UNIT-1

PHP:

PHP Hypertext Preprocessor is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. It allows web develop to create dynamic content that interacts with databases.

History of PHP:

PHP is Created in 1994 by Rasmus Lerdorf. Initially php it is used for track online visitors on web.

Basic Points about PHP:

- PHP stands for HyperText Preprocessor.
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP files have extension ".php"
- PHP code is executed on the server so you need web browser to execute php code
- It is an interpreted language, i.e. there is no need for compilation.
- It is a server-side scripting language.
- It is faster than other scripting language e.g. asp and jsp.

Advantage of PHP:

- It is open source.
- Widely used in all over the world
- Free to download
- It is executed on the server
- To execute php code no need compiler.

Why PHP and MYSQL?

PHP is the most popular scripting language for web development. It is free, open source and server-side (the code is executed on the server). MySQL is a Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). It is also free and open source. The combination of PHP and MySQL gives unmet options to create just about any kind of website - from small contact form to large corporate portal.

- Following is considered to be the six most important things in modern PHP web development. (BASIC DEVELOPMENT CONCEPTS)

  - 1. Code: Code is the most important or says foundation for every development process and so is with development of PHP applications. The standards of coding have been important right from the early times and continue to be so even in this modern world of web development. Maintenance of standards in coding will add consistency to your development project and hence, ensure its comprehended ability so that other team members can easily understand and append or modify as per requirements.

  - 2. Testing: Testing is an important requirement in development process because a thoroughly tested application has many advantages like it is easy to refactor the tested software or application, increased re-usability of the test assured code and last but not the least good code without any future headaches.

  - 3. Dependencies: Composer is a good tool for dependency handling. The dependency declaration should also be under version control because code is strongly coupled with package or library. But the files of this package or library should not be part of version control because the revisions of them are handled by the declaration which is already under the version control. So no need to do both.

  - 4. Configuration: Good configuration of your application code is essential for migrating your PHP application or else it can pose security threats. So, it is a good practice to separate configuration from code.

  - 5. Runtime Data: The runtime data typically includes the one related to the file and not the databases. This is generated at runtime with user interaction methods like file uploads. This is usually handled by a local or network attached file system. Cloud storage is recommended instead of this in order to handle the scalability issues that arise during scaling an application out over multiple servers or geographical regions or prevent future migration headaches.

  - 6. Deployment: Simple, transparent and fast deployment is usually the best way to go for a web application project in order to prevent the things that are more feared and disaster prone. Go for deployment process or mechanism that allows integration of built scripts

easily. All in all, an easy-to-use deployment workflow is essential besides good quality code and testing measures.

Unique Features

If you're familiar with other server-side languages like ASP.NET or JSP, you might be wondering what makes PHP so special, or so different from these competing alternatives. Well, here are some reasons:

Performance Scripts written in PHP execute faster than those written in other scripting languages, with numerous independent benchmarks putting the language ahead of competing alternatives like JSP, ASP.NET, and Perl. The PHP 5.0 engine was completely redesigned with an optimized memory manager to improve performance, and is noticeably faster than previous versions. In addition, third-party accelerators are available to further improve performance and response time.

Portability PHP is available for UNIX, Microsoft Windows, Mac OS, and OS/2, and PHP programs are portable between platforms. As a result, a PHP application developed on, say, Windows will typically run on UNIX without any significant issues. This ability to easily undertake cross-platform development is a valuable one, especially when operating in a multiplatform corporate environment or when trying to address multiple market segments.

Ease of Use "Simplicity is the ultimate sophistication," said Leonardo da Vinci, and by that measure, PHP is an extremely sophisticated programming language. Its syntax is clear and consistent, and it comes with exhaustive documentation for the 5000+ functions included with the core distributions. This significantly reduces the learning curve for both novice and experienced programmers, and it's one of the reasons that PHP is favoured as a rapid prototyping tool for Web-based applications.

Open Source PHP is an open-source project—the language is developed by a worldwide team of volunteers who make its source code freely available on the Web, and it may be used without payment of licensing fees or investments in expensive hardware or software. This reduces software development costs without affecting either flexibility or reliability. The open-source nature of the code further means that any developer, anywhere, can inspect the code tree, spot errors, and suggest possible fixes; this produces a stable, robust product wherein bugs, once discovered, are rapidly resolved—sometimes within a few hours of discovery!

Community Support One of the nice things about a community-supported language like PHP is the access it offers to the creativity and imagination of hundreds of developers across the world. Within the PHP community, the fruits of this creativity may be found in PEAR, the PHP Extension and Application Repository (http://pear.php.net/), and PECL, the PHP Extension Community Library (http://pecl.php.net/), which contains
hundreds of ready-made widgets and extensions that developers can use to painlessly add new functionality to PHP. Using these widgets is often a more time- and cost-efficient alternative to rolling your own code.

Third-Party Application Support One of PHP's strengths has historically been its support for a wide range of different databases, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. PHP 5.3 supports more than fifteen different database engines, and it includes a common API for database access. XML support makes it easy to read (and write) XML documents as though they were native PHP data structures, access XML node collections using XPath, and transform XML into other formats with XSLT style sheets.

Creating first PHP Scripts:

PHP Syntax:

You can run php code on any web browser. PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic Syntax of PHP

•PHP code is start with

- All PHP statements end with a semicolon (;)

.• PHP code save with .php extension.

- PHP contain some HTML tag and PHP code.
- You can place PHP code anywhere in your document.

PHP Syntax:

```
<?php
//statements;
?>
```

PHP files save with .php extension and it contain some HTML and PHP code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
```

```
?>

</body>
</html>
```

- **Note:** PHP statements end with a semicolon (;).

PHP Case Sensitivity

In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.
In the example below, all three echo statements below are equal and legal:

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

- **Note:** However; all variable names are case-sensitive!

Look at the example below; only the first statement will display the value of the $color variable! This is because $color, $COLOR, and $coLOR are treated as three different variables:

```
<!DOCTYPE html>
<html>
<body>

<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

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

- Output:

```
My car is red
My house is
My boat is
```

Comments in PHP

- A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.
- Comments can be used to:
- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting

-
```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single-line comment

# This is also a single-line comment
?>
</body>
</html>
```

- ```
  <!DOCTYPE html>
  <html>
  <body>

  <?php
  /*
  This is a multiple-lines comment block
  that spans over multiple
  lines
  */
  ?>

  </body>
  </html>
  ```

- ```
  <!DOCTYPE html>
  <html>
  <body>

  <?php
  // You can also use comments to leave out parts of a code line
  $x = 5 /* + 15 */ + 5;
  echo $x;
  ?>

  </body>
  </html>
  ```
- Output:

10

Variables & storing data in variables

- Variables in PHP are identifiers prefixed with a dollar sign ($). A variable may hold a value of any type.
- A *variable* is simply a container that's used to store both numeric and non-numeric information.

ex: $name, $Age , $_debugging, $MAXIMUM_IMPACT

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

```
<html>
<body>
<?php
$a = 25; // Numerical variable
$b = 'Hello'; // String variable
$c = 5.7; // Float variable
echo "Number is : $a<br/>";
echo "String is : $b<br/>";
echo "Float value : $c";
?>
</body>
<html>
```

OUTPUT :

Number is : 25

String is : Hello

Float value : 5.7

Assigning Values to Variables

Assigning a value to a variable in PHP is quite easy: use the equality (=) symbol, which also happens to be PHP's assignment operator. This assigns the value on the right side of the equation to the

variable on the left. To use a variable in a script, simply call it by name in an expression and PHP will replace it with its value when the script is executed. Here's an example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head><title /></head>
<body>
<?php
// assign value to variable
$name = 'Simon';
?>
<h2>Welcome to <?php echo $name; ?>'s Blog!</h2>
</body></html>
```

In this example, the variable $name is assigned the value 'Simon'. The echo statement is then used to print the value of this variable to the Web page. You can also assign a variable the value of another variable, or the result of a calculation. The following example demonstrates both these situations:

```
<?php
// assign value to variable
$now = 2008;
// assign variable to another variable
$currentYear = $now;
// perform calculation
$lastYear = $currentYear - 1;
// output: '2007 has ended. Welcome to 2008!'
echo "$lastYear has ended. Welcome to $currentYear!";
?>
```

Variable

- You can reference the value of a variable whose name is stored in another variable. For example:

```
$foo = 'bar';

$$foo = 'baz';
```

- After the second statement executes, the variable $bar has the value "baz".

Variable References

- In PHP, references are how you create variable aliases. To make $black an alias for the variable $white, use:

  $black =& $white;

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

Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

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.
- Ex:

```
<?php
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

Global scope

- Variables declared outside a function are global.
- That is, they can be accessed from any part of the program.

- However, by default, they are not available inside functions. To allow a function to access a global variable, you can use the global keyword inside the function to declare the variable within the function

```
<?php
$x = 5;
$y = 10;

function myTest() {
  global $x, $y;
  $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>

<html>

<body>

<?php

$x=24; // global scope

// Function definition

function myFunction() {

$y=59; // local scope

echo "Variable x is: $x <br>";

echo "Variable y is: $y";

}

myFunction();// Function call

echo "Variable x is: $x";

echo "<br>";

echo "Variable y is: $y";

?>

</body>
```

```
</html>
```

OUTPUT of the above given Example is as follows:

Variable x is:

Variable y is: 59

Test variables outside the function:

Variable x is: 24 Variable y is:

Static variables

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. To do this, use the static keyword when you first declare the variable.

A static variable retains its value between calls to a function but is visible only within that function. You declare a variable static with the static keyword.

Each time the function is called, that variable will still have the information it contained from the last time the function was called.

Example on static variable

```
<html>
<body>
<?php
// Function definition
function myFunction() {
static $x=45;
echo $x;
echo "<br/>";
$x++;
}
myFunction();
myFunction();
```

```
myFunction();

myFunction();

myFunction();

?>

<body>

<html>
```

OUTPUT of the above given Example is as follows:

45 46 47 48 49

Operators & Manipulating variables with operators

- Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- **Arithmetic Operators**

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example
+	Addition	$x + $y
-	Subtraction	$x - $y

*	Multiplication	$x * $y
/	Division	$x / $y
%	Modulus	$x % $y
**	Exponentiation	$x ** $y

```
<html>
<body>
<?php
$i=(20 + 10);
$j=($i - 5);
$k=($j * 4);
$l=($k / 2);
$m=($l % 5);
echo "i = ".$i."<br/>";
echo "j = ".$j."<br/>";
echo "k = ".$k."<br/>";
echo "l = ".$l."<br/>";
echo "m = ".$m."<br/>";
?>
</body>
</html>
```

OUTPUT of the above given Example is as follows:

i = 30 j = 25 k = 100 l = 50 m = 0

Assignment Operators

- The PHP 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

```
<html>
<body>
<?php
$a=5;
echo "a=".$a;   echo "<br/>";
$b=10;
$b += 20;
echo "b=".$b;    echo "<br/>";
$c=15;
$c -= 5;
echo "c=".$c;    echo "<br/>";
$d=20;
```

```
$d *= 2;
echo "d=".$d;   echo "<br/>";
$e=25;
$e /= 5;
echo "e=".$e;    echo "<br/>";
$f=30;
$f %= 4;
echo "f=".$f;
?>
</body>
</html>
```

OUTPUT of the above given Example is as follows:

a=5 b=30 c=10 d=40 e=5 f=2

Comparison Operators

- The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y

<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>= / <=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y

- ```
  <html>
  <body>
  <?php
  $x = 100;
  $y = "100";
  var_dump($x === $y); // returns false because types are not equal
  var_dump($x !== $y); // returns true because types are not equal
  ?>
  </body>
  </html>
  ```

- **Output:**

```
bool(false)
bool(true)
```

Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
```

++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

```
<?php
$i=10;
$j=20;
$i++;
$j++;
echo $i."<br/>";
echo $j."<br/>";
$k=$i++;      // Post increment
$l=++$j;      // Pre increment
echo $k."<br/>";
echo $l;
?>
```

OUTPUT

11 21 11 22

Logical Operators

- The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result

And	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $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
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

```
<html>
<body>
<?php
$x = 100;
$y = 50;
if ($x == 100 and $y == 50) {
   echo "Hello world!";
}
?>
</body>
</html>
```

- **Output:**

Hello world!

String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result

.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

```
<html>
<body>
<?php
$a = "Hello";
$b = $a . " Friend!";
echo $b;
echo "<br/>";
$c="Good";
$c .= " Day!";
echo $c;
?>
</body>
</html>
```

OUTPUT

Hello Friend!

Good Day!

Array Operators

The PHP array operators are used to compare arrays.

Example of array operator

```
<html>
<body>
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");
print_r($x + $y); // union of $x and $y

    var_dump($x == $y);

    var_dump($x != $y);
?>
</body>
</html>
```

- output:

```
Array ( [a] => red [b] => green [c] => blue [d] => yellow )

bool(false)

bool(true)
```

Data Types & Understanding DataTypes

- 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.

Integers

- Integers are whole numbers, such as 1, 12, and 256.
- An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
- Rules for integers:
- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative

- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation
- Example:

```
<html>

<body>

<?php
$x = 5985;
var_dump($x);

?>

</body>

</html>
```

Output:

```
int(5985)
```

Floating-Point Numbers

- Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits.
- Usually, this allows numbers between 1.7E-308 and 1.7E+308 with 15 digits of accuracy.

  Ex: 3.14, 0.017, -7.1
- Floating-point values are only approximate representations of numbers.
- Use the is_float( ) function (or its is_real( ) alias) to test whether a value is a floating-point number:

```
if (is_float($x))

 {

// $x is a floating-point number

}
```

- In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<html>
<body>
<?php
$x = 10.365;
var_dump($x);
?>
</body>
</html>
```

Output:

float(10.365)

Strings

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

Ex: 'BCA' "College"

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

Ex:

$name = "Sara";

echo "Hi, $name\n";

echo 'Hi, $name';

Hi, Sara

Hi, $name

Ex:

```
<html>
<body>
<?php
$x = "Hello world!";
$y = 'Hello world!';
echo $x;
echo "<br>";
echo $y;
?>
```

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

- **Output:**
  Hello world!
  Hello world!

Escape sequences in double-quoted strings

Escape sequence	Character represented
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\$	Dollar sign
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket

- carriage return is **a special code that moves the cursor (or print head) to the beginning of the current line**

Booleans

- A Boolean value represents a "truth value "it says whether something is true or not.
- In PHP, the following values all evaluate to 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

Arrays

- An array stores multiple values in one single variable.
- 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), called an associative:
- $person[0]="Edison";
- $person[1]="Zony";
- $person[2]="Zaze";
- In the following example $cars is an array. The PHP var_dump() function returns the data type and value:
-
```
<html>
<body>

<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>

</body>
</html>
```

- **Output:**
- array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:
- **Example on object data type**

```
<html>
<body>
<?php
class Car {
  function Car() {
    $this->model = "VW";
  }
}
// create an object
$herbie = new Car();
// show object properties
echo $herbie->model;
?>
</body>
</html>
```

- output:

VW

NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.

```
<html>
<body>
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
</body>
</html>
```

- **Output:**

null

Understanding PHP's Data Types

The values assigned to a PHP variable may be of different *data types,* ranging from simple string and numeric types to more complex arrays and objects. You've already seen two of these, strings and numbers, in action in previous examples. Here's a full-fledged example, which introduces three more data types:

```
<?php
// Boolean
$validUser = true;
// integer
$size = 15;
// floating point
$temp = 98.6;
// string
$cat = 'Siamese';
// null
$here = null;
?>
```

- *Booleans* are the simplest of all PHP data types. Like a switch that has only two states, on and off, it consists of a single value that may be set to either 1 (true) or 0 (false). In this listing, $validUser is a Boolean variable set to true.
- PHP also supports two numeric data types: *integers* and *floating-point values.* Floatingpoint values (also known as *floats* or *doubles*) are decimal or fractional numbers, while integers are round numbers. Both may be less than, greater than, or equal to zero. In this listing, $size holds an integer value, while $temp holds a floating-point value.
- For non-numeric data, PHP offers the *string* data type, which can hold letters, numbers, and special characters. String values must be enclosed in either single quotes or double quotes. In the previous listing, $cat is a string variable containing the value 'Siamese'.
- You may also encounter the NULL data type, which is a “special” data type first introduced in PHP 4. NULLs are used to represent “empty” variables in PHP; a variable of type NULL is a variable without any data. In the preceding listing, $here is NULL.

Setting and checking variables – Data Types

Unlike other programming languages, where a variable’s data type must be explicitly defined by the programmer, PHP automatically determines a variable’s data type from the content it holds. And if the variable’s content changes over the duration of a script, the language will automatically set the variable to the appropriate new data type.
Here’s an example which illustrates this *type juggling*:

```
<?php
// define string variable
$whoami = 'Sarah';
// output: 'string'
echo gettype($whoami);
// assign new integer value to variable
$whoami = 99.8;
```

```
// output: 'double'
echo gettype($whoami);
// destroy variable
unset($whoami);
// output: 'NULL'
echo gettype($whoami);
?>
```

This example introduces PHP's gettype() operator, which is a handy little tool for finding out the type of a particular variable. As the script output demonstrates, the variable $whoami begins life as a string, assigned the value 'Sarah'. It's then assigned the number 99.8, which automatically converts it to a floating-point variable. Following this, the variable is de-initialized with the unset() method, which removes its value and turns it into a NULL. PHP has been the invisible hand behind each of these conversions, internally resetting the data type of $whoami from string to floating-point to null.

This doesn't mean that you're entirely at PHP's mercy, however; it's possible to explicitly set the type of a PHP variable by *casting* a variable to a specific type before using it. Casting is a technique commonly used by Java programmers; to use it, simply specify the desired data type in parentheses on the right side of the assignment equation. Consider the following example, which illustrates turning a floating-point value into an
integer value:

```
<?php
// define floating-point variable
$speed = 501.789;
// cast to integer
$newSpeed = (integer) $speed;
// output: 501
echo $newSpeed;
?>
```

In addition to the gettype() function, PHP includes a number of more specialized functions, to test if a variable is of a specific type. Table 2-1 has a list.

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

Table 2-1 PHP Functions to Test Variable Data Types

Using Constants

So far, this chapter has focused mainly on variables, which are good for storing and changing values over the lifetime of a PHP script. But what if you need to store a fixed value, one that remains static over the course of a script? Well, that's when you reach for a *constant.*

As the name suggests, constants are PHP containers for values that remain constant and never change. They're mostly used for data that is known well in advance and that is used, unchanged, in multiple places within your application. Good candidates for constants are debug and log levels, version numbers, configuration flags, and formulae.

Constants are defined using PHP's define() function, which accepts two arguments: the name of the constant, and its value. Constant names must follow the same rules as variable names, with one exception: the $ prefix is not required for constant names.
Here's an example of defining and using a constant in a script:

```
<?php
// define constants
define ('PROGRAM', 'The Matrix');
define ('VERSION', 11.7);
// use constants
// output: 'Welcome to The Matrix (version 11.7)'
echo 'Welcome to ' . PROGRAM . ' (version ' . VERSION . ')';
?>
```

Manipulating Variables with Operators

By themselves, variables are simply containers for information. In order to do anything useful with them, you need *operators.* Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than (>) symbol is an operator that tells PHP to compare two values.

PHP supports more than 50 such operators, ranging from operators for arithmetical operations to operators for logical comparison and bitwise calculations. This section discusses the most commonly used operators.

Performing Arithmetic Operations

PHP supports all standard arithmetic operations, as illustrated by the list of operators in Table 2-2.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide and return quotient
%	Divide and return modulus

Table 2-2 Common Arithmetic Operators

And here's an example illustrating these operators in action:

```
<?php
// define variables
$x = 10;
$y = 5;
$z = 3;

// add
$sum = $x + $y;
echo "$x + $y = $sum\n";

// subtract
$diff = $x - $y;
echo "$x - $y = $diff\n";

// multiply
$product = $x * $y;
echo "$x * $y = $product\n";

// divide and get quotient
$quotient = $x / $y;
```

```
echo "$x / $y = $quotient\n";

// divide and get modulus
$modulus = $x % $y;
echo "$x % $y = $modulus\n";
?>
```

Concatenating Strings

To combine strings, use PHP's concatenation operator, which happens to be a period (.). The following example illustrates:

```
<?php
// define variables
$country = 'England';
$city = 'London';

// combine into single string
// output: 'Welcome to London, the coolest city in all of England'
echo 'Welcome to ' . $city . ', the coolest city in all of ' . $country;
?>
```

Comparing Variables

PHP lets you compare one variable or value with another via its wide range of comparison operators, listed in Table 2-3. And here's an example illustrating these operators in action:

```
<?php
// define variables
$p = 10;
$q = 11;
$r = 11.3;
$s = 11;

// test if $q is greater than $p
// returns true
echo ($q > $p);

// test if $q is less than $p
// returns false
echo ($q < $p);

// test if $q is greater than or equal to $s
// returns true
echo ($q >= $s);
```

```
// test if $r is less than or equal to $s
// returns false
echo ($r <= $s);
```

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
===	Equal to and of the same type

Table 2-3 Common Comparison Operators

```
// test if $q is equal to $s
// returns true
echo ($q == $s);

// test if $q is equal to $r
// returns false
echo ($q == $r);
?>
```

It's worth making special mention here of the === operator, which is excluded from the preceding example. This operator allows for stricter comparison between variables: it only returns true if the two variables or values being compared hold the same information and are of the same data type. Thus, in the next example, a comparison between $bool and $num would return true when compared with the == operator, but false when compared with the === operator:

```
<?php
// define variables of two types
// but with the same value
$bool = (boolean) 1;
$int = (integer) 1;

// returns true
echo ($bool == $int);

// returns false
echo ($bool === $int);
```

?>

Performing Logical Tests

When building complex conditional expressions you will often come across situations where it's necessary to combine one or more logical tests. PHP's three most commonly used logical operators, listed in
Table 2-4, are intended specifically for this situation.

Operator	Description
&&	AND
||	OR
!	NOT

Table 2-4 Common Logical Operators

Logical operators really come into their own when combined with conditional tests, so the following example is illustrative only

```
<?php
// define variables
$price = 100;
$size = 18;

// logical AND test
// returns true if both comparisons are true
// returns true here
echo ($price > 50 && $size < 25);

// logical OR test
// returns true if any of the comparisons are true
// returns false here
echo ($price > 150 || $size > 75);

// logical NOT test
// reverses the logical test
// returns false here
echo !($size > 10);
?>
```

Other Useful Operators

There are a few other operators that tend to come in handy during PHP development. First, the addition-assignment operator, represented by the symbol +=, lets you simultaneously add and assign a new value to a variable. The following example illustrates:

```
<?php
// define variable
$count = 7;

// add 2 and re-assign new value to variable
$count += 2;

// output: 9
echo $count;
?>
```

Here, the expression $count +=7 is equivalent to the expression $count = $count + 2—an addition operation followed by an assignment of the result back to the same variable. In a similar vein, there exist operators for other mathematical and string assignments. Table 2-5 has a list.

Operator	Description
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign quotient
%=	Divide and assign modulus
.=	Concatenate and assign (strings only)

Table 2-5 Common Assignment Operators

Here are examples of these in action:

```
<?php
// define variables
$count = 7;
$age = 60;
$greeting = 'We';

// subtract 2 and re-assign new value to variable
// equivalent to $count = $count - 2
// output: 5
$count -= 2;
echo $count;

// divide by 5 and re-assign new value to variable
```

```
// equivalent to $age = $age / 5
// output: 12
$age /= 5;
echo $age;

// add new string and re-assign new value to variable
// equivalent to $greeting = $greeting . 'lcome!'
// output: 'Welcome!'
$greeting .= 'lcome!';
echo $greeting;
?>
```

As you proceed through this book, you'll also comes across the auto-increment and auto-decrement operators, represented by the ++ and -- symbols respectively.

The operators automatically add 1 to, or subtract 1 from, the variable they are applied to. Here's an example:

```
<?php
// define variable
$count = 19;

// increment
$count++;

// output: 20
echo $count;

// now decrement
$count--;

// output: 19
echo $count;
?>
```

These operators are commonly found in loop counters

Understanding Operator Precedence

Back in math class, you probably learned about BODMAS, a mnemonic that specifies the order in which a calculator or a computer performs a sequence of mathematical operations: Brackets, Order, Division, Multiplication, Addition, and Subtraction. Well, PHP follows a similar set of rules when determining which operators have precedence over others—and learning these

rules can save you countless frustrating hours debugging a calculation that looks right, yet somehow always returns the wrong result!

The following list (a short version of a much longer list in the PHP manual) illustrates PHP's most important precedence rules. Operators at the same level have equal precedence.

- ++ --
- !
- * / %
- + - .
- < <= > >=
- == != === !==
- &&
- ||
- = += -= *= /= .= %= &= |= ^=

Assignment Questions

2 Marks

1. What is PHP?
2. Mention the advantages of PHP?
3. Explain Basic PHP syntax? And create a simple PHP Script.
4. What is variable ?write the syntax.
5. How to store data in variables?
6. What is variables?Explain the rules for PHP Variables.
7. What is operator?
8. List the types of operator.
9. What is Data Type? List types of Data Types.
10. What is Constant?Explain with example.
11. What is resource?
12. What is Null value?
13. What is object?
14. What is Array?
15. Explain strings in PHP?

5 Marks

1. What is PHP? Mention the advantages of PHP.
2. Explain comments in PHP.
3. What is variables?Explain the rules for PHP Variables.
4. Explain Manipulating variables with operators.
5. Write a PHP program to generate prime numbers.

10 Marks

6. Explain setting and checking variables in PHP.
7. List and explain Basic Development Concepts?
8. What is operator? List and explain operators in PHP.
9. What is Data Type? Explain types of Data Types.