https://httpd.apache.org/download.cgi
- **Database** − PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here − https://www.mysql.com/downloads/
- **PHP Parser** − In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser. This tutorial will guide you how to install PHP parser on your computer.

PHP Parser Installation

Before you proceed it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

http://127.0.0.1/info.php

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

This section will guide you to install and configure PHP over the following four platforms −

- PHP Installation on Linux or Unix with Apache
- PHP Installation on Mac OS X with Apache
- PHP Installation on Windows NT/2000/XP with IIS
- PHP Installation on Windows NT/2000/XP with Apache

Apache Configuration

If you are using Apache as a Web Server then this section will guide you to edit Apache Configuration Files.

PHP.INI File Configuration

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

Windows IIS Configuration

To configure IIS on your Windows machine you can refer your IIS Reference Manual shipped along with IIS.

Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP'. There are four ways to do this −

Canonical PHP tags

The most universally effective PHP tag style is −

<?php...?>

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 tags −

- 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.

ASP-style tags

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this −

<%...%>

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

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 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";
?>
```

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";
?>
```

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>=<tab2<tab>+<tab>2 ; // spaces and tabs
$four =
2+
2; // multiple lines
```

PHP is case sensitive

Yeah it is true that PHP is a case sensitive language. Try out following example −

```
<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

Yes you can 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!!!!!
```

Hope now you have basic knowledge of PHP Syntax.

PHP - Variable Types

The main way to store information in the middle of a PHP program is by using a variable.

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.

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 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.

We will explain only simple data type in this chapters. Array and Objects will be explained separately.

Integers

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

```
$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 −

```
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>");
```

Interpreting other types as Booleans

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; you could just as well have typed −

```
$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!

```
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
```

Variable Scope

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 −

- Local variables
- Function parameters
- Global variables
- Static variables

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.

PHP - Constants Types

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.

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.

constant() example

```
<?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");
define("__THREE__", "third value");

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

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).

PHP - Operator Types

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

Lets have a look on all operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by PHP language −

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

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

There are following comparison operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

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

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then −

Show Examples

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

There are following assignment operators supported by PHP language −

Show Examples

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 −

Show Examples

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

PHP - Decision Making

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 −

if...else statement − use this statement if you want to execute a set of code when a condition is true and another if the condition is not true

- **elseif statement** − is used with the if...else statement to execute a set of code if **one** of the several condition is true
- **switch statement** − is used 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 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.

Syntax

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

Example

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!
```

The ElseIf Statement

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

Syntax

```
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;
```

Example

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!
```

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.

Syntax

```
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;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable 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 lable 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
```

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.

We will discuss about **continue** and **break** keywords used to control the loops execution.

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.

Syntax

```
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.

Example

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
```

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.

Syntax

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

Example

This 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
```

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.

Syntax

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

Example

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
```

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.

Syntax

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

Example

Try out 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
```

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.

Example

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
```

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.

Example

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
```

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** − An array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array** − An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** − An array containing one or more arrays and values are accessed using multiple indices

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
  <body>

    <?php
      /* First method to create array. */
      $numbers = array( 1, 2, 3, 4, 5);

      foreach( $numbers as $value ) {
        echo "Value is $value <br />";
      }

      /* Second method to create array. */
      $numbers[0] = "one";
      $numbers[1] = "two";
      $numbers[2] = "three";
      $numbers[3] = "four";
      $numbers[4] = "five";

      foreach( $numbers 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
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Example

```
<html>
   <body>
      
      <?php
         /* First method to associate create array. */
         $salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);
         
         echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
         echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
         echo "Salary of zara is ".  $salaries['zara']. "<br />";
         
         /* Second method to create array. */
         $salaries['mohammad'] = "high";
         $salaries['qadir'] = "medium";
         $salaries['zara'] = "low";
         
         echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
         echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
         echo "Salary of zara is ".  $salaries['zara']. "<br />";
      ?>

   </body>
</html>
```

This will produce the following result −

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects −

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
   <body>

      <?php
         $marks = array(
            "mohammad" => array (
               "physics" => 35,
               "maths" => 30,
               "chemistry" => 39
            ),

            "qadir" => array (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
            ),

            "zara" => array (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
            )
         );

         /* Accessing multi-dimensional array values */
         echo "Marks for mohammad in physics : " ;
         echo $marks['mohammad']['physics'] . "<br />";

         echo "Marks for qadir in maths : ";
         echo $marks['qadir']['maths'] . "<br />";

         echo "Marks for zara in chemistry : " ;
         echo $marks['zara']['chemistry'] . "<br />";
      ?>

   </body>
</html>
```

This will produce the following result −

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

PHP - Strings

They are sequences of characters, like "PHP supports string operations".

Built-in string functions is given in function reference PHP String Functions

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!\\n';

  print($literally);
  print "<br />";

  $literally = "My $variable will print!\\n";

  print($literally);
```

```
?>
```

This will produce the following result −

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

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 (\)

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```
<?php
  $string1="Hello World";
  $string2="1234";

  echo $string1 . " " . $string2;
?>
```

This will produce the following result −

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!" −

```
<?php
  echo strlen("Hello world!");
?>
```

This will produce the following result −

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string −

```
<?php
  echo strpos("Hello world!","world");
?>
```

This will produce the following result −

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Perl – Introduction

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

What is Perl?

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**.
- It is used for mission critical projects in the public and private sectors.

- Perl is an *Open Source* software, licensed under its *Artistic License*, or the *GNU General Public License (GPL)*.
- Perl was created by Larry Wall.
- Perl 1.0 was released to usenet's alt.comp.sources in 1987.
- At the time of writing this tutorial, the latest version of perl was 5.16.2.
- Perl is listed in the *Oxford English Dictionary*.

PC Magazine announced Perl as the finalist for its 1998 Technical Excellence Award in the Development Tool category.

Perl Features

- Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
- Perls database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl is Y2K compliant.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG.
- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).
- The Perl interpreter can be embedded into other systems.

Perl and the Web

- Perl used to be the most popular web programming language due to its text manipulation capabilities and rapid development cycle.
- Perl is widely known as "the duct-tape of the Internet".
- Perl can handle encrypted Web data, including e-commerce transactions.
- Perl can be embedded into web servers to speed up processing by as much as 2000%.
- Perl's mod_perl allows the Apache web server to embed a Perl interpreter.
- Perl's DBI package makes web-database integration easy.

Perl is Interpreted

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non portable executable program.

Traditional compilers convert programs into machine language. When you run a Perl program, it's first compiled into a byte code, which is then converted (as the program runs) into machine instructions. So it is not quite the same as shells, or Tcl, which are **strictly** interpreted without an intermediate representation.

It is also not like most versions of C or C++, which are compiled directly into a machine dependent format. It is somewhere in between, along with *Python* and *awk* and Emacs .elc files.

Lexing and Parsing

Perl programmers spend a lot of time munging data: reading it in, transforming it, and writing out the results. Perl's great at this ad hoc text transformation, with all sorts of string manipulation tools, including regular expressions.

Regular expressions will only get you so far: witness the repeated advice that you cannot parse HTML or XML with regular expressions themselves. When Perl's builtin text processing tools aren't enough, you have to turn to something more powerful.

That something is *parsing*.

An Overview of Lexing and Parsing

For a more formal discussion of what exactly lexing and parsing are, start with Wikipedia's definitions: Lexing and Parsing.

Unfortunately, when people use the word parsing, they sometimes include the idea of lexing. Other times they don't. This can cause confusion, but I'll try to keep them clear. Such situations arise with other words, and our minds usually resolve the specific meaning intended by analysing the context in which the word is used. So, keep your mind in mind.

The lex phase and the parse phase can be combined into a single process, but I advocate always keeping them separate. Trust me for a moment; I'll explain shortly. If you're having trouble keeping the ideas separate, note that the phases very conveniently run in alphabetical order: first we lex, and then we parse.

But Why Study Lexing and Parsing?

There are many situations where the only path to a solution requires a lexer and a parser:

1. *Running a program*

   This is trivial to understand, but not to implement. In order to run a program we need to set up a range of pre-conditions:

   - Define the language, perhaps called Perl
   - Write a compiler (combined lexer and parser) for that language's grammar

    - Write a program in that language
    - *Lex and parse* the source code

      After all, it must be syntactically correct before we run it. If not, we display syntax errors. The real point of this step is to determine the programmer's *intention*, that is, the reason for writing the code. We don't *run* the code in this step, but we do get output. How do we do that?

    - Run the code

      Then we can gaze at the output which, hopefully, is correct. Otherwise, perhaps, we must find and fix logic errors.

2. *Rendering a web page of HTML + content*

   The steps are identical to those of the first example, with HTML replacing Perl, although I can't bring myself to call writing HTML writing a program.

   This time, we're asking: What is the web page designer's *intention*. What would they like to render and how? Of course, syntax checking is far looser that with a programming language, but must still be undertaken. For instance, here's an example of clearly-corrupt HTML which can be parsed by Marpa:

```
<title>Short</title><p>Text</head><head>
```

   See Marpa::HTML for more details. So far, I have used Marpa::R2 in all my work, which does not involve HTML.

3. Rendering an image, perhaps in SVG

   Consider this file, written in the DOT language, as used by the Graphviz graph visualizer (*teamwork.dot*):

```
digraph Perl
{
graph [ rankdir="LR" ]
node  [ fontsize="12pt" shape="rectangle" style="filled, solid" ]
edge  [ color="grey" ]
"Teamwork" [ fillcolor="yellow" ]
"Victory"  [ fillcolor="red" ]
"Teamwork" -> "Victory" [ label="is the key to" ]
}
```

   Given this "program", a renderer give effects to the author's *intention* by rendering an image:

What's required to do that? As above, *lex*, *parse*, *render*. Using Graphviz's dot command to carry out these tasks, we would run:

```
shell> dot -Tsvg teamwork.dot > teamwork.svg
```

Note: Files used in these examples can be downloaded from http://savage.net.au/Ron/html/graphviz2.marpa/teamwork.tgz.

The link to the DOT language points to a definition of DOT's syntax, written in a somewhat casual version of BNF: Backus-Naur Form. This is significant, as it's usually straight-forward to translate a BNF description of a language into code within a lexer and parser.

4. Rendering that same image, using a different language in the input file

   Suppose that you decide that the Graphviz language is too complex, and hence you write a wrapper around it, so end users can code in a simplified version of that language. This actually happened, with the original effort available in the now-obsolete Perl module Graph::Easy. Tels, the author, devised his own very clever little language, which he called Graph::Easy.

   When I took over maintenance of Graph::Easy, I found the code too complex to read, let alone work on, so I wrote another implementation of the lexer and parser, released as Graph::Easy::Marpa. I'll have much more to say about that in another article. For now, let's discuss the previous graph rewritten in Graph::Easy (*teamwork.easy*):

```
graph {rankdir: LR}

node {fontsize: 12pt; shape: rectangle; style: filled, solid}

edge {color: grey}

[Teamwork]{fillcolor: yellow}

-> {label: is the key to}

[Victory]{fillcolor: red}
```

   That's simpler for sure, but how does Graph::Easy::Marpa work? Easy: *lex*, *parse*, render. More samples of Graph::Easy::Marpa's work are my Graph::Easy::Marpa examples.

It should be clear by now that lexing and parsing are in fact widespread, although they often operate out of sight, with only their rendered output visible to the average programmer, user, or web surfer.

All of these problems have in common a complex but well-structured source text format, with a bit of hand-waving over the tacky details available to authors of documents in HTML. In each case, it is the responsibility of the programmer writing the lexer and parser to honour the intention of the original text's author. We can only do that by recognizing each token in the input as a discrete unit of meaning (where a word such as `print` *means* to output something of the author's choosing), and by bringing that meaning to fruition (for `print`, to make the output visible on a device).

With all that I can safely claim that the ubiquity and success of lexing and parsing justify their recognition as vital constituents in the world of software engineering. Why study them indeed!