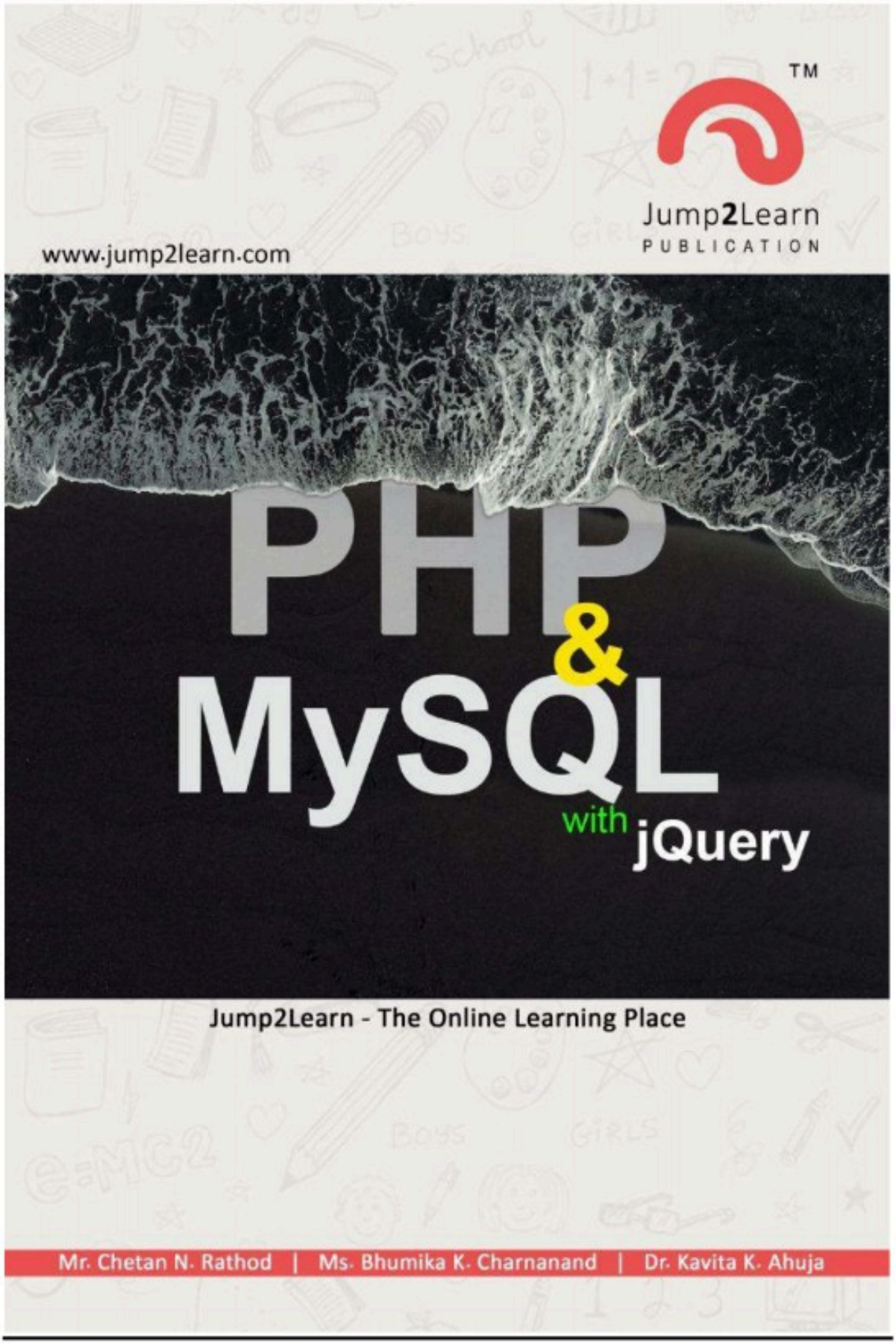

TM
Jump2Learn
PUBLICATION
www.jump2learn.com
PHP
&
MySQL
with jQuery
Jump2Learn - The Online Learning Place
Mr. Chetan N. Rathod | Ms. Bhumika K. Charnanand | Dr. Kavita K. Ahuja

Unit - 1

Introduction

- Features of PHP
- Web Communication fundamentals
  - Request-Response
- Parsing of PHP code
- Embedding PHP and HTML
- Executing PHP and viewing in Browser
- Data types
- Operators
- PHP variables: Static and Global variables
- Comments in PHP
- Control Statements
  - Condition Statements
    - If...Else
    - Switch
    - '?' Operator
  - Loops
    - While
    - Break Statement
    - Continue
    - Do...While
    - For
    - For each
- Exit, Die, Return
- Arrays in PHP

INTRODUCTION

PHP is open source software (OSS) and a powerful server-side scripting language for creating dynamic and interactive websites. PHP is the widely used, free and efficient alternative to competitors such as Microsoft's ASP. PHP is perfectly suited for Web development and can be embedded directly into the HTML code.

PHP's father was **Rasmus Lerdorf.** - Software engineer, Apache team member, and international man of mystery—is the creator and original driving force behind PHP. The first part of PHP was developed for his personal use in late 1994. By the middle of 1997, PHP was being used on approximately 50,000 sites worldwide.

The PHP syntax is very similar to Perl and C language. PHP is often used together with Apache (Web server) on various operating systems. It also supports ISAPI and can be used with IIS on windows. A PHP file may contain text, HTML tags and scripts. Scripts in a PHP file are executed on the server.

PHP stands for: **Hyper-text Preprocessor.** PHP is also known as Personal Home Page.

PHP is a sever-side scripting language, like ASP that's why PHP scripts are executed on the server. PHP supports various built-in functions for fast development. Source-code of PHP is not visible by client('View Source' in browsers does not display the PHP code)

PHP supports many databases(MySQL, Informix, Oracle, Sybase, Solid, PostgerSQL, Generic ODBC, etc)

NOTE:

PHP is FREE to download from the official PHP resource: www.php.net

PHP File:

- Structurally similar to C/C++
- PHP files may contain text, HTML tags and scripts.
- PHP Supports procedural and object-oriented paradigm (to some degree)
- All PHP statements end with a semi-colon
- Each PHP script must be enclosed in the reserved PHP tag(<?PHP....?>)
- PHP files are returned to the browser as plain HTML.
- PHP files have a file extension: ".php", ".php3" or ".phtml"

MySQL:

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

- MySQL is free to download and use

Apache:

- Apache is a public domain Web server developed by a loosely knit group of programmers.
- Public domain refers to any program that is not copyrighted. Public-domain software is free and can be used without restrictions.
- The first version of Apache, based on the NCSA httpd Web server, was developed in 1995. Because it was developed from existing NCSA code plus various patches, it was called a patchy server - hence the name Apache Server.
- By excellent performance, and low price - free, Apache has become the world's most popular Web server.
- Core development of the Apache Web server is performed by a group of about 20 volunteer programmers, called the Apache Group.
- The original version of Apache was written for UNIX, but there are now versions that run under OS/2, Windows and other platforms.
- Apache has been shown to be substantially faster, more stable, and more feature-full than many other web servers.
- Apache is run on over 25 million Internet servers.
- Without support of an internet it works properly.

What we need to Run PHP?

- To get access to a web server with PHP support, you can:
- Install Apache (or IIS) on your own server,
- Install PHP on a windows or Linux machine
- Install MySQL on a windows or Linux machine

Note: Instead of all these we can install a WAMP or LAMP.

WAMP: It is for Windows, install APACHE, MYSQL and PHP

LAMP: It is for Linux, install APACHE, MYSQL and PHP

XAMP: It is for Windows and Linux, install APACHE, MYSQL and PHP

FEATURES OF PHP:

- **OPEN SOURCE :**

  The PHP package is completely free. It is licensed under the GNU/GPL which allows you to use the software for any purpose, commercial or otherwise.

- **VARIABLES, ARRAYS, ASSOCIATIVE ARRAYS :**

  PHP supports typed variables, arrays and even Perl-like associative arrays.

  These can all be passed from one web page to another using either GET or POST method forms.

- **CONDITIONALS, WHILE LOOPS**

  PHP supports a full-featured C-like scripting language. You can have if/then/elseif/else/endif conditions as well as while loops and switch/case statements to guide the logical flow of how the html page should be displayed.

- **EXTENDED REGULAR EXPRESSIONS:**

  Regular expressions are heavily used for pattern matching, pattern substitutions and general string manipulation. PHP supports all common regular expression operations.

- **STANDARD CGI, FASTCGI AND APACHE MODULE SUPPORT:**

  As a standard CGI program, PHP can be installed on any UNIX machine running any UNIX web server. With support for the new FastCGI standard, PHP can take advantage of the speed improvements gained through this mechanism. As an Apache module, PHP becomes an extremely powerful and lightning fast alternative to CGI programming.

- **ACCESS LOGGING:**

  With the access logging capabilities of PHP, users can maintain their own hit counting and logging. It does not use the system's central access log files in any way, and it provides real-time access monitoring. The Log Viewer Script provides a quick summary of the accesses to a set of pages owned by an individual user. In addition to that, the package can be configured to generate a footer on every page which shows access information.

- **ACCESS CONTROL:**

  A built-in web-based configuration screen handles access control configuration. It is possible to create rules for all or some web pages owned by a certain person which place various restrictions on who can view these pages and how they will be viewed. Pages can be password protected, completely restricted, logging disabled and more based on the client's domain, browser, e-mail address or even the referring document.

- **FILE UPLOAD SUPPORT :**

  File Upload allows users to upload files to a web server. PHP provides the actual Mime decoding to make this work and also provides the additional framework to do something useful with the uploaded file once it has been received.

- **HTTP-BASED AUTHENTICATION CONTROL :**

  PHP can be used to create customized HTTP-based authentication mechanisms for the Apache web server.

- **RAW HTTP HEADER CONTROL:**

  The ability to have web pages send customized raw HTTP headers based on some condition is important for high-level web site design. A frequent use is to send a Location: URL header to redirect the calling client to some other URL. It can also be used to turn off caching or manipulate the last update header of pages.

- **ISP "SAFE MODE" SUPPORT :**

  PHP supports an unique "Safe Mode" which makes it safe to have multiple users run PHP scripts on the same server.

WEB COMMUNICATION FUNDAMENTALS:

REQUEST-RESPONSE:

request–response is one of the basic methods use to communicate with each other in a network, in which the first computer sends a request for some data and the second responds to the request. It is a message exchange pattern in which a requestor sends a request message to a replier system, which receives and processes the request, ultimately returning a message in response.

HOW IT WORKS

- A user opens his browser, types in a URL, and presses Enter.
- When a user presses Enter, the browser makes a request for that URL.
- The request hits the router. The router maps the URL to the correct controller and action to handle the request.
- The action receives the request and passes it on to the view.

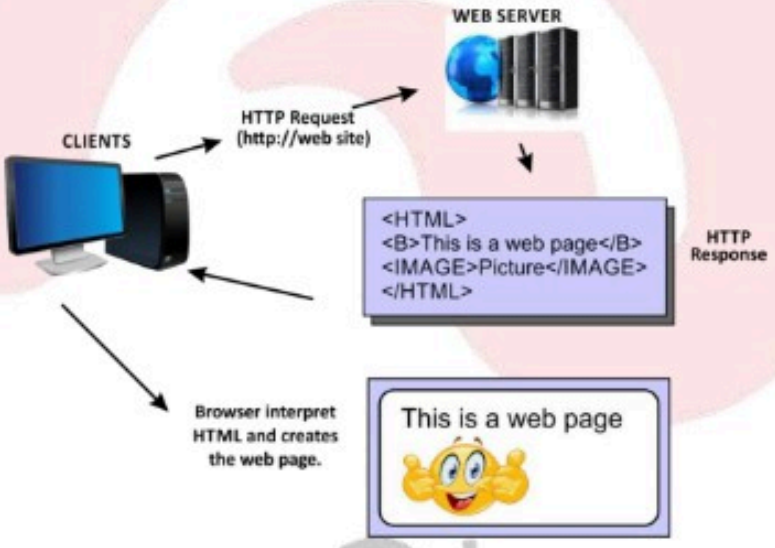
- The view renders the page as HTML.
- The controller sends the HTML back to the browser. The page loads and the user sees it.

PARSING OF PHP CODE:

What happens to HTML Pages?

When a request for a page comes from the browser, the web server performs three steps:

- Read the request from the browser.
- Find the page in the server.
- Send the page back across the Internet (or intranet) to the browser.



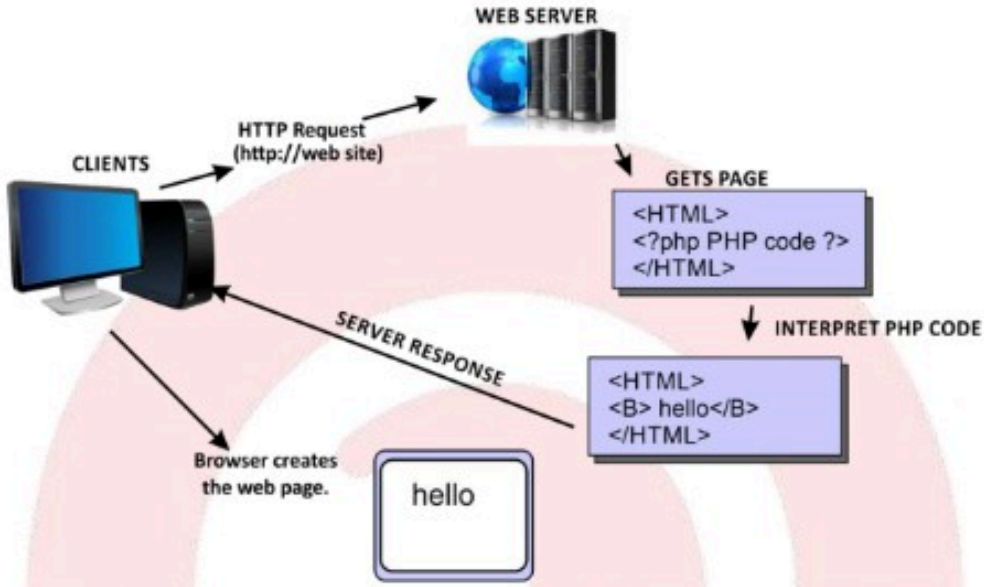
What happens to php pages?

Instead of throwing a static HTML page out to the user, the server take some actions according to our PHP code:

The PHP will make some decisions and create a page that is appropriate for the exact situation.

When using PHP the server actions are as follows:

1. Read the request from the browser.
2. Find the page in the server.
3. Perform any instructions provided in PHP to modify the page.
4. Send the page back across the Internet to the browser(In HTML form)



When a browser calls a PHP program, it first searches through the entire code line by line to locate all PHP sections (those encased in the < ?php and ? > tags), and it then processes them one at a time.

To the server, all PHP code is treated as one line.After the PHP code has been parsed accordingly, the server goes back and up the remaining HTML and spits it out to the browser, PHP sections included.

Example:

PHP can easily understand the following statement:


```
echo "I'll call you back.";
```

The statement instructs PHP to output the sequence of characters I 'll call you back. to the browser.

It knows where the start and end of the sequence is because the text is surrounded in double quotation marks.

While single quotation marks are an entirely valid way to delimit a string, PHP would become confused with the following statement:

```
echo 'I'll call you back.';
```

In fact, PHP would display a scary error message:

Parse error: syntax error, unexpected T_STRING, expecting ',' or ';' in C:\
Program Files\Apache Software Foundation\Apache2.2\htdocs\firstprog.php on line.

EMBEDDING PHP AND HTML

PHP is embedded within HTML.In other words, PHP code is a part of HTML page. that escape into PHP mode only Insert PHP tag inside HTML file (with .php extension)

- **XML Style**
  <?php PHP statement; ?>
- **Short Style (Need to be enabled)**
  <? statement; ?>
- **Script Style**
  <SCRIPT LANGUAGE='php'> PHP statement;
  </SCRIPT>
- **ASP Style (Need to be enabled)**
  <% PHP statement; %>

PHP is an HTML-embedded server-side scripting language. Much of its syntax is borrowed from C, Java and Perl. The goal of the language is to allow web developers to write dynamically generated pages quickly. In an HTML page, PHP code is enclosed within special PHP tags. When a visitor opens the page, the server processes the PHP code and then sends the output to the visitor's browser.

```
<html>
    < head >
        < title > My First PHP Program < /title >
    < /head >
        < body >
            < ?php
                echo " < h1 > I am Chetan. < /h1 > ";
                echo " < h1 > I am Fine. < /h1 > ";
            ? >
        < /body >
< /html >
```

The echo statement basically outputs whatever it ' s told to the browser, whether it be HTML code, variable values, or plaintext.

```
echo " < h1 > I am Chetan. < /h1 > ";
echo " < h1 > I am Fine. < /h1 > ";
```

EMBEDDING MULTIPLE CODE BLOCKS

We can add PHP code as number of time in our HTML code.

All PHP codes are related in single HTML page.

Example:

```
<html>
    <head>
        <title><?php echo "Welcome to Jump2Learn!";?></title>
    </head>
        <body>
            <?php
                $date = "July 26, 2020";
            ?>
                <p>Today's date is <?php echo $date;?></p>
        </body>
</html>
```

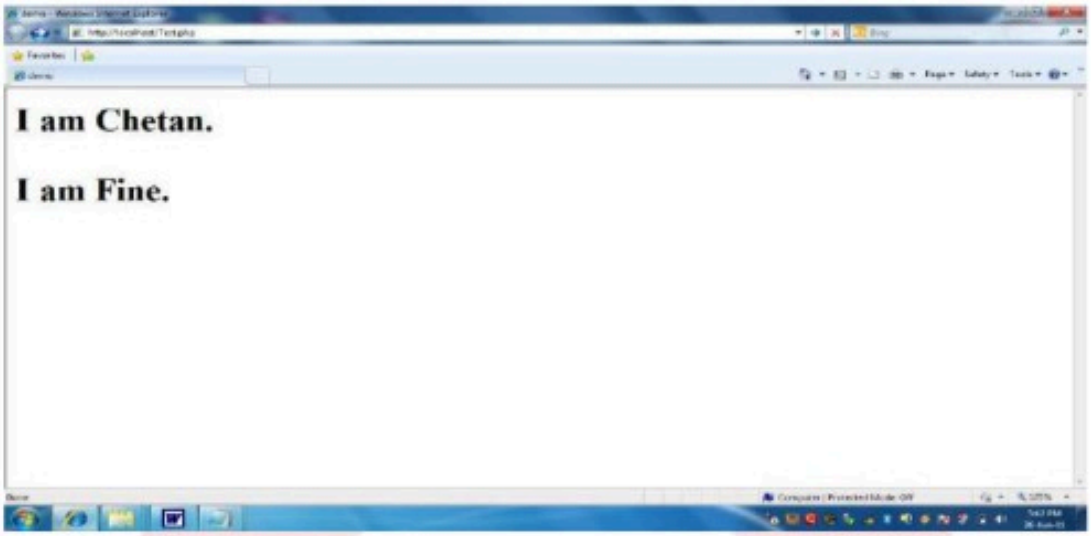

EXECUTING PHP AND VIEWING IN BROWSER

1. First Write the Code in Notepad.

```
<html>
    < head >
        < title > My First PHP Program < /title >
    < /head >
        < body >
            < ?php
                echo " < h1 > I am Chetan. < /h1 > ";
                echo " < h1 > I am Fine. < /h1 > ";
            ? >
        < /body >
< /html >
```

2. Save this file with .php extension on the PHP's works folder in my case Folder is : c:\wamp\www\
   I save file with the name Test.php
3. Open The Internet Explorer Browser
4. In URL bar write http://localhost/Test.php

YOU WILL SEE FOLLOWING PAGE IN BROWSER:



OUTPUT FUNCTION

There are number of function to displaying data to the browser use for displaying dynamic and static data.

PRINT() STATEMENT:

The print() statement outputs data passed to it .

Syntax:

```
int print(argument)
```

ALL OF THE FOLLOWING ARE POSSIBLE PRINT() STATEMENTS:

1.
```
<?php
    print("<p>I love using PHP.</p>");
?>
```

2.
```
<?php
    $a = "PHP";
        print "<p>I love using $a.</p>";
?>
```

3.
```
<?php
    print "<p>I love
        using PHP.</p>";
?>
```

All these statements produce the same output:

I love using PHP.

PRINTF() STATEMENT:

The printf() statement is perfect when you want to output text as well as the variable together.

Syntax:

integer printf(string format [, mixed args])

Example:

printf("He get %d marks in PHP.", 60);

Output:

He get 60 marks in PHP.

In this example, %d is a placeholder known as a type specifier. The %d indicates an integer value will be placed in that position. When the printf() statement executes, 60 will be inserted into the placeholder.

TYPE DESCRIPTION:

- %b Argument considered an integer; presented as a binary number
- %c Argument considered an integer; presented as a character corresponding to that ASCII value
- %d Argument considered an integer; presented as a signed decimal number
- %f Argument considered a floating-point number; presented as a floating-point number
- %o Argument considered an integer; presented as an octal number
- %s Argument considered a string; presented as a string
- %u Argument considered an integer; presented as an unsigned decimal number
- %x Argument considered an integer; presented as a lowercase hexadecimal number
- %X Argument considered an integer; presented as an uppercase hexadecimal number

```
$mark=60;
$sub="PHP";
    printf(" He get %d marks in %S", $mark, $sub);
```

SPRINTF() STATEMENT:

The sprint() statement is functionally same as to printf() except that the output is assigned to a string rather than displayed to the browser.

Syntax:

string 13print(string format [, mixed arguments])

Example:

$cost = 13print("Rs %.2f", 43.2); // $cost = Rs 43.20

ECHO STATEMENT:

PHP command echo is use to outputting text to the web browser to output a string we use echo statement. We can place either a string variable or we can use quotes, like we do below, to create a string that the echo function will output.

Example:

```
<?php
        $myString = "Hello!";
                echo $myString;
                echo "<h5>I love using PHP!</h5>";
?>
```

Output:

Hello!
I love using PHP!

In the above example we output "Hello!" without any problem. The text we are outputting is being sent to the user in the form of a web page, so it is important that we use proper HTML syntax!
In our second echo statement we use echo to write a valid Header 5 HTML statement. To do this we simply put the <h5> tag with echo.
You must be careful when using HTML code or any other string that includes quotes. The echo function uses quotes to define the beginning and end of the string, so you must use one of the following strategy if your string contains quotations:

- Don't use quotes inside your string

- Escape your quotes that are within the string with a slash. To escape a quote just place a slash directly before the quotation mark, i.e. \"
- Use single quotes (apostrophes) for quotes inside your string.

Example:

```
<?php
        // This won't work because of the quotes around specialH5!
        echo "<h5 class="specialH5">I love using PHP!</h5>";
            // OK because we escaped the quotes!
        echo "<h5 class=\"specialH5\">I love using PHP!</h5>";
            // OK because we used an apostrophe '
        echo "<h5 class='specialH5'>I love using PHP!</h5>";
?>
```

If you want to output a string that includes quotations, either use an apostrophe (') or escape the quotations by placing a slash in front of it (\"). The slash will tell PHP that you want the quotation to be used within the string and NOT to be used to end echo's string.

DATA TYPES:

PHP have three types of data types. Scalar (Basic), Compound and Special types. PHP supports eight different data types, five of which are scalar and each of the remaining three has its own uniqueness.

SCALAR TYPES	COMPOUND TYPES	SPECIAL TYPES
integer	Array	Resource
float		
string	Object	NULL
boolean		

BASIC (SCALAR) DATA TYPES:

1) INTEGERS

Integers are whole numbers and are equivalent in range as your C compiler's long value. On many common machines, 32-bit signed integer with a range between –2,147,483,648 to +2,147,483,647.

Integers can be written in decimal, hexadecimal (prefixed with 0x), and octal notation (prefixed with 0), and can include +/- signs.

Some examples of integers include

240000

0xABCD
007
-100

2) FLOATING-POINT NUMBERS

Floating-point numbers (also known as real numbers) represent real numbers and are equivalent to your platform C compiler's double data type.
On common platforms, the data type size is 8 bytes and it has a range of approximately 2.2E–308 to 1.8E+308.
Floating-point numbers include a decimal point and can include a +/- sign and an exponent value.

Examples of floating-point numbers include

3.14
+0.9e-2
-170000.5
54.6E42

3) STRINGS

Strings in PHP are a sequence of characters that are always internally null terminated.
When writing string values in your source code, you can use double quotes ("), single quotes (') or here-docs or nowdocs to delimit them.

A string literal can be specified in four different ways:

1. double quoted
2. single quoted
3. heredoc syntax
4. nowdoc syntax (since PHP 5.3.0)

1. DOUBLE QUOTES

Normally string is enclosed in double-quotes (")

"PHP: Hypertext Pre-processor"
"GET / HTTP/1.0\n"
"1234567890"

Strings can contain all characters. Some characters can't be written as is, however, and require special notation known as escape sequences.

SEQUENCE	MEANING
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\f	form feed
\\	Backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	The character represented by the specified octal #—for example, \70 represents the letter 8
\x[0-9A-Fa-f]{1,2}	The character represented by the specified hexadecimal #—for_example, \0x32 represents the letter 2.


```
<?php
        echo "My name is \n Chetan Rathod";
?>
```

Output:

```
My name is
Chetan Rathod
```


2. SINGLE QUOTES

The simplest way to specify a string is to enclose it in single quotes (the character ').

Single quotes do not support all the double quotes' escaping characters.

IT SUPPORTS:

SEQUENCE	MEANING
\'	Single quote.
\\	Backslash, used when wanting to represent a backslash followed by a single quote

In single quotes we can not use escape sequences which are used in double quotes.

```
<?php
echo 'this is a simple string'; Outputs: this is a simple string
echo ' "I\'ll call you later"'; // Outputs: "I'll call you later"
echo 'My name is \n Chetan Rathod';
 // Outputs: My name is \n Chetan Rathod
echo 'Path is C:\\Abc'; // Outputs: Path is C:\Abc
?>
```

3. HEREDOC

Here-docs enable you to embed large pieces of text in your scripts. It is done by here-doc syntax (<<<). First <<< operator, after that an identifier, then in new line original string and again same identifier to close the quotation. Which may include lots of double quotes and single quotes. The closing identifier *must* begin in the first column of the line.

Also, the identifier must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores, and must start with a non-digit character or underscore.

Syntax:

```
<<< Identifier
Strings...
.....
.....
Identifier
```

```
<?php
        echo <<<ABC
                Example of "string"
                        spanning multiple lines_using 'heredoc' syntax.
```

```
                    ABC;
    ?>
```

Note: Heredoc text behaves just like a double-quoted string, without the double quotes. We can use all double quotes escape sequence character.

4. NOWDOC

Nowdocs are single-quoted strings, what heredocs are to double-quoted strings. A nowdoc is specified similarly to a heredoc, but *no parsing is done* inside a nowdoc. A nowdoc is identified with the same <<< sequence used for heredocs, but the identifier which follows is enclosed in single quotes, e.g. <<<'ABC'. All the rules for heredoc identifiers also apply to nowdoc identifiers.

Syntax:

```
<<< 'Identifier'
 Strings...
 .....
 .....
 Identifier
```

```
<?php
        echo <<<'ABC'
                Example of "string"
spanning multiple lines_using 'nowdoc' syntax.
                ABC;
    ?>
```

4) BOOLEANS:

A Boolean value can be either TRUE or FALSE. Booleans were introduced for the first time in PHP 4 and didn't exist in prior versions.

Syntax:

To specify a boolean literal, use the keywords TRUE or FALSE.

```
$variable = TRUE/FALSE;
```

```
<?php
    $flag1 = TRUE; // assign the value TRUE to $flag1
?>
```

Boolean variable is mostly used in control structures like 'if'.

SPECIAL TYPES

- **NULL**
- **RESOURCES**

- **NULL:**

Null is a data type with only one possible value: the NULL value.

It marks variables as being empty, and it's especially useful to differentiate between the empty string and null values of databases.

The isset($variable) operator of PHP returns false for NULL, and true for any other data type.

Syntax:

```
$var = NULL;
```

- **RESOURCES:**

Resources is a special data type, represent a PHP extension resource such as a database query, an open file, a database connection, and lots of other external types.

You will never directly touch variables of this type, but will pass them around to the relevant functions that know how to interact with the specified resource.

COMPOUND TYPES

- **ARRAY**
- **OBJECT**

1) ARRAY:

An array is a special variable, which can store multiple values in one single variable. If you have a list of items (a list of car names), storing the cars in single variables could look like this:

```
$cars1="I20";
$cars2="Verna";
$cars3="Ritz";
```

a. It is easy to store three cars in three variable but what about 300 cars?

b. The best solution here is to use an array!

c. An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

d. Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three types of arrays:

- Numeric array - An array with a numeric index
- Associative array - An array where each ID key is associated with a value
- Multidimensional array - An array containing one or more arrays

2) OBJECT:

a. The other compound data type supported by PHP is the object.
b. The object is a central concept of the object-oriented programming concept.
c. An object must be explicitly declared.
d. We can say that object is real world entity Persons, Bank Account, Cars, etc
e. Object is declared inside the class definition.

Example:

```
class Test
        {
            private $a;
                function setValue($val)
                        {
                            $this->a = $val;
                        }
        }
...
$T1 = new Test;
```


A class definition creates several attributes and functions. Here class name is Test. There is only one attribute, $a, which can be modified by using the method setValue(). If you want to access a data member or data function then you have to use object to access it.

Example:

```
$T1->setValue(200);
```

- Converting Data Types Using Type Casting
- Converting values from one data type to another is known as type casting.

- A variable can be evaluated once as a different type by casting it to another.
- This is possible by placing the future type in front of the variable to be cast.
- A type can be cast by inserting one of the operators.

TYPE CASTING OPERATORS:

(array) : Convert into Array
(bool) or (boolean): Convert into Boolean
(int) or (integer): Convert into Integer
(object): Convert into Object
(real) or (double) or (float): Convert into Float
(string): Convert into String

Examples:

- Cast an integer as a double:

```
$Height = (double) 13; // $Height = 13.0
```

- Cast the float to integer:

```
$score = (int) 14.8; // $score = 14
```

- Cast the string to integer:

```
$str = "Hello World";
echo (int) $Str; // returns 0
```

- It gives unexpected outcome.
- You can also cast a data type to be a member of an array.
- The value being cast simply becomes the first element of the array:

```
$Mark = 63;
    $Sub = (array) $Mark;
    echo $Sub[0]; // Outputs 63
```


Any data type can be cast as an object. The result is that the variable becomes an attribute of the object, the attribute having the name scalar:

```
$Car = "MG";
$obj = (object) $Car;
```

The value can then be referenced as follows:

```
print $obj->scalar; // returns "MG"
```

OPERATORS:

Operators are used to operate on values. In all programming languages, operators are used to manipulate or perform operations on variables and values. Operators in PHP are categorized in:

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- String Operators

PHP has two special operators:

- Execution Operator(` `)
- Error Control Operator(@)

1) ASSIGNMENT OPERATOR:

Assignment operators assign the value to the variable. Such an assignment of value is done with the "=" operator.

Example:

```
$my_var = 5;
$another_var = $my_var;
```

Now both $my_var and $another_var contain the value 5.

Note: Assignments can also be used in conjunction with arithmetic operators.(Short Hand Operator)

2) ARITHMETIC OPERATORS:

OPERATOR	MEANING	EXAMPLE
+	Addition	2 + 4
-	Subtraction	6 – 2
*	Multiplication	5 * 3
/	Division	15 / 3

%	Modulus	43 % 10

Example:

```
$addition = 2 + 4;
$subtraction = 6 – 2;
$multiplication = 5 * 3;
$division = 15 / 3;
$modulus = 5 % 2;
echo "Perform addition: 2 + 4 = ".$addition."<br />";
echo "Perform subtraction: 6 – 2 = ".$subtraction."<br />";
echo "Perform multiplication:  5 * 3 = ".$multiplication."<br />";
echo "Perform division: 15 / 3 = ".$division."<br />";
echo "Perform modulus: 5 % 2 = " . $modulus . ". Modulus is the remainder after the division operation has been performed. In this case it was 5 / 2, which has a remainder of 1.";
```

Output:

Perform_addition:_2_+_4_=_6

Perform subtraction: 6 – 2 = 4

Perform multiplication: 5 * 3 = 15

Perform division: 15 / 3 = 5

Perform modulus: 5 % 2 = 1. Modulus is the remainder after the division operation has been performed. In this case it was 5 / 2, which has a remainder of 1.

23

3) Comparison Operators:

Comparisons are used to check the relationship between variables and/or values. Comparison operators are used inside conditional statements and evaluate to either true or false.

Example:

Assume: $x = 4 and $y = 5;

OPERATOR	MEANING	EXAMPLE	RESULT
==	Equal To	$x == $y	false
!=	Not Equal To	$x != $y	true

<	Less Than	$x < $y	true
>	Greater Than	$x > $y	false
<=	Less Than or Equal To	$x <= $y	true
>=	Greater Than or Equal To	$x >= $y	false
===	Is identical Equal To	$x ===$y	True

Note: $x === $y True if $x and $y are equivalent and $x and $y have the same type

4) LOGICAL OPERATORS:

Much like the arithmetic operators, logical operators will play a major role in many of your PHP applications. They are used to make decisions based on the values of multiple variables. Logical operators make it possible to direct the flow of a program and are used frequently with control structures such as the if conditional and the while and for loops.

OPERATOR	MEANING	EXAMPLE	RESULT
&& , AND	Logical AND	$x && $y , $x AND $y	AND True if both $x and $y are true
|| , OR	Logical OR	$x || $y , $x OR $y	OR True if both $x and $y are true
! , NOT	Logical NOT	!$x , NOT $x	NOT True if $x is not true
XOR	Logical XOR	$x XOR $y	Exclusive OR True if only $x or only $y is true

5) BITWISE OPERATORS:

Bitwise operators examine and manipulate integer values on the level of individual bits. First see some binary value of decimal numbers

DECIMAL	BINARY
2	10
5	101
10	1010
12	1100
145	10010001
1,452,012	101100010011111101100

OPERATOR	MEANING	EXAMPLE	RESULT
&	Bitwise AND	$x & $y	And together each bit contained in $x and $y
|	Bitwise OR	$x | $y	Or together each bit contained in $x and $y
^	Bitwise XOR	$ax^ $y	Exclusive—or together each bit contained in $x and $y
~	Bitwise NOT	~$x	Negate each bit in $x
<<	Bitwise_Shift Left	$x << $y	$x will receive the value of $y shifted left two bits
>>	Bitwise_Shift Right	$x >> $y	$x will receive the value of $y shifted right two bits

6) STRING OPERATOR:

Dot (.) is used as a string operator for concatenation of strings. "." is used to add two strings together.

Example:

```
$a_string = "Hello";
$another_string = " Chetan";
$new_string = $a_string . $another_string;
echo $new_string . "!";
```

Output:

Hello Chetan!

7) COMBINATION ARITHMETIC & ASSIGNMENT OPERATORS (SHORTHAND OPERATOR):

In programming it is a very common task to have to increment a variable by some fixed amount. The most common example of this is a counter. For example you want to increment a counter by 1, you would have:

```
$counter = $counter + 1;
```

However, there is a shorthand for doing this.

```
$counter += 1;
```

This combination assignment/arithmetic operator would use for same task.

Examples:

OPERATOR	ENGLISH	EXAMPLE	EQUIVALENT OPERATION
+=	Plus Equals	$x += 2;	$x = $x + 2;
-=	Minus Equals	$x -= 4;	$x = $x – 4;
*=	Multiply Equals	$x *= 3;	$x = $x * 3;
/=	Divide Equals	$x /= 2;	$x = $x / 2;
%=	Modulo Equals	$x %= 5;	$x = $x % 5;
.=	Concatenate Equals	$my_str.="hello";	$my_str = $my_str . "hello";

Pre/Post-Increment & Pre/Post-Decrement

To add one to a variable or "increment" use the "++" operator:

$x++; Which is equivalent to $x += 1; or $x = $x + 1;

To subtract 1 from a variable, or "decrement" use the "–" operator:

$x--; Which is equivalent to $x -= 1; or $x = $x – 1;

- There are different between pre increment and post increment.
- In pre increment (if we use it with an assignment) then first the variable is incremented after that value (incremented) is assigned to the right side from the =.
- In post increment (if we use it with an assignment) then first value is assigned to the right side from the = after that variable is incremented..

Example:

```
$x = 4;
echo "The value of x with post-plusplus = " . $x++;
echo "<br /> The value of x after the post-plusplus is " . $x;
$x = 4;
echo "<br />The value of x with with pre-plusplus = " . ++$x;
echo "<br /> The value of x after the pre-plusplus is " . $x;
```

Output:

```
The value of x with post-plusplus = 4
The value of x after the post-plusplus is = 5
The value of x with with pre-plusplus = 5
The value of x after the pre-plusplus is = 5
```

8) EXECUTION OPERATOR:

PHP supports one execution operator: backticks (``). Note that these are not single-quotes. PHP will execute the contents of the backticks as a shell command. It will produce the output of the shell command.

```
<?php
        $output = `ls -al`;
                echo "<pre>$output</pre>";
?>
```

9) ERROR CONTROL OPERATOR:

PHP supports one error control operator: the at sign (@). Error message generated by the expression will be saved in the variable $php errormsg. And any error messages that might be generated by that expression will be ignored. This variable will be overwritten on each error, so check early if you want to use it.

```
<?php
        /* Intentional file error */

$my_file = @file ('non_existent_file') or  die ("Failed opening file: error was '
$php_errormsg'");
        // this works for any expression, not just functions:
$value = @$cache[$key];
        // will not issue a notice if the index $key doesn't exist.
?>
```

OPERATOR PRECEDENCE:

Operator precedence is a characteristic of operators that determines the order in which they evaluate the operands.

NEW	NA	OBJECT INSTANTIATION
( )	NA	Expression subgrouping
[ ]	Right	Index enclosure
! ~ ++ --	Right	Boolean NOT, bitwise NOT, increment, decrement
@	Right	Error suppression
/ * %	Left	Division, multiplication, modulus
+ - .	Left	Addition, subtraction, concatenation
<< >>	Left	Shift left, shift right (bitwise)

< <= > >=	NA	Less than, less than or equal to, greater than, greater than or equal to
== != === <>	NA	Is equal to, is not equal to, is identical to, is not equal to
& ^ |	Left	Bitwise AND, bitwise XOR, bitwise OR
&& ||	Left	Boolean AND, Boolean OR
?:	Right	Ternary operator
= += *= /= .= %=&= |= ^= <<= >>=	Right	Assignment operators
AND XOR OR Left Boolean AND, Boolean XOR, Boolean OR	Left	Expression separation

Consider few examples:

```
$total = $a + $b * 0.06;
$total = $a + ($b * 0.06);
```

Because the multiplication operator has higher precedence than the addition operator.

OPERATOR ASSOCIATIVITY

The associativity is characteristic of an operator specifies how operations of the same precedence (i.e. having the same precedence value like two +) are evaluated as they are executed. It follow the left-to-right associativity. Left-to-right associativity means that the various operations making up the expression are evaluated from left to right.

Example:

```
$value = 5 * 44 + 21 / 3 * 2;
here $value=234;
```


VARIABLES: STATIC AND GLOBAL VARIABLES:

IDENTIFIERS:

Identifier is a general term(name) applied to variables, functions, and various other user-defined objects. An identifier can consist of one or more characters and must begin with a

letter or an underscore. Identifiers can consist of only letters, numbers, underscore characters shows a few examples of valid and invalid identifiers.

Valid and invalid identifiers are:

- my_function // Valid
- marks&tot // Invalid
- Total // Valid
- !sub // Invalid
- _name // Valid
- 2sub // Invalid

Rules of Identifiers:

- Identifiers are case sensitive.
- So that $total and $Total both are different identifiers name.
- Identifiers can be any length.
- An identifier name can't be the same to any of PHP's predefined keywords.

VARIABLES:

Variables are used for storing a values, like text strings, numbers or arrays.
When a variable is declared, it can be used over and over again in your script.
Value of the variable may be changed during run time.
All variables in PHP start with a $ sign symbol.

Syntax:

```
$variable_name = value;
```

Example:

```
$sub = 'PHP';
$total = 350;
```


NAMING RULES FOR VARIABLES

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)

- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string), or with capitalization ($myString)

```
<?php
    $2sub = 'PHP';    // invalid; starts with a number
        $_2sub = 'UNIX';   // valid; starts with an underscore
?>
```

The variable name is case-sensitive.

```
<?php
    $Sub = 'PHP';
        $sub = 'UNIX';
            echo "$Sub, $sub";    // outputs "PHP, UNIX"
?>
```

There are two way to assign the value to the variable

- **Asign by Value**
- **Asign by Refference**

a. ASSIGN BY VALUE:

By default, variables are always assigned by value but we can also assign an expression to the variable.

```
$total = 350;
$avg = $total / 5;
```

Note : It is not necessary to initialize variables in PHP however it is a very good practice

b. ASSIGN BY REFERENCE

Another way of assigning the value is assign by reference (Like a pointer in C). It means we assign the real value of variable (address) to another variable(an alias name). To assign by reference, simply put an ampersand (&) to the beginning of the variable which is being assigned.

Example:

```
<?php
    $sub1 = 'UNIX';    // Assign the value 'UNIX' to $sub1
        $sub2 = &$sub1;   // Reference $sub1 via $sub2.
            echo $sub1;    // Output UNIX
            echo $sub2;     // Output UNIX
        $sub1 = 'PHP';   // Assign the value 'PHP' to $sub1
            echo $sub1;    // Output PHP
```

```
                    echo $sub2;       // Output PHP
        ?>
```

In above example we see that $sub1 is assigned to $sub2 by reference. So changed in $sub1 is also effected to $sub2. This is possible only in Assign by reference.

```
<?php
        $sub1 = 'UNIX';          // Assign the value 'UNIX' to $sub1
                $sub2 = $sub1;          // assign $sub1 to $sub2.
                        echo $sub1;     // Output UNIX
                        echo $sub2;       // Output UNIX
                $sub1 = 'PHP';
                        echo $sub1;  // Output PHP
                        echo $sub2;     // Output UNIX
?>
```

Here we can see that changes of $sub1 is not effected to $sub2.

SCOPE OF THE VARIABLE:

The scope of a variable is the context within which it is defined. However you declare your variables (by value or by reference), you can declare them anywhere in a PHP script.

```
<?php
        $a = 1; // global scope
        function test()
                {
                        $b = 2; // local scope
                         echo $a; // not produce any output
                        echo $b; // output is 2
                }
                 echo $b; // not produce any output
        test();
?>
```

This script will not produce any output of echo $a; statement in test() function because the *$a* variable is out of test() function, and it is not of this scope. But echo $b; statement produce the output(2). Like this echo $b; (outside of test()) will not produce any output because $b is a variable of test() function.

NOTE: You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions. This can cause some problems in

that people may accidentally change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function.

PHP VARIABLES CAN BE ONE OF FOUR SCOPE TYPES:

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

LOCAL VARIABLES:

A variable declared in a function is considered local. That is, it can be referenced only in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function. When you exit the function in which a local variable has been declared, that variable is destroyed.

```
<?php
        $x = 40;
            function assignx ()
                {
                    $x = 10;
                    echo $x;  // inside function
                }
        assignx();
        echo $x;  // outside of function
    ?>
```

EXECUTING THIS LISTING RESULTS IN THE FOLLOWING:

```
10
40
```

As you can see, two different values for $x are output.

This is because the $x located inside the assignx() function is local. Modifying the value of the local $x has not affected on any values located outside of the function. Modifying the $x located outside of the function has not affected on any variables contained in assignx().

FUNCTION PARAMETERS:

In PHP, as in many other programming languages, any function that accepts arguments must declare those arguments in the function header. Those arguments accept values that come from outside of the function; they are no longer accessible once the function has exited.

Note: This section applies only to parameters passed by value and not to those passed by reference. Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

Example:

```
function x10 ($value)
    {
        $value = $value * 10;
        return $value;
    }
```

Keep in mind that although you can access and manipulate any function parameter in the function in which it is declared, it is destroyed when the function execution ends.

GLOBAL VARIABLES:

A global variable can be accessed in any part of the program. To modify a global variable, inside a function we can use global keyword or $GLOBALS array for accessing global variable.

- **The GLOBAL Keyword:**
  global keyword is use to access a global variable inside a local function.

Example :

```
<?php
    $a = 1;
    $b = 2;
        function Sum()
            {
              global $a, $b;
                $b = $a + $b;
                echo $b;
```

```
        }
    Sum();
?>
```

The above script will output 3.

By declaring *$a* and *$b* global within the function. There is no limit to the number of global variables that can be manipulated by a function.

- **The $GLOBALS array:**

  A second way to access variables from the global scope is to use the special PHP-defined $GLOBALS array.

  The previous example can be rewritten as:

Example:

```
<?php
    $a = 1;
    $b = 2;
        function Sum()
            {
              $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
            }
     Sum();
        echo $b;
 ?>
```

STATIC VARIABLES:

Another important feature of variable scoping is the *static* variable. A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

Example:

```
<?php
    function test()
        {
         $a = 10;
            echo $a;
                $a++;
        }
 ?>
```

This function is quite useless since every time it is called it sets *$a* to 10 and prints 10. The *$a++* which increments the variable has no meaning since as soon as the function exits the *$a* variable disappears. To make a useful counting function which will not lose track of the current count, the *$a* variable is declared static:

Example:

```
<?php
    function test()
        {
         static $a = 10;
                echo $a;
                    $a++;
        }
 ?>
```

Now, *$a* is initialized only in first call of function and every time the test() function is called it will print the value of *$a* and increment it. Static variables also provide one way to deal with recursive functions. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. The following simple function recursively counts to 10, using the static variable *$count* to know when to stop:

Example:

```
<?php
    function test()
        {
            static $count = 0;
                $count++;
                    echo $count . "<br>";
                    if ($count < 10)
                    {
                        test();
                    }
                }
    test();
 ?>
```

Output:

1
2

```
3
4
5
6
7
8
9
10
```

Note: We can only assign the value to static variable but not the result of expressions it will cause a parse error.

Example:

```
<?php
    function test()
    {
        static $a = 0;        // correct
        static $b = 1+2;      // wrong  (as it is an expression)
        static $c = sqrt(121); // wrong  (as it is an expression too)
    }
?>
```

Note: Static declarations are resolved in compile-time.

VARIABLE VARIABLES:

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement such as:

```
<?php
    $a = 'hello';
?>
```

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example, *hello*, can be used as the name of a variable by using two dollar signs. i.e.

```
<?php
    $$a = 'world';
?>
```

At this point two variables have been defined and stored in the PHP symbol tree. $a with contents "hello" and $hello with contents "world". Therefore, this statement:

```
<?php
```

```
        echo "$a ${$a}";
?>
```

produces the exact same output as:

```
<?php
        echo "$a $hello";
?>
```

i.e. they both produce: hello world.

CONSTANT:

Constant is used to store the fixed value. You can define a constant by using the define()-function or by using the const keyword. Once a constant is defined, it can never be changed or undefined. You can get the value of a constant by simply specifying its name no need to use $ with the constant name.

1. Defining Constants using define() function

```
<?php
        define("NAME", "Sachin Tendulkar");
            echo NAME; // outputs "Sachin Tendulkar"
                echo Name; // outputs "Name" and issues a notice.
?>
```

2. Defining Constants using the const keyword

```
<?php
        // Works as of PHP 5.3.0
        const NAME = 'Sachin Tendulkar';
        echo NAME;
?>
```


DIFFERENCES BETWEEN CONSTANTS AND VARIABLES:

A. Constants do not have a dollar sign ($) before them;

B. Constants may only be defined using the define() function, not by simple assignment;

C. Constants may be defined and accessed anywhere without regard to variable scoping rules;

D. Constants may not be redefined or undefined once they have been set; and

E. Constants may only evaluate to scalar values(boolean, integer, float and string).

F. Constants must be declared at the top-level scope because they are defined at compile-time. This means that they cannot be declared inside functions, loops or if statements.

G. You can also use the function constant() to read a constant's value if you wish to get the constant's name dynamically. Use get_defined_constants() to get a list of all defined constants.

COMMENTS

Comments in PHP are similar to comments that are used in HTML. Main purpose of the comment is to serve as a note to you, the web developer or to others who may view your website's source code. However, PHP's comments are different in that they will not be displayed to your visitors. The only way to view PHP comments is to open the PHP file for editing. This makes PHP comments only useful to PHP programmers.

PHP has two types of comments.

SINGLE LINE COMMENT:

The first type is the single line comment. The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. To do a single line comment type "//" or "#" and all text to the right will be ignored by PHP interpreter.

Example:

```
<?php
echo "Hello World!"; // This will print out Hello World!
echo "<br /> You can't see my PHP comments!"; // echo "nothing";
        // echo "My name is Chetan Rathod!";
        # echo "I don't do anything either";
?>
```

Output:

```
Hello World!
You can't see my PHP comments!
```


MULTIPLE LINE COMMENT:

The multi-line PHP comment can be used to comment out large blocks of code or writing multiple line comments. The multiple line PHP comment begins with " /* " and ends with " */ " like multiple line comment in' C' language .

Example:

```
<?php
        /* This Echo statement will print out my message to the
        the place in which I reside on.  In other words, the World. */
        echo "Hello World!";
        /* echo "My name is Chetan Rathod!";
        echo "I am PHP Programmer!";*/
?>
```

Output:

```
Hello World!
```

CONTROL STATEMENTS:

CONDITION STATEMENTS

Conditional statements are used to perform different actions based on different conditions. You can use conditional statements in your code to do this. In php we have the following conditional statements:

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

IF...ELSE

THE IF STATEMENT

Use the if statement to execute some code only if a specified condition is true.

Syntax :

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

Example:

```
<html>
    <body>
        <?php
```

```
                $n=5;
                if ($n==5)
                  echo "YES N==5";
            ?>
        </body>
    </html>
```

Note: Notice that there is no ..else.. in this syntax. The code is executed only if the specified condition is true. And if the condition is false, then it will not display anything.

THE IF...ELSE STATEMENT

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

Syntax:

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

Example:

```
<html>
    <body>
        <?php
            $d=date("D"); // date("D")Mon through Sun
                if ($d=="Sat")
                    echo "Have a nice weekend!";
                else
                    echo "Have a nice day!";
            ?>
    </body>
</html>
```


Another example:

```
<html>
    <head>
        <title>if_else</title>
    </head>
        <body>
```

```
<?php
    $a = 5;
        $b = "10";
  if ($a>$b)
      echo "$a is greater than $b";
  else
      echo  "$b is greater than $a";
?>
</body>
</html>
```

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

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Sat")
  {
    echo "Hello!<br />";
    echo "Have a nice weekend!";
    echo "See you on Monday!";
  }
else
    echo "Hello!, Have a nice Day! <br />";
?>
</body>
</html>
```


THE IF...ELSEIF....ELSE STATEMENT

Use the if....elseif...else statement to select one of several blocks of code to be executed.

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 depend on the current day

```
<html>
    <body>
        <?php
            $d=date("D");
            if ($d=="Sun")
                echo "Have a great Holiday!";
            elseif ($d=="Mon")
                echo "Today is Monday!";
            elseif ($d=="Tue")
                echo " Today is Tuesday!";
            elseif ($d=="Wed")
                echo " Today is Wednesday!";
            elseif ($d=="Thu")
                echo " Today is Thursday!";
            elseif ($d=="Fri")
                echo " Today is Friday!";
            elseif ($d=="Sat")
                echo "Have a nice Weekend!";
            else
                echo "Have a nice day!";
        ?>
    </body>
</html>
```

WRITE A PRG TO CHECK WHETHER A VALUE IS LARGER, SMALLER, OR EQUAL TO ANOTHER VALUE.

```
<html>
  <head>
    <title>Demo</title>
  </head>
  <body>
    <?php
        $a = 5;
        $b = 5;
```

```
                    if ($a==$b)
                        echo "$a is equal to $b";
                    elseif($a<$b)
                        echo  "$a is smaller than $b";
                    elseif($a>$b)
                        echo  "$a is greter than $b";
            ?>
        </body>
    </html>
```

SWITCH

If you want to select one of many blocks of code to be executed, use the switch statement. Conditional statements are used to perform different actions based on different conditions. You can use it instead of if...elseif....else statement.

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
             label1 and label2;
    }
```

First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example:

```
<html>
    <body>
        <?php
```

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

Output :

Number 3

'?' OPERATOR (TERNARY OPERATOR)

The ternary operator is a great way to apply a quick true/false. It is the combination of if else statement.

Syntax:

(Condition)? True Result : False Result;

Example:

```
<?php
    $num = 42;
      if ($num < 0)
        {
          $value = 'The value is negative.';
        }
      else
        {
```

```
        if ($num > 0)
            $value = 'The value is positive.';
        else
            $value = 'The value is zero.';
    }
? >
```

This is much easier to read and understand to use ?:

```
< ?php
    $num = 42;
        echo 'The value is ';
    echo ($num < 0) ? 'negative.' : (($num > 0) ? 'positive.' : 'zero.');
? >
```

LOOPS

Loops execute a block of code a specified number of times, or while a specified condition is true. Often when you write code, you want the same block of code to run over and over again in a row. If you want to execute some block of code then you can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

WHILE

The while loop executes a block of code while a condition is true.

Syntax:

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

Example:

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

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

Output:

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

BREAK STATEMENT

break ends the execution of the current for, foreach, while, do-while or switch structure. break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

```
<?php
    $arr = array('one', 'two', 'three', 'four', 'stop', 'five');
    while (list(, $val) = each($arr))
    {
        if ($val == 'stop')
            {
                break;    /* You could also write 'break 1;' here. */
```

```
                }
                echo "$val<br />\n";
        }
    /* Using the optional argument. */
        $i = 0;
        while (++$i)
          {
            switch ($i)
             {
                case 5:
                    echo "At 5<br />\n";
                     break 1;  /* Exit only the switch. */
                 case 10:
                    echo "At 10; quitting<br />\n";
                     break 2;  /* Exit the switch and the while. */
                     default:
                     break;
            }
          }
        ?>
```

CONTINUE

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration. The continue statement causes execution of the current loop iteration to end and commence at the beginning of the next iteration.

Example:

```
<?php
    $friends =array("Raju","Ram","Kiaan","Manish","missing","Payal");
    for ($i=0; $i < count($friends); $i++)
        {
            if ($friends[$i] == "missing") continue;
            printf("Friend : %s <br />", $friend[$i]);
        }
?>
```

Output:

Friend : Raju
Friend : Ram
Friend : Kiaan
Friend : Manish
Friend : Payal

Note: continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

DO...WHILE

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

Syntax:

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

Example:

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

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

Output:

```
The number is 2
The number is 3
The number is 4
```

FOR

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

Syntax:

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

PARAMETERS:

- **init:** Mostly used to set a counter.(or beginning of loop)
- **condition:** Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **increment:** Mostly used to increment a counter (or decrement)

Note: Each of the parameters above can be empty, or have multiple expressions (separated by commas).

Example:

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

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


Output:

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

FOR EACH

The foreach loop is used to loop through arrays.

Syntax:

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

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

Example:

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

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

Output:

```
one
two
three
```

EXIT, DIE, RETURN

EXIT():

exit()- Output a message and terminate the current script

exit ([string $status])

exit (int $status)

Terminates execution of the script.

PARAMETERS

status

- If status is a string, this function prints the status just before exiting.
- If status is an integer, that value will be used as the exit status and not printed.

Example:

```
<?php
        //exit program normally
                exit;
                exit();
                exit(0);
        //exit with an error code
        exit(1);
?>
```

DIE:

die — Equivalent to exit()

It terminate the execution and print the error message which user want to print.

```
<?php
        if ($marks > 100)
          {
            die ("Error, Marks is always less then 100");
          }
?>
```

If Marks Is grater then 100 it will print the Message and exit from the script.

RETURN:

If called from within a function, the **return()** statement immediately ends execution of the current function, and returns its argument as the value of the function call.

```
<?php
        function get_value($name)
                {
                        return $GLOBALS[$name];
                }
        $num = 10;
                $value = get_value($num);
        print $value;
?>
```

This code prints the number 10.

ARRAYS:

An array is a special variable, which can store multiple values in one single variable. If you have a list of items (a list of car names), storing the cars in single variables could look like this:

```
$city1="Surat";
$city2="Baroda";
$city3="Navsari";
```

It is easy to store three cars in three variable but what about 300 cities? The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own index so that it can be easily accessed.

IN PHP, THERE ARE THREE TYPES OF ARRAYS:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

NUMERIC ARRAY:

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. Using array() function, index will assign automatically (the index starts at 0):

```
$cities=array("Surat","Baroda","Mumbai","Delhi");
```

2. Assign the index manually:

```
$cities[0]="Surat";
$cities[1]="Baroda";
$cities[2]="Mumbai";
$cities [3]="Delhi";
```


ACCESSING THE ARRAY:

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cities[0]="Surat";
$cities[1]="Baroda";
```

```
$cities[2]="Mumbai";
$cities [3]="Delhi";
echo $cities[0]. "and" . $cities[1]. "are cities of Gujarat.";
?>
```

Output:

Surat and Baroda are cities of Gujarat.

ASSOCIATIVE ARRAY:

An associative array, each ID key is associated with a value. When storing data about specific named values, a numerical array is not always the best way to do it. With associative arrays we can use the values as keys and assign values to them.

Example1:

In this example1 we use an array to assign ages to the different persons:

```
$result = array("Prince"=>78, "Vishal"=>80, "Ajay"=>64);
```

Example2:

This example2 is the same as example1, but shows a different way of creating the array:

```
$result['Prince'] = 78;
$result['Vishal'] = 80;
$result['Ajay'] = 64;
```

The ID keys can be used in a script:

```
<?php
$result['Prince'] = 78;
$result['Vishal'] = 80;
$result['Ajay'] = 64;
echo "Result of Vishal is " . $result['Vishal'] . "%.";
?>
```

Output:

Result of Vishal is 80%.

MULTIDIMENSIONAL ARRAY:

In a multidimensional 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.

Example 1:

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
        (
         "Shah"=>array
            (
                "Hetal",
                "Neel",
                "Rahul"
            ),
         "Rathod"=>array
            (
                "Gayatri"
            ),
        "Patel"=>array
            (
                "Kiran",
                "Hemant",
                "Chirag"
            )
        );
```

Output:

```
Array
(
    [Shah] => Array
        (
            [0] => Hetal
            [1] => Neel
            [2] => Rahul
        )
    [Rathod] => Array
        (
```

```
    [0] => Gayatri
  )
[Patel] => Array
  (
   [0] => Kiran
   [1] => Hemant
   [2] => Chirag
  )   )
```

Example 2:

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Shah'][2] . " a part of the Shah family?";
```

Output:

Is Rahul a part of the Shah family?