Unit No.	Topics	No. of Lectures
1.	**Introduction to PHP** • HTTP basics, Introduction to web server and web browser • Introduction to PHP • PHP - Lexical structure, Language basics.	6
2.	**Functions and Arrays** • Defining and calling a function • Default parameters • Variable parameters, Missing parameters • Variable function, Anonymous function • Indexed Vs Associative arrays • Identifying elements of an array • Storing data in arrays • Multidimensional arrays • Extracting multiple values • Traversing arrays • Sorting Using arrays	8
3.	**Object Oriented Programming** • Classes • Objects • Introspection • Serialization • Inheritance • Interfaces • Encapsulation	10
5.	**Web Techniques** • Variables • Server information • Processing forms • Setting response headers • Maintaining state • SSL	12

6.	**Databases** • Using PHP to access a database • Relational databases and SQL • PEAR DB basics • Advanced database techniques	12
7.	**Ajax and XML** • Understanding java scripts for AJAX • AJAX web application model • AJAX –PHP framework • Performing AJAX validation • Handling XML data using php and AJAX • What is XML? • XML document Structure • PHP and XML • XML parser • The document object model • The simple XML extension • Changing a value with simple XML	12

CONTENTS

VISION

www.visionpune.com

1. Online book store (Check-Select-Buy)
2. Know different courses at a glance.
3. Complete contents & other details at a glance.
4. Quick Search for books of different Universities.
5. Open your account & get discount points on each purchase.
6. Get the book at your door, don't waste time & money on petrol.
7. Know the top selling & special books along with new arrivals.
8. One solution for all your needs - Text Books / Paper Solution & Competitive Exams.

visionpublications@gmail.com

Unit 1

Introduction to PHP

1. Introduction

Internet (or The Web) is a massive distributed client/server information system. Many applications are running concurrently over the Web, such as web browsing/surfing, e-mail, file transfer, audio and video streaming, and so on. In order for proper communication to take place between the client and the server, these applications must agree on a specific application-level protocol such as HTTP, FTP, SMTP, POP, and etc.

HTTP Basics

HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).

i. HTTP is an *asymmetric request-response client-server* protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a *pull protocol*, the client *pulls* information from the server (instead of server *pushes* information down to the client).

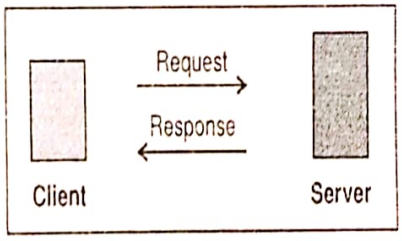

Figure 1.1: HTTP Protocol Basics

ii. HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.

iii. HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

iv. Quoting from the RFC2616: "The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers."

Browser

Whenever we issue a URL from our browser to get a web resource using HTTP, e.g. `http://www.visionpublications.com/index.html`, the browser turns the URL into a *request message* and sends it to the HTTP server. The HTTP server interprets the request message, and returns us an appropriate response message, which is either the resource we requested or an error message.

Uniform Resource Locator (URL)

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following *syntax*:

```
protocol://hostname:port/path-and-file-name
```

There are four parts in a URL

i. **Protocol:** The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.

ii. **Hostname:** The DNS domain name *(e.g.,* www.visionpublications.com) *or* IP address (e.g., 192.128.1.2) of the server.

iii. **Port:** The TCP port number that the server is listening for incoming requests from the clients.

iv. **Path-and-file-name:** The name and location of the requested resource, under the server document base directory.

For example, in the URL `http://www.visionpublications.com/docs/index.html`, the communication protocol is HTTP; the hostname is `www.visionpublications.com`. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is `"/docs/index.html"`.

HTTP Request and Response Messages

HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*.

The HTTP Request

Request and response message structures are the same as shown below.

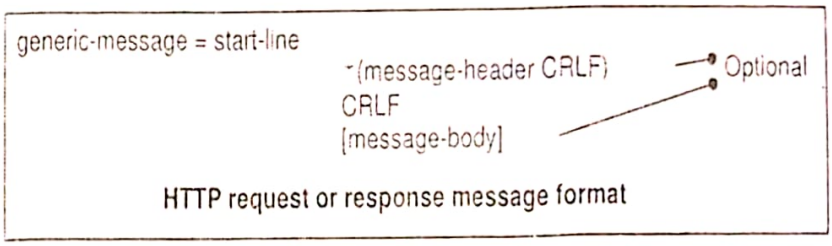

Figure 1.2

An HTTP client sends an HTTP request to a server in the form of a request message which includes the following format:

i. A Request-line.

ii. Zero or more header (General | Request | Entity) fields followed by CRLF.

iii. An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields.

iv. Optionally a message-body.

The HTTP response

After receiving and interpreting a request message, a server responds with an HTTP response message:

1. A Status-line.

2. Zero or more header (General | Response | Entity) fields followed by CRLF.

3. An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields.
4. Optionally a message-body.

The two most common HTTP methods are GET and POST. The GET method is designed for retrieving information, such as a document, an image, or the results of a database query, from the server. The POST method is meant for posting information, such as a credit-card number or information that is to be stored in a database, to the server. The GET method is what a web browser uses when the user types in a URL or clicks on a link. When the user submits a form, either the GET or POST method can be used, as specified by the method attribute of the form tag.

2. Introduction to Web Server and Web Browser

Web server is a computer where the web content is stored. Basically, web server is used to host the websites but there exists other web servers also such as gaming, storage, FTP, email etc.

Website is a collection of web pages while web server is software that responds to the request for web resources.

2.1 Web Server Working

Web server responds to the client request in either of the following two ways:

1. Sending the file to the client associated with the requested URL.
2. Generating response by invoking a script and communicating with database.

Some Important Points

1. When client sends request for a web page, the web server searches for the requested page, if requested page is found then it will send it to client with an HTTP response.
2. If the requested web page is not found, web server sends an HTTP response: Error 404 Not found.

3. If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

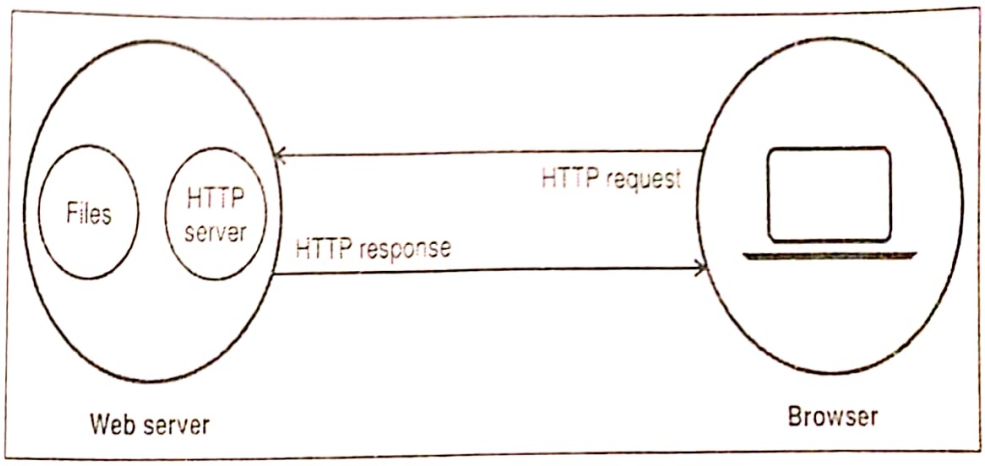

Figure 1.3

Some of the Famous Web Servers are

1. **Apache HTTP Server:** This is the most popular web server in the world developed by the Apache Software Foundation. Apache web server is an open source software and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more. About 60% of the web server machines run the Apache Web Server.

2. **Internet Information Services (IIS):** The Internet Information Server (IIS) is a high performance web server from Microsoft. Th... ...s on Windows NT/2000 and 2003. platforms (and may ... upcomin... Windo... version ... IIS comes bundled with Windows NT/... in ... ...ting system it is relatively ea...

Web Browser

A web browser is a ... images, videos, music, a... web page can contain ... browsers allow a user to quic... many websites by traversing th... the appearance of a web page may ...

Protocols and Standards

Web browsers communicate with web servers primarily using HTTP (hypertext transfer protocol) to fetch web pages. HTTP allows web browsers to submit information to web servers as well as fetch web pages from them. Pages are identified by means of a URL (uniform resource locater), which is treated as an address, beginning with "http://" for HTTP access.

The file format for a web page is usually HTML (hyper-text markup language) and is identified in the HTTP protocol. Most web browsers also support a variety of additional formats, such as JPEG, PNG, and GIF image formats, and can be extended to support more through the use of plugins. The combination of HTTP content type and URL protocol specification allows web page designers to embed images, animations, video, sound, and streaming media into a web page, or to make them accessible through the web page.

Popular Browsers

1. **Firefox:** Firefox is a very popular web browser. One of the great things about Firefox is that it is supported on all different OSs. Firefox is also open source which makes its support group a very large community of open source developers. Firefox is also known for its vast range of plugins/add-ons that let the user customize in a variety of ways. Firefox is a product of the Mozilla Foundation. The latest version of Firefox is Firefox 3.

2. **Internet Explorer:** Internet Explorer (IE - created by Microsoft) is a very prominant web browser for the Windows OS. IE is the most popular web browser. It comes pre-installed on all Windows computers. The latest version of IE is IE7 with IE8 in beta. IE was designed to view a broad range of web pages and to provide certain features within the OS.

3. Introduction to PHP

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.

1. PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
2. 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.

3. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

4. 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.

5. PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 add support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.

What PHP can do?

PHP is mainly focused on server-side scripting, so we can program, such as collect form data, generate dynamic page content, or send and receive cookies. There are three main areas where PHP scripts are used.

1. **Server-side scripting:** This is the most traditional and main target field for PHP. We need three things to make this work: the PHP parser (CGI or server module), a web server and a web browser. We need to run the web server, with a connected PHP installation. We can access the PHP program output with a web browser, viewing the PHP page through the server. All these can run on our home machine if we are just experimenting with PHP programming.

2. **Command line scripting:** We can make a PHP script to run it without any server or browser. We only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks.

3. **Writing desktop applications:** PHP is probably not the very best language to create a desktop application with a graphical user interface, but if we know PHP very well, and would like to use some advanced PHP features in our client-side applications, you can also use PHP-GTK to write such programs. We also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution.

Features of php

The main features of php are:

1. It is open source scripting language so you can freely download this and use.
2. PHP is a server site scripting language.
3. It is open source scripting language.
4. It is widely used all over the world.
5. It is faster than other scripting languages.

Some important features of php are given below:

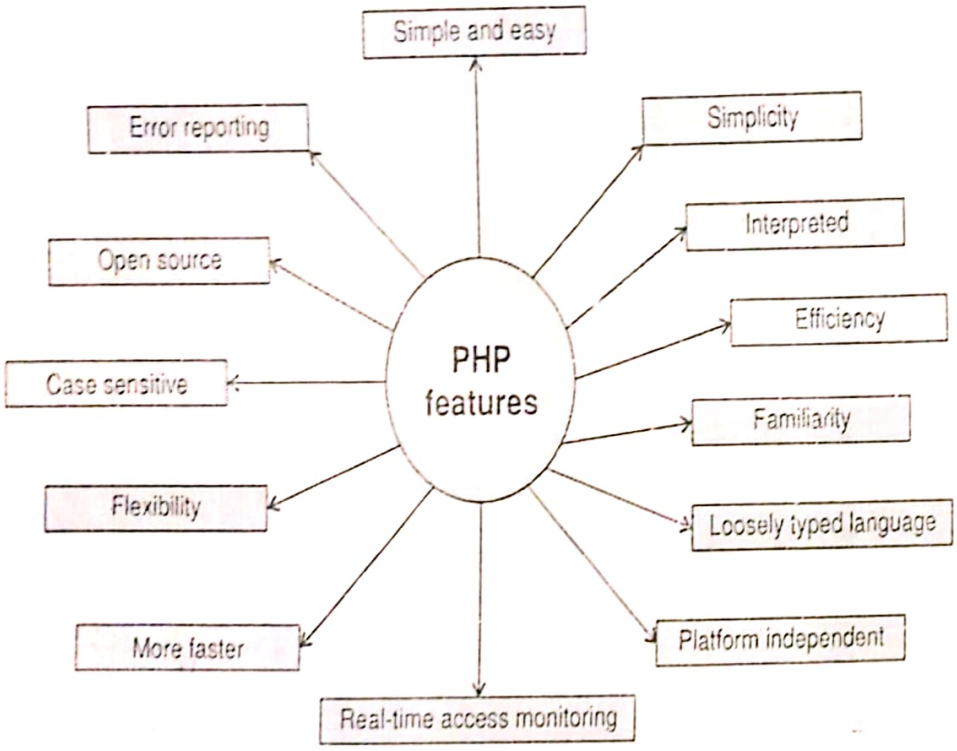

Figure 1.4

1. **Simple:** It is very simple and easy to use, compared to other scripting languages. It is widely used all over the world.
2. **Interpreted:** It is an interpreted language, i.e., there is no need for compilation.
3. **Faster:** It is faster than other scripting languages e.g. ASP and JSP.

4. **Open Source:** Open source means no need to pay for using php. we can freely download and use.

5. **Platform Independent:** PHP code can be run on every platform. Linux. Unix. Mac OS X. Windows.

6. **Case Sensitive:** PHP is case sensitive scripting language at time of variable declaration. In PHP. all keywords (e.g. if. else. while. echo. etc.). classes. functions. and user-defined functions are NOT case-sensitive.

7. **Error Reporting:** PHP has some predefined error reporting constants to generate a warning or error notice.

8. **Real-Time Access Monitoring:** PHP provides access logging by creating the summary of recent accesses for the user.

9. **Loosely Typed Language:** PHP supports variable usage without declaring its data type. It will be taken at the time of the execution based on the type of data it has on its value.

How Php Works

PHP is an interpreted language: it means that PHP will need to interpret and compile the code of our application for each request made to it.

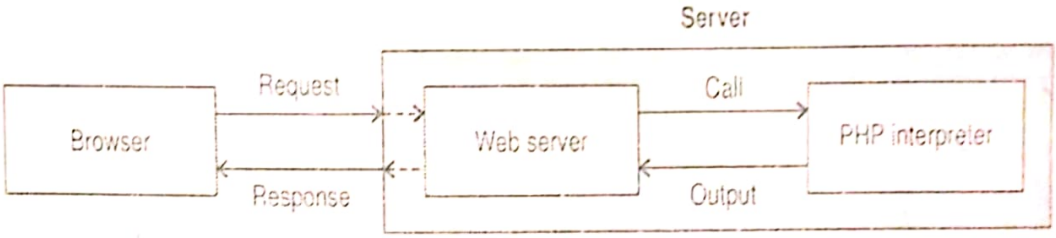

Figure 1.5

As we can see in the diagram above,

1. A browser makes a request for a web page.
2. This request is going to hit the web server.
3. The web server will then analyze it and determine what to do with it.

If the web server determines that the request is for a PHP file , it'll pass that file to the PHP interpreter.

1. The PHP interpreter will read the PHP file, parse it (and other included files) and then execute it.
2. Once the PHP interpreter finishes executing the PHP file, it'll return an output. The web server will take that output and send it back as a response to the browser.

4. PHP - Lexical Structure

Computer languages, like human languages, have a lexical structure. The lexical structure of a programming language is the set of basic rules that governs how to write programs in that language. It is the lowest-level syntax of the language and specifies such things as what variable names look like, what characters are used for comments, and how program statements are separated from each other. A source code of a PHP script consists of tokens. Tokens are atomic code elements. In PHP language, we have comments, variables, literals, operators, delimiters, and keywords.

PHP Comments

Comments give information to people who read our code, but they are ignored by PHP. All comments in PHP follow the # character.

i. **Shell-style comments:** When PHP encounters a hash mark (#) within the code, everything from the hash mark to the end of the line or the end of the section of PHP code (whichever comes first) is considered a comment. This method of commenting is found in Unix shell scripting languages and is useful for annotating single lines of code or making short notes.

```
<?php
  # comments.php
  # learning to comments
  # vision publication
  echo "This is comments.php script\n";
```

Everything that follows the # character is ignored by the PHP interpreter.

ii. **C++ comments:** When PHP encounters two slash characters (//) within the code, everything from the slashes to the end of the line or the end of the section of code,

whichever comes first, is considered a comment. This method of commenting is derived from C++.

```
// comments.php
// single line comment
// vision publications
```

iii. **C comments:** PHP supports block comments, whose syntax comes from the C programming language. When PHP encounters a slash followed by an asterisk (/*), everything after that until it encounters an asterisk followed by a slash (*/) is considered a comment.

```
/* comments.php
 multiple line comments like the C language
*/
```

PHP also recognizes the comments from the C language.

Statements and Semicolons

A statement is a collection of PHP code that does something. It can be as simple as a variable assignment or as complicated as a loop with multiple exit points.

PHP uses semicolons to separate simple statements. A compound statement that uses curly braces to mark a block of code, such as a conditional test or loop, does not need a semicolon after a closing brace. Unlike in other languages, in PHP the semicolon before the closing brace is not optional:

```
if($needed)
{
  echo "We must have it!"; // semicolon required here
} // no semicolon required here
```

The semicolon is optional before a closing PHP tag: It's good programming practice to include optional semicolons, as they make it easier to add code later.

PHP White Space

White space in PHP is used to separate tokens in PHP source file. It is used to improve the readability of the source code.

```
public $varname;
```

White spaces are required in some places; *for example*, between the access specifier and the variable name. In other places, it is forbidden. It cannot be present in variable identifiers.

```
$x=1;
$y=2;
$z=3;
```

The amount of space put between tokens is irrelevant for the PHP interpreter. It is based on the preferences and the style of a programmer.

```
$x = 1;
$y = 2; $z = 3;
$a = 4;
```

We can put two statements into one line. Or one statement into three lines. We can take advantage of this flexible formatting to make our code more readable (by lining up assignments, indenting, etc.).

Identifier

An identifier is simply a name. In PHP, identifiers are used to name variables, functions, constants, and classes.

i. **Variable Names:** A `variable` is an identifier, which holds a value. A variable is a reference to a computer memory, where the value is stored. In PHP language, a variable can hold a string, a number, or various objects like a function or a class. Variables can be assigned different values over time.

   Variables in PHP consist of the $ character, called a sigil, and a label. The label, first character of an identifier must be either an ASCII letter (uppercase or lowercase), the underscore character (_), or any of the characters between ASCII 0x7F and ASCII 0xFF. After the initial character, these characters and the digits 0-9 are valid.

   A variable cannot begin with a number. The PHP interpreter can then distinguish between a number and a variable more easily. These were valid PHP identifiers.

```
$num1
$num22
$name
```

These were *examples* of invalid PHP identifiers.

```
$12Value
$exxp$
$first-name-my
```

The variables are *case sensitive*. This means that `$Price`, `$price`, and `$PPICE` are different variable.

ii. **Variable Variables:** We can reference the value of a variable whose name is stored in another variable.

For example

```
$name = 'vision';
$$name1 = 'publications';
```

After the second statement executes, the variable $name1 has the value "publications".

iii. **Variable Reference:** Variable reference means to create a variable aliases. To make $name an alias for the variable $name1, use:

```
$name =& $name1;
```

The old value of $name is lost. Instead, $name is now another name for the value that is stored in $name1.

iv. **Variable Scope:** The scope of a variable, which is controlled by the location of the variable's declaration, determines those parts of the program that can access it.

There are four types of variable scope in PHP: local, global, static, and function

a. *Local scope:* A variable declared in a function is local to that function. That is, it is visible only to code in that function (including nested function definitions); it is not accessible outside the function. In addition, by default, variables defined outside a function (called global variables) are not accessible inside the function.

```
<?php
function Test()
{
  $x = 51; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
Test();
  // using x outside the function error
  echo "<p>Variable x outside function is: $x</p>";
?>
```

These were *examples of invalid PHP identifiers*

```
[illegible]
[illegible]
$first name [illegible]
```

The variables are *case sensitive*. This means that $Price, $price, and $PRICE are different variable.

ii. **Variable Variables:** We can reference the value of a variable whose name is stored in another variable.

For example

```
$name = 'vision';
$$name1 = 'publications';
```

After the second statement executes, the variable $name1 has the value "publications"

iii. **Variable Reference:** Variable reference means to create a variable aliases. To make $name an alias for the variable $name1, use:

```
$name =& $name1;
```

The old value of $name is lost. instead, $name is now another name for the value that is stored in $name1.

iv. **Variable Scope:** The scope of a variable, which is controlled by the location of the variable's declaration, determines those parts of the program that can access it.

There are four types of variable scope in PHP: local, global, static, and function

a. *Local scope:* A variable declared in a function is local to that function. That is, it is visible only to code in that function (including nested function definitions); it is not accessible outside the function. In addition, by default, variables defined outside a function (called global variables) are not accessible inside the function.

```
<?php
function Test()
{
  $x = 51; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
Test();
  // using x outside the function error
  echo "<p>Variable x outside function is: $x</p>";
?>
```

Scanned by CamScanner

White spaces are required in some places; *for example*, between the access specifier and the variable name. In other places, it is forbidden. It cannot be present in variable identifiers.

```
$x=1;
$y=2;
$z=3;
```

The amount of space put between tokens is irrelevant for the PHP interpreter. It is based on the preferences and the style of a programmer.

```
$x = 1;
$y = 2; $z = 3;
$a = 4;
```

We can put two statements into one line. Or one statement into three lines. We can take advantage of this flexible formatting to make our code more readable (by lining up assignments, indenting, etc.).

Identifier

An identifier is simply a name. In PHP, identifiers are used to name variables, functions, constants, and classes.

i. **Variable Names:** A `variable` is an identifier, which holds a value. A variable is a reference to a computer memory, where the value is stored. In PHP language, a variable can hold a string, a number, or various objects like a function or a class. Variables can be assigned different values over time.

Variables in PHP consist of the $ character, called a sigil, and a label. The label, first character of an identifier must be either an ASCII letter (uppercase or lowercase), the underscore character (_), or any of the characters between ASCII 0x7F and ASCII 0xFF. After the initial character, these characters and the digits 0-9 are valid.

A variable cannot begin with a number. The PHP interpreter can then distinguish between a number and a variable more easily. These were valid PHP identifiers.

```
$num1
$num22
$name
```

These were *examples* of invalid PHP identifiers.

```
$12Value
$exxp$
$first-name-my
```

The variables are *case sensitive*. This means that `$Price`, `$price`, and `$PRICE` are different variable.

ii. **Variable Variables:** We can reference the value of a variable whose name is stored in another variable.

For example

```
$name = 'vision';
$$name1 = 'publications';
```

After the second statement executes, the variable $name1 has the value "publications".

iii. **Variable Reference:** Variable reference means to create a variable aliases. To make $name an alias for the variable $name1, use:

```
$name =& $name1;
```

The old value of $name is lost. Instead, $name is now another name for the value that is stored in $name1.

iv. **Variable Scope:** The scope of a variable, which is controlled by the location of the variable's declaration, determines those parts of the program that can access it.

There are four types of variable scope in PHP: local, global, static, and function

a. *Local scope:* A variable declared in a function is local to that function. That is, it is visible only to code in that function (including nested function definitions); it is not accessible outside the function. In addition, by default, variables defined outside a function (called global variables) are not accessible inside the function.

```
<?php
function Test()
{
  $x = 51; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
Test();
  // using x outside the function error
  echo "<p>Variable x outside function is: $x</p>";
?>
```

```
Output
Variable x inside function is: 51
Variable x outside function is
```

b. *Global scope:* Variables declared outside a function are global. That is, they can be accessed from any part of the program.

To allow a function to access a global variable, we can use the global keyword inside the function to declare the variable within the function.

```
<?php
$x = 51; // global scope
function Test() {
  // using x inside this function will generate an error
  echo "<p>Variable value  x inside function is: $x</p>";
}
Test();
  echo "<p>Variable value x outside function is: $x</p>";
?>
```

```
Output
Variable value x inside function is:
Variable value x outside function is: 51
```

c. *Static variables:* A static variable retains its value between calls to a function but is visible only within that function. Static variable is declared with the static keyword.

For example

```
function Test()
{
  static $x = 10;
  echo $x;
  $x++;
}
Test();
e  o "<    ,
```

```
Output
10
11
12
```

d. Function parameters

A function definition can have named parameters:

```
function name($name)
{
 echo "My name is , $name\n";
}
 name("vision");
```

Output

My name is vision

Function parameters are local, meaning that they are available only inside their functions. In this case, $name is inaccessible from outside name().

Function names

Function names are not case-sensitive

Here are some valid function names:

Calculate_area
area_OF_circle
calculate_PAY

These function names refer to the same function:

payment
PaYmenT
PAYMENT
PAYment

Class names

Class names follow the standard rules for PHP identifiers and are not case-sensitive. Here are some valid class names:

Person
account

The class name stdClass is reserved.

PHP Literal

A literal is any notation for representing a value within the PHP source code. Technically, a literal is assigned a value at compile time, while a variable is assigned at runtime. The following are all literals in PHP:

```
2001
0xFE
1.4142
```

"Hello Vision Publications"

'Hii'

True

Null

Constants

A constant is an identifier for a value which cannot change during the execution of the script. By convention, constant identifiers are always uppercase. Only scalar values boolean, integer, double, and string can be constants.

Once set, the value of a constant cannot change. Constants are referred to by their identifiers and are set using the define() function:

```
define('PUBLISH', "Vision Publications"); echo PUBLISH;
```

Expressions and Operators

An expression is a bit of PHP that can be evaluated to produce a value. The simplest expressions are literal values and variables. A literal value evaluates to itself, while a variable evaluates to the value stored in the variable.

An operator is a symbol used to perform an action on some value.

Php Operator Table

Operators	Purpose
new	create new object
**	Exponentiation
++ --	increment/decrement
~(int) (float) (string) (array) (object) (bool)	Casting
instanceof	Types
!	Logical
* /	multiplication/division
%	Modulo
+ - .	arithmetic and string
<< >>	bitwise shift
< <= > >=	Comparison
== != === !== <> <=>	Comparison
&	bitwise and/references
^	bitwise XOR
|	bitwise OR
&&	logical and
||	logical or
??	null coalescing
? :	Ternary
= += -= *= **= /= .= %= &= |= ^= <<= >>= ??=	assignment operators

Number of Operands

Most of the operators in PHP are binary operators; they combine two operands (or expressions) into a single, more complex expression. PHP also supports a number of unary operators, which convert a single expression into a more complex expression. PHP also supports a single ternary operator that combines three expressions into a single expression.

Operator Precedence

The precedence of an operator specifies how "tightly" it binds two expressions together. In general, operators have a set precedence, or order, in which they are evaluated. Operators also have associativity, which is the order in which operators of the same precedence are evaluated. This order is generally left to right (called left for short), right to left (called right for short), or not relevant.

For example, in the expression **10 + 5 * 2**, the answer is **20** and not **30** because the multiplication * operator has a higher precedence than the addition + operator. Parentheses () may be used to force precedence, if necessary. For instance: **(10 + 5) * 2** evaluates to **30**.

Operator Associativity

Associativity is used when two operators of the same precedence appear in an expression.

For example: consider the following statement:

```
$f = 5 - 5 - 5;
```

Here we can have two possible results, 5 and -5. Here is how those two possibilities would look if we made our intentions explicit with brackets:

```
$f = 5 - (5 - 5); $f = (5 - 5) - 5;
```

Here the operator associativity, decides the direction in which operations are performed. The middle 5 has a minus on either side, but, because - is left-associative, the left-hand operation is performed first, giving the second possibility, and therefore -5.

Associativity	Operators
left	,
left	or
left	xor
left	and
right	print
right	= += -= *= /= .= %= &= |= ^= <<= >>=
left	? :
left	||
left	&&
left	|
left	^
left	&
non-associative	== != === !==
non-associative	< <= > >=
left	<< >>
left	+ - .
left	* / %
right	! ~ ++ -- (int) (float) (string) (array) (object) @
right	[
non-associative	New

Arithmetic Operators

Most of the arithmetic operators are binary; however, the arithmetic negation and arithmetic assertion operators are unary. These operators require numeric values, and non-numeric values are converted into numeric values. The arithmetic operators are:

1. **Addition (+):** The result of the addition operator is the sum of the two operands.
2. **Subtraction (-):** The result of the subtraction operator is the difference between the two operands; i.e., the value of the second operand subtracted from the first.
3. **Multiplication(*):** The result of the multiplication operator is the product of the two operands. *For example*, 4 * 4 is 16.
4. **Division (/):** The result of the division operator is the quotient of the two operands. Dividing two integers can give an integer (e.g., 14/2) or a floating-point result (e.g., 1/2).
5. **Modulus (%):** The modulus operator converts both operands to integers and returns the remainder of the division of the first operand by the second operand. *For example*, 10 % 6 is 4.
6. **Arithmetic negation (-):** The arithmetic negation operator returns the operand multiplied by -1, effectively changing its sign. *For example*, -(3 - 4) evaluates to 1. Arithmetic negation is different from the subtraction operator, even though they both are written as a minus sign. Arithmetic negation is always unary and before the operand. Subtraction is binary and between its operands.
7. **Arithmetic assertion (+):** The arithmetic assertion operator returns the operand multiplied by +1, which has no effect. It is used only as a visual cue to indicate the sign of a value. *For example*, +(3 - 4) evaluates to -1, just as (3 - 4) does.

String Concatenation Operator

Manipulating strings is such a core part of PHP applications that PHP has a separate string concatenation operator (.). The concatenation operator appends the right-hand operand to the left-hand operand and returns the resulting string. Operands are first converted to strings, if necessary.

For example

```
$n =15;
$s = 'years of completion' . $n . ' of work.';
// $s is 'years of completion 15 of work.'
```

Autoincrement and Autodecrement Operators

One of the most common operations is to increase or decrease the value of a variable by one. The unary autoincrement (++) and autodecrement (--) operators provide shortcuts for these common operations.

These operators are unique in that they work only on variables; the operators change their operands' values as well as returning a value. There are two ways to use autoincrement or autodecrement in expressions. If we put the operator in front of the operand, it returns the new value of the operand (incremented or decremented). If we put the operator after the operand, it returns the original value of the operand (before the increment or decrement).

These operators can be applied to strings as well as numbers. Incrementing an alphabetic character turns it into the next letter in the alphabet.

Comparison Operators

Comparison operators compare operands. The result is always either true, if the comparison is truthful, or false, otherwise. Operands to the comparison operators can be both numeric, both string, or one numeric and one string.

The comparison operators are

1. **Equality (==):** If both operands are equal, this operator returns true; otherwise, it returns false.

2. **Identical (===):** If both operands are equal and are of the same type, this operator returns true; otherwise, it returns false. Note that this operator does not do implicit type casting. This operator is useful when you don't know if the values you're comparing are of the same type. Simple comparison may involve value conversion. For instance, the strings "0.0" and "0" are not equal.

   The == operator says they are, but === says they are not.

3. **Inequality(!= or <>):** If both operands are not equal, this operator returns true; otherwise, it returns false.

4. **Not identical (!==):** If both operands are not equal, or they are not of the same type, this operator returns true; otherwise, it returns false.

5. **Greater than (>):** If the lefthand operator is greater than the righthand operator, this operator returns true; otherwise, it returns false.

6. **Greater than or equal to (>=):** If the lefthand operator is greater than or equal to the righthand operator, this operator returns true; otherwise, it returns false.

7. **Less than (<):** If the lefthand operator is less than the righthand operator, this operator returns true; otherwise, it returns false.

8. **Less than or equal to (<=):** If the lefthand operator is less than or equal to the righthand operator, this operator returns true; otherwise, it returns false.

The Bitwise Operators

Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then calculation is performed on the operands. The mathematical operations such as addition, subtraction, multiplication etc. can be performed at bit-level for faster processing.

i. **& (Bitwise AND):** This is a binary operator, i.e., it works on two operands. Bitwise AND operator in PHP takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

Syntax

```
$First & $Second
```

This will return another number whose bits are set, if both the bit of first and second are set.

Example: Input: $First = 5, $Second = 3

Output: The bitwise & of both these value will be 1.

Binary representation of 5 is 0101 and 3 is 0011.Therefore their bitwise & will be 0001 (i.e. set if both first and second have their bit set.)

ii. **| (Bitwise OR):** This is also binary operator, i.e., works on two operands. Bitwise OR operator takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

Syntax

```
$First | $Second
```

This will return another number whose bits are set if either the bit of first or second are set.

Example: Input: First = 5, Second = 3

Output: The bitwise | of both these value will be 7.

Binary representation of 5 is 0101 and 3 is 0011.Therefore their bitwise | will be 0111 (i.e., set if either first or second have their bit set.)

iii. **^ (Bitwise XOR):** This is also binary operator, i.e., works on two operands. This is also known as Exclusive OR operator. Bitwise XOR takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

Syntax

```
$First ^ $Second
```

This will return another number whose bits are set if one of the bit in first or second is set but not both.

Example: Input: First = 5, Second = 3

Output: The bitwise ^ of both these values will be 6.

Binary representation of 5 is 0101 and 3 is 0011. Therefore their bitwise ^ will be 0110 (i.e., set if either first or second have their bit set but not both.)

iv. **~ (Bitwise NOT):** This is a unary operator, i.e., works on only one operand. Bitwise NOT operator takes one number and inverts all bits of it.

Syntax

```
~$number
```

This will invert all the bits of $number.

Example: Input: number = 5

Output: The bitwise '~' of this number will be -6.

Binary representation of 5 is 0101. Therefore the bitwise ~ of this will be 1010 (inverts all the bits of the input number)

v. **<< (Bitwise Left Shift):** This is a binary operator, i.e., works on two operand.s Bitwise Left Shift operator takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

Syntax

```
$First << $Second
```

This will shift the bits of $First towards the left. $Second decides the number of time the bits will be shifted.

Example: Input: First = 5, Second = 1

Output: The bitwise << of both these value will be 10.

Binary representation of 5 is 0101. Therefore, bitwise << will shift the bits of 5 one time towards the left (i.e., 01010).

Note: Bitwise left shift with one bit is equivalent to multiplication with 2.

vi. **>> (Bitwise Right Shift):** This is also binary operator, i.e., works on two operands. Bitwise Right Shift operator takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

Syntax

```
$First >> $Second
```

This will shift the bits of $First towards the right. $Second decides the number of time the bits will be shifted.

Example: Input: First = 5, Second = 1

Output: The bitwise >> of both these value will be 2.

Binary representation of 5 is 010. Therefore, bitwise >> will shift the bits of 5 one time towards the right(i.e., 010).

vi. **Logical Operators:** Logical operators provide ways for you to build complex logical expressions. The logical operators are used to combine conditional statements. Logical operators treat their operands as Boolean values and return a Boolean value.

Note: There are both punctuation and English versions of the operators (|| and or are the same operator).

The logical operators are:

Operator	Name	Example	Result
and(&&)	And	$x and $y $x && $y	True if both $x and $y are true.
Or(||)	Or	$x or $y $x || $y	True if either $x or $y is true.
Xor	Xor	$x xor $y	True if either $x or $y is true, but not both.
!	Not	!$x	True if $x is not true.

vii. **Casting operators:** The casting operators, (int), (float), (string), (bool), (array), and (object), allow us to force a value into a particular type. To use a casting operator, put the operator to the left of the operand.

Casting affects the way other operators interpret a value, rather than changing the value in a variable. For example,

```
$a = "15";
$b = (int) $a;
```

assigns $b the integer value of $a; $a remains the string "15".

To cast the value of the variable itself, we must assign the result of a cast back into the variable:

```
$a = "15";
$a = (int) $a; // now $a holds an integer
```

vii. **Assignment Operators:** The assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right.
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

viii. **Miscellaneous Operators:** The remaining PHP operators are for error suppression, executing an external command, and selecting values:

- Error suppression (@): Some operators or functions can generate error messages. The error suppression operator is used to prevent these messages from being created.
- Execution (`...`): The backtick operator executes the string contained between the backticks as a shell command and returns the output.

For example

```
$list = `ls -ls /tmp`; echo $list;
```

- Conditional (?:): The conditional operator is, depending on the code we look at, either the most overused or most underused operator. It is the only ternary (three-operand) operator and is therefore sometimes just called the ternary operator. The conditional operator evaluates the expression before the ?. If the expression is true, the operator returns the value of the expression between the ? and :; otherwise, the operator returns the value of the expression after the : .

Example

```
$a=10;
$b=20;
// If condition is true then assign a to result otheriwse b
$result=($a>$b)?$a:$b;
```

ix. **PHP delimiters:** A `delimiter` is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data stream.

```
$a = "PHP";
$b = 'Java';
```

The single and double characters are used to mark the beginning and the end of a string.

```
function setDate($date)
{
    $this->date = $data;
}
if($a > $b)
{  echo "\$a is bigger than \$b"; }
```

Parentheses are used to mark the function signature. The signature is the function parameters. Curly brackets are used to mark the beginning and the end of the function body. They are also used in flow control.

```
$a = array(1, 2, 3);
echo $a[1];
```

The square brackets are used to mark the array index.

```
/*
Publication vision
January 2019
Innovation throughout
*/
/* */ delimiters are used to provide C style comments in PHP.
<?php
// PHP code
```

The <?php delimiter is used to declare the start of PHP code and and ?> to end the php code.

Keywords

A keyword is a word reserved by the language for its core functionality—we cannot give a variable, function, class, or constant the same name as a keyword.

Table below lists the keywords in PHP, which are case-insensitive.

PHP core language keywords .

Table PHP core language keywords

and	$argc	$argv	as
break	case	cfunction	class
continue	declare	default	die
do	E_All	echo	E_ERROR
else	Elseif	empty	enddeclare
endfor	Endforeach	endif	endswitch
E_PARSE	Eval	E_WARNING	exit
extends	FALSE	for	foreach
function	$HTTP_COOKIE_VARS	$HTTP_ENV_VARS	$HTTP_GET_VARS
include	include_once	global	list
new	Not	NULL	old_function
or	Parent	PHP_OS	$PHP_SELF
PHP_VERSION	Print	require	require_once
return	Static	stdclass	switch
$this	TRUE	var	Virtual
while	Xor	__FILE__	__LINE__
__sleep	__wakeup	$_COOKIE	$_ENV
$_FILES	$_GET	$_POST	$_SERVER

Case Sensitivity

The names of user-defined classes and functions, as well as built-in constructs and keywords such as echo, while, class, etc., are case-insensitive. Thus, these three lines are equivalent:

```
echo("hello, Vision publications");
ECHO("hello, Vision publications ");
EcHo("hello, Vision publications ");
```

Variables, on the other hand, are case-sensitive. That is, $name1, $NAME1, and $NaME1 are three different variables.

5. Language Basics

Data Types

PHP provides eight types of values, or data types. Four are scalar (single-value) types: integers, floating-point numbers, strings, and booleans. Two are compound (collection) types: arrays and objects.

The remaining two are special types: resource and NULL.

1. Integers

Integers are whole numbers, like 1, 10, and 256.

Integer literals can be written in decimal, octal, or hexadecimal. Decimal values are represented by a sequence of digits, without leading zeros. The sequence may begin with a plus (+) or minus (-) sign.

Examples of decimal integers include the following:

```
2021
-651
+21
```

Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Like decimal numbers, octal numbers can be prefixed with a plus or minus. Here are some *examples of* octal values and their equivalent decimal values:

```
0755      // decimal  493
+010      // decimal  8
```

Hexadecimal values begin with 0x, followed by a sequence of digits (0-9) or letters (A-F). The letters can be upper- or lowercase but are usually written in capitals. Like decimal and octal values, we can include a sign in hexadecimal numbers:

```
0xFF      // decimal 255
0x10      // decimal 16
-0xDAD1   // decimal -56017
```

2. Floating-Point Numbers

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. PHP recognizes floating-point numbers written in two different formats.

```
3.14 0.017 -7.1 but PHP also recognizes numbers in scientific
               notation:
0.314E1     // 0.314*101, or 3.14
17.0E-3     // 17.0*10-3, or 0.017
```

3. Strings

A string is a sequence of characters of arbitrary length. String literals are delimited by either single or double quotes:

```
'vision publications'
"vision publications"
```

Variables are expanded within double quotes, while within single quotes they are not:

```
$name = "VisionPublications";
echo "Welcome to, $name\n";     // Welcome to ,VisionPublications
echo 'Welcome to, $name';       // Welcome to, $name
```

Double quotes also support a variety of string escapes like: newline(\n), dollar sign(\$),left brace(\{) etc.

4. Booleans

A boolean value represents a "truth value"—it says whether something is true or not. Like most programming languages, PHP defines some values as true and others as false.

Truth and falseness determine the outcome of conditional code such as:

```
if($num) { ... }
```

In PHP, the following values are false:

The keyword false

The integer 0

The floating-point value 0.0

The empty string ("") and the string "0"

An array with zero elements

An object with no values or functions

The NULL value

Any value that is not false is true, including all resource values.

5. Arrays

An array holds a group of values, which you can identify by position (a number, with zero being the first position) or some identifying name (a string):

```
$sub[0] = "c";
$sub[1] = "c++";
$sub[2] = "DS";
```

The array( ) construct creates an array:

```
$sub = array('c', 'c++', 'DS');
```

6. Objects

PHP supports object-oriented programming (OOP). Classes are the unit of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the class keyword:

```
class emp
{
 var $name = '';
 function name($newname = NULL)
 {
  if(! is_null($newname)) { $this->name = $newname; }
  return $this->name;
 }
 }
```

Once a class is defined, any number of objects can be made from it with the new keyword, and the properties and methods can be accessed with the -> construct:

```
$obj = new emp;
$obj->name('ram');
printf("Hello, %s\n", $obj->name);
$obj2 = newemp;
$obj2->name('shyam');
printf("List of employees %s\n", $obj2->name);
```

Hello, Ram List of employees shyam.

The is_object() function is used to test whether a value is an object:

```
if(is_object($x)) { // $x is an object }
```

7. Resources

A resource is a special variable, holding a reference to an external resource.

Resource variables typically hold special handlers to opened files and database connections.

```
<?php
// Open a file for reading
$handle = fopen("try.txt", "r");
var_dump($handle);
echo "<br>";
?>
```

Output

resource(2) of type (stream)

8. NULL

Null is a special data type that has only one value: NULL. There is a convention of writing it in capital letters as it is case sensitive.

NULL is defined as variable with no value.

```
<?php
    $num = NULL;
    echo $num;    //it will not give any output
?>
```

5.1 Flow-Control Statements

PHP supports conditional statements, such as if/else and switch, allows a program to execute different pieces of code, or none at all, depending on some condition. Loops, such as while and for, support the repeated execution of particular code.

1. **if statement :** Executes some code if one condition is true.

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

The ternary conditional operator (?:) can be used to shorten simple true/false tests. Compare the syntax of the two:

```
 if(expression) true_statement else false_statement
(expression)? true_expression : false_expression
```

The main difference here is that the conditional operator is not a statement at all. This means that it is used on expressions, and the result of a complete ternary expression is itself an expression.

- if...else statement: Executes some code if a condition is true and another code if that condition is false.
- if...elseif...else statement: Executes different codes for more than two conditions.

2. **Switch statement:** Selects one of many blocks of code to be executed.

Syntax

```
switch(n)
{
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
    break;
  case label3:
    code to be executed if n=label3;
    break;
    ...
  default:
    code to be executed if n is different from all labels;
}
```

A switch statement is given an expression and compares its value to all cases in the switch; all statements in a matching case are executed, up to the first break keyword it finds. If none match, and a default is given, all statements following the default keyword are executed, up to the first break keyword encountered.

3. **The while loop:** Loops through a block of code as long as the specified condition is true.

Syntax

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

The *example* below displays the numbers from 10 to 15:

```
<?php
$x = 10;
while($x <=15)
{
  echo "The number is: $x <br>";
  $x++;
}
?>
```

4. **Do....while loop:** PHP also supports a do /while loop, which takes the following form:

```
do statement
while(expression)
```

A do/while loop is used to ensure that the loop body is executed at least once.

```
$total = 0;
$i = 1;
do
{
 $total += $i++;
} while($i <= 10);
echo $total;
```

The *example* above first sets a variable $total to 0 and counter variable $i to 1 ($i = 1). Then, the do while loop will add the total with the counter $i , and then increment the variable $i with 1. Then the condition is checked (is $i less than, or equal to 10?), and the loop will continue to run as long as $x is less than, or equal to 10

```
The following code outputs: 55
```

We can use break and continue statements in a do/while statement just as in a normal while statement.

5. **for loop**: Loops through a block of code a specified number of times.

Syntax

```
for(init counter; test counter; increment counter)
{
  code to be executed for each iteration;
}
```

Parameters

init counter: Initialize the loop counter value.

test counter: Evaluates for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

increment counter: Increases the loop counter value.

The example below displays the numbers from 10 to 20.

```
<?php
for($x = 10; $x <= 20; $x++)
{
  echo "The number is: $x <br>";
}
?>
```

6. **foreach:** 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;
}
```

The following *example* will output both the keys and the values of the given array ($age):

```
<?php
$age = array("shahrukh"=>"35", "amitabh"=>"67", "ranbir"=>"28");
foreach($age as $x => $val)
{
  echo "$x = $val<br>";
}
?>
```

Output

shahrukh = 35
amitabh = 67
ranbir = 28

7. **Declare:** The declare statement allows us to specify execution directives for a block of code. The structure of a declare statement is:

```
declare (directive)
  statement;
```

8. **exit and return:** The exit statement ends execution of the script as soon as it is reached. The return statement returns from a function or (at the top level of the program) from the script. The exit statement takes an optional value. If this is a number, it's the exit status of the process. If it's a string, the value is printed before the process terminates. The exit() construct is an alias for die().

6. **foreach:** 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;
}
```

The following *example* will output both the keys and the values of the given array ($age):

```
<?php
$age = array("shahrukh"=>"35", "amitabh"=>"67", "ranbir"=>"28");
foreach($age as $x => $val)
{
  echo "$x = $val<br>";
}
?>
```

Output

shahrukh = 35
amitabh = 67
ranbir = 28

7. **Declare:** The declare statement allows us to specify execution directives for a block of code. The structure of a declare statement is:

```
declare (directive)
  statement;
```

8. **exit and return:** The exit statement ends execution of the script as soon as it is reached. The return statement returns from a function or (at the top level of the program) from the script. The exit statement takes an optional value. If this is a number, it's the exit status of the process. If it's a string, the value is printed before the process terminates. The exit() construct is an alias for die().

SUMMARY

- HTTP (Hypertext Transfer Protocol) is a stateless protocol.
- A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web.
- The client sends a *request message* to the server. The server, in turn, returns a *response message*.
- Web server is a computer where the web content is stored.
- A web browser is a software application which enables a user to display and interact with text, images, videos, music, and other information that could be on a website.
- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is mainly focused on server-side scripting, command line scripting, writing desktop applications.
- PHP is an interpreted language: it means that PHP will need to interpret and compile the code of our application for each request made to it.
- PHP comments : #, /*...*/ and //.
- Variable names start with $sign.
- There are four types of variable scope in PHP: local, global, static, and function.
- Function names are not case-sensitive.
- Constants are referred to by their identifiers and are set using the define() function.
- PHP has a separate string concatenation operator (.).
- The comparison operator identical (===) checks if both operands are equal and are of the same type, this operator returns true, otherwise, it returns false.
- PHP supports following data types: Four are scalar (single-value) types: integers, floating-point numbers, strings, and booleans. Two are compound (collection) types: arrays and objects. Special type (NULL,Resource)
- PHP supports conditional statements, such as if/else and switch, allows a program to execute different pieces of code, or none at all, depending on some condition.
- The for each 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, until it reaches the last array element (added)

Exercise

1. What is HTTP?
2. Explain HTTP request and HTTP response?
3. What is URL?
4. Explain web browser.
5. Explain the working of web server.
6. Differentiate between web browser and web server.
7. Explain any two web browsers
8. Explain any two web servers.
9. Explain the different features of PHP.
10. What PHP can do?
11. How PHP works?
12. Why PHP is called Loosely Typed Language.
13. PHP is an interpreted language justify true/false.
14. Explain variable scope in PHP.
15. How constants are defined in PHP.
16. Explain String Concatenation Operator in PHP.
17. Explain any two Comparison Operators.
18. How does equality and identical operator differ.
19. What are bitwise operators.
20. Explain logical operators with examples.
21. What are scalar data type in PHP?
22. What are compound data type.

23. Explain special data type in PHP.

24. Explain in detail data type of PHP.

25. What are the different flow control statements available in PHP.

26. What are different looping constructs supported in PHP with examples.

27. Explain foreach construct with an example.

28. Give syntax and example of switch statement.