Unit-1: PHP fundamentals

1.1 Concepts of PHP and introduction

PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side. PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page).

PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995. PHP 8.1.0 is the latest version of PHP, which was released on 07th June 2022. Some important points need to be noticed about PHP are as followed:

- PHP stands for "**Hypertext Preprocessor**"
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.
- PHP is simple and easy to learn.
- PHP Syntax is C-Like.

What can PHP do?

PHP has two main applications:

- Server-side scripting – PHP is well-suited for developing dynamic websites and web applications.
- Command-line scripting – like Python and Perl, you can run PHP script from the command line to perform administrative tasks like sending emails and generating PD F files.

How PHP works?

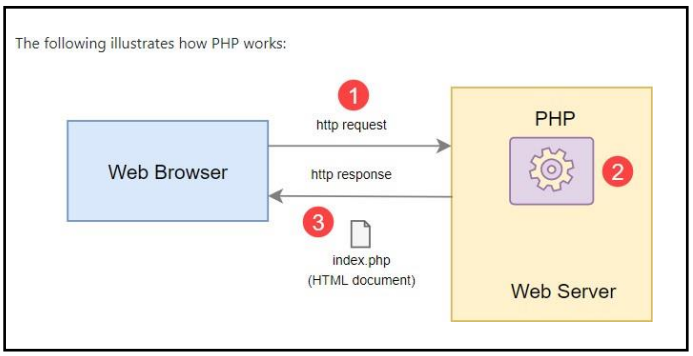


Figure: How PHP works?

First, the web browser sends an HTTP request to the web server, e.g., index.php.
Second, the PHP pre-processor that locates on the web server processes PHP code to generate the HTML document.

Third, the web server sends the HTML document back to the web browser.

❖ **Why use PHP?**

PHP is a server-side scripting language, which is used to design dynamic web applications with MySQL databases.

- It handles dynamic content, database as well as session tracking for the website.
- You can create sessions in PHP.
- It can access cookie variables and also set cookies.
- It helps to encrypt the data and apply validation.
- PHP supports several protocols such as HTTP, POP3, SNMP, LDAP, IMAP, and many more.

- Using the PHP language, you can control the user to access some pages of your website.
- As PHP is easy to install and set up, this is the main reason why PHP is the best language to learn.
- PHP can handle the forms, such as - collect the data from users using forms, save it into the database, and return useful information to the user. For example - Registration form.

❖ Characteristics of PHP

Five important characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

❖ Applications of PHP

As mentioned before, PHP is one of the most widely used languages over the web. I'm going to list few of them here:

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

1.2 PHP syntax: variables, constants, echo and print commands

PHP Syntax:

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax:

A PHP script can be placed anywhere in the document.

A PHP script starts with <?php and ends with ?>:

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

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting 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:

Example:

```
<!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>
```

PHP Variables:
Variables are "containers" for storing information. In PHP, a variable starts with the $ sign, followed by the name of the variable. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

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)

Remember that PHP variable names are case-sensitive!

Example:

```
<!DOCTYPE html> <html><body>
<?php
	$txt = "Hello world!";
	$x = 5;
	$y = 10.5;
	echo $txt;
	echo "<br>";
	echo $x;
	echo "<br>";
	echo $y;
?>
</body></html>
```

Variable Scope
PHP has three types of variable scopes:
1. Global variable
2. Local variable
3. Static variable

1. Global

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example1: Variable with global scope:

```
<?php
   $name = "Tom Cruise";            //Global Variable
   function global_var()
   {
      global $name;
      echo "Variable inside the function: ". $name;
      echo "</br>";
   }
   global_var();
   echo "Variable outside the function: ". $name;
?>
```

Example2: Another way to use the global variable inside the function is predefined $GLOBALS array.

```
<?php
   $num1 = 5;        //global variable
   $num2 = 13;       //global variable
   function global_var()
   {
         $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
         echo "Sum of global variables is: " .$sum;
   }
   global_var();
?>
```

2. Local

- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example: Variable with local scope:

```
<?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>";
?>
```

3. Static

- It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.
- Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

Example:

```
<?php
    function static_var()
    {
      static $num1 = 3;            //static variable
      $num2 = 6;                   //Non-static variable
      //increment in non-static variable
```

```
        $num1++;
        //increment in static variable
        $num2++;
        echo "Static: " .$num1 ."</br>";
        echo "Non-static: " .$num2 ."</br>";
    }
    //first function call
    static_var();

    //second function call
    static_var();
?>
```

Output:
Static: 4
Non-static: 7
Static: 5
Non-static: 7

PHP $ and $$ Variables

The **$var** (single dollar) is a normal variable with the name **var** that stores any value like string, integer, float, etc.

The **$$var** (double dollar) is a reference variable that stores the value of the $variable inside it.

Example: To understand the difference better, let's see some examples.

```
<?php
        $hello = "I am Awesome";
        $b = "hello";
        echo $b . "<br>";
        echo $$b;
?>
```

Output:
hello
I am Awesome

Note:

In the above example, The $hello is a normal variable used to store a value. It can store any value like integer, float, char, string etc. On the other hand, the $$b is known as reference variable where $hello is a normal variable. The $$b used to refer to the variable with the name as value of the variable $hello.

PHP Constants

Constants are like variables except that once they are defined they cannot be changed or undefined.
A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
A valid constant name starts with a letter or underscore (no $ sign before the constant name).
Note: Unlike variables, constants are automatically global across the entire script.
PHP constants can be defined by 2 ways:

1. Using define() function
2. Using const keyword

PHP constant: define()

To create a constant, use the define() function.

Syntax :

```
define(name, value, case-insensitive)
```

Parameters:

name: Specifies the name of the constant
value: Specifies the value of the constant
case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

Example1: Create a constant with a case-sensitive name:

```
<!DOCTYPE html><html><body>
<?php
        // case-sensitive constant name
        define("GREETING", "Welcome to udhna college!");echo
        GREETING;
?> </body></html>
```

OUTPUT: Welcome to udhna college!

Constants are Global

Constants are automatically global and can be used across the entire script.
This example uses a constant inside a function, even if it is defined outside the function:

```
<!DOCTYPE html><html><body>
<?php
        define("GREETING", "Welcome to udhna college!");
        function myTest()
        {
              echo GREETING;
        }
 myTest();
?> </body></html>
```

OUTPUT: Welcome to udhna college!

PHP constant: const keyword

PHP introduced a keyword const to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function. The constants defined using **const** keywords are case-sensitive.

Example:

```
<?php
        const MESSAGE="Hello i am const in PHP";
        echo MESSAGE;
?>
```

OUTPUT:

Hello i am const in PHP

Constant() function

There is another way to print the value of constants using constant() function instead of using the echo statement.

Syntax:

```
constant (name)
```

Example:

```
<?php
         define("MSG", "UDHNA");
        echo MSG, "</br>";
        echo constant("MSG");
        //both are similar
?>
```

OUTPUT:
UDHNA
UDHNA

❖ **echo and print Statements**

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

1) PHP Echo

PHP echo is a language construct, not a function. Therefore, you don't need to use parentheses with it. But if you want to use more than one parameter, it is required to use parenthesis.

Syntax:

```
void echo ( string $arg1 [, string $... ] )
```

PHP echo statements can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:

- echo is a statement, which is used to display the output.
- echo can be used with or without parentheses: echo(), and echo.
- echo does not return any value.
- We can pass multiple strings separated by a comma (,) in echo.
- echo is faster than the print statement.

PHP echo: printing Multiline string

```
<?php
        echo "Hello by PHP echo
        this is multi line
        text printed by
        PHP echo statement
        ";
?>
```

OUTPUT:

Hello by PHP echo this is multi line text printed by PHP echo statement

Example:

```
<?php
        echo "Hello escape \"sequence\" characters";
?>
```

OUTPUT:

Hello escape "sequence" characters

2) The PHP print Statement

The print statement can be used with or without parentheses: print or print().

syntax:

```
int print(string $arg)
```

PHP print statements can be used to print the string, multi-line strings, escaping characters, variables, array, etc. Some important points that you must know about the echo statement are:

- print is a statement, used as an alternative to echo at many times to display the output.
- print can be used with or without parentheses.
- print always returns an integer value, which is 1.
- Using print, we cannot pass multiple arguments.
- print is slower than the echo statement.

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        print "<h2>PHP is Fun!</h2>";
        print "Hello world!<br>";
        print "I'm about to learn PHP!";
?>
</body>
</html>
```

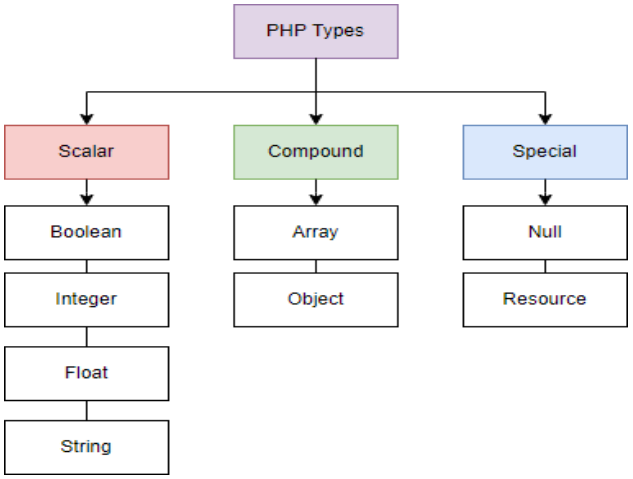
OUTPUT:

PHP is Fun!
Hello world!
I'm about to learn PHP!

1.3 Data types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types (predefined)
2. Compound Types (user-defined)
3. Special Types



PHP Data Types: Scalar Types

It holds only a single value. There are 4 scalar data types in PHP.

1. boolean
2. integer
3. float
4. string

PHP Data Types: Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

1. array
2. object

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. resource
2. NULL

- **PHP Boolean**

Booleans are the simplest data type that works like a switch. It holds only two values: TRUE (1) or FALSE (0). It is often used with conditional statements. If the condition is correct, it returns TRUE otherwise FALSE.

Example:

```
$x = true;
$y = false;
```

- **PHP Integer**

Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional parts or decimal points.

Rules for integer:

- An integer can be either positive or negative.
- An integer must not contain a decimal point.
- Integers can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
- The range of an integer must lie between -2,147,483,648 and 2,147,483,647 i.e., -2^31 to 2^31.

Example:

```
<?php
   $dec1 = 34;
   $oct1 = 0243;
   $hexa1 = 0x45;
   echo "Decimal number: " .$dec1. "</br>";
   echo "Octal number: " .$oct1. "</br>";
   echo "HexaDecimal number: " .$hexa1. "</br>";
?>
```

OUTPUT:
Decimal number: 34
Octal number: 163
HexaDecimal number: 69

- **PHP Float**

A floating-point number is a number with a decimal point. Unlike integers, it can hold numbers with a fractional or decimal point, including a negative or positive sign.

Example:

```
<?php
   $n1 = 19.34;
   $n2 = 54.472;
   $sum = $n1 + $n2;
   echo "Addition of floating numbers: " .$sum;
?>
```

OUTPUT:
Addition of floating numbers: 73.812

- **PHP String**

A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.

String values must be enclosed either within single quotes or in double quotes. But both are treated differently. To clarify this, see the example below:

Example:

```
<?php
   $company = "ABC Limited";

   //both single and double quote statements will treat different
   echo "Hello $company";
   echo "</br>";
   echo 'Hello $company';
?>
```

OUTPUT:
Hello ABC Limited
Hello $company

PHP Array

An array is a compound data type. It can store multiple values of the same data type in a single variable. **Example:**

```
<?php
   $bikes = array ("Royal Enfield", "Yamaha", "KTM");
   var_dump($bikes);       //the var_dump() function returns the datatype and values
   echo "</br>";
   echo "Array Element1: $bikes[0] </br>";
   echo "Array Element2: $bikes[1] </br>";
   echo "Array Element3: $bikes[2] </br>";
?>
```

OUTPUT:

```
array(3) { [0]=> string(13) "Royal Enfield" [1]=> string(6) "Yamaha" [2]=> string(3) "KTM" }
Array Element1: Royal Enfield
Array Element2: Yamaha
Array Element3: KTM
```

PHP object

Objects are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.

Example:

```
<?php
    class bike
   {
        function model()
        {
           $model_name = "Royal Enfield";
           echo "Bike Model: " .$model_name;
        }
    }
    $obj = new bike();
    $obj -> model();
?>
```

Output:

Bike Model: Royal Enfield

PHP Resource

- Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources. For example - a database call. It is an external resource.
- This is an advanced topic of PHP, so we will discuss it later in detail with examples.

PHP 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.

The special type of data type NULL defines a variable with no value.

Example:

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

Output:

NULL

1.4 Operators, Conditional Statements (if. Else, Switch. Case), Arrays

PHP Operators:

PHP Operator is a symbol used to perform operations on operands. In simple words, operators are used to perform operations on variables or values. For example:

```
$num=10+20;
```

In this expression + is the operator and 10,20 are operands

PHP Operators can be categorized in following forms:

1. Arithmetic Operators
2. Assignment Operators
3. Bitwise Operators
4. Comparison Operators
5. Incrementing/Decrementing Operators
6. Logical Operators
7. String Operators
8. Array Operators
9. Conditional Assignment Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

1. Unary Operators: works on single operands such as ++, -- etc.
2. Binary Operators: works on two operands such as binary +, -, *, / etc.
3. Ternary Operators: works on three operands such as "?:".

1. **Arithmetic Operators:** The PHP arithmetic operators are used to perform common arithmetic operations such as addition, subtraction, etc. with numeric values.

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

2. **Assignment Operators:** The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of the right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

3. **Bitwise Operators:** The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

4. **Comparison Operators:** Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

5. **Incrementing/Decrementing Operators:** The increment and decrement operators are used to increase and decrease the value of a variable.

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

6. **Logical Operators:** The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

7. **String Operators:** The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

8. **Array Operators:** The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

9. **PHP Conditional Assignment Operators:** The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = expr1 ? expr2 : expr3	Returns the value of $x. The value of $x is expr2 if expr1 = TRUE. The value of $x is expr3 if expr1 = FALSE
??	Null coalescing	$x = expr1 ?? expr2	Returns the value of $x. The value of $x is expr1 if expr1 exists, and is not NULL. If expr1 does not exist, or is NULL, the value of $x is expr2. Introduced in PHP 7

❖ **PHP Comments**

PHP comments can be used to describe any line of code so that other developers can understand the code easily. It can also be used to hide any code.

PHP supports single line and multi line comments. These comments are similar to C/C++ and Perl style (Unix shell style) comments.

➢ **PHP Single Line Comments**

There are two ways to use single line comments in PHP.

1. // (C++ style single line comment)
2. # (Unix Shell style single line comment)

Example:

```
<?php
        // this is C++ style single line comment
        # this is Unix Shell style single line comment
        echo "Welcome to PHP single line comments";
?>
```

OUTPUT:

```
Welcome to PHP single line comments
```

➢ **PHP Multi Line Comments**

In PHP, we can comment on multiple lines also. To do so, we need to enclose all lines within /* */.

Let's see a simple example of PHP multiple line comment.

Example:

```
<?php
/*
        Anything placed
        within comment
```

```
        will not be displayed
        on the browser;
*/
echo "Welcome to PHP multi line comment";
?>
```

OUTPUT:

```
Welcome to PHP multi line comment
```

❖ Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

1. if statement - executes some code if one condition is true
2. if...else statement - executes some code if a condition is true and another code if that condition is false
3. if...elseif...else statement - executes different codes for more than two conditions
4. switch statement - selects one of many blocks of code to be executed

1. **The if Statement:** The if statement executes some code if one condition is true.

Syntax:

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

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
        $num=12;
        if($num<100)
        {
                echo "$num is less than 100";
        }
?>
</body>
</html>
```

2. **The if...else Statement:** The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax:

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

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
   $num=12;
   if($num%2==0)
   {
        echo "$num is even number";
   }
   else
   {
        echo "$num is odd number";
    }
?>
</body>
</html>
```

3. **The if...elseif...else Statement:** The if...elseif...else statement executes different codes for more than two conditions.

Syntax:

```
if (condition)
{
  code to be executed if this condition is true;
}
elseif (condition)
{
  code to be executed if first condition is false and this condition is true;
}
else
{
  code to be executed if all conditions are false;
}
```

Example:

```
<!DOCTYPE html><html><body>
<?php
// speed in kmph
$speed = 110;
if($speed < 60)
{
   echo "Safe driving speed";
}
elseif($speed > 60 && $speed < 100)
{
   echo "You are burning extra fuel";
}
else
{
   // when speed is greater than 100
   echo "Its dangerous";
}?>
</body></html>
```

4. **switch Statement:** Use the switch statement to select 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;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
$favcolor = "red";
switch ($favcolor)
{
  case "red":
    echo "Your favourite color is red!";
    break;
  case "blue":
    echo "Your favourite color is blue!";
    break;
  case "green":
    echo "Your favourite color is green!";
    break;
  default:
    echo "Your favourite color is neither red, blue, nor green!";
}
?>
</body>
</html>
```

Important points to be noticed about switch case:

1. The default is an optional statement. Even if it is not important, that default must always be the last statement.
2. There can be only one default in a switch statement. More than one default may lead to a Fatal error.

3. Each case can have a break statement, which is used to terminate the sequence of statements.
4. The break statement is optional to use in switch. If break is not used, all the statements will execute after finding a matched case value.
5. PHP allows you to use number, character, string, as well as functions in switch expressions.
6. Nesting of switch statements is allowed, but it makes the program more complex and less readable.
7. You can use semicolon (;) instead of colon (:). It will not generate any errors.

❖ **Arrays**

What is an Array?

An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

1. Indexed arrays - Arrays with a numeric index
2. Associative arrays - Arrays with named keys
3. Multidimensional arrays - Arrays containing one or more arrays

1. PHP Indexed Arrays

There are two ways to create indexed arrays:
The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or
the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

Example:

```
<!DOCTYPE html>
<html><body>
<?php
        $cars = array("Volvo", "BMW", "Toyota");
        echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
</body></html>
```

OUTPUT:

I like Volvo, BMW and Toyota.

2. PHP Associative Arrays

Associative arrays are arrays that use **named keys** that you assign to them.
There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
</body>
</html>
```

OUTPUT:

Peter is 35 years old.

3. Multidimensional Arrays

- A multidimensional array is an array containing one or more arrays.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.
To get access to the elements of the $cars array we must point to the two indices (row and column):

Example1:

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array (
```

```
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
</body>
</html>
```

OUTPUT:

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

Example:2

We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

```
<!DOCTYPE html><html><body>
<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}?>
</body></html>
```

1.5 Sorting Arrays, Php Loops

❖ **Sorting Arrays**

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.
In this chapter, we will go through the following PHP array sort functions:

1. sort() - sort arrays in ascending order
2. rsort() - sort arrays in descending order
3. asort() - sort associative arrays in ascending order, according to the value
4. ksort() - sort associative arrays in ascending order, according to the key
5. arsort() - sort associative arrays in descending order, according to the value
6. krsort() - sort associative arrays in descending order, according to the key

1. Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

```
<!DOCTYPE html>
<html><body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
$clength = count($cars);
for($x = 0; $x < $clength; $x++)
{
  echo $cars[$x];
  echo "<br>";
}
?>
</body></html>
```

OUTPUT:
BMW
Toyota
Volvo

The following example sorts the elements of the $numbers array in ascending numerical order:

```
<!DOCTYPE html><html><body>
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
  echo $numbers[$x];
  echo "<br>";
}
?>
</body></html>
```

OUTPUT:
2
4
6
11
22

2. Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

```
<!DOCTYPE html><html><body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
     echo $cars[$x];
     echo "<br>";
}?>     </body></html>
```

OUTPUT:
Volvo
Toyota
BMW

3. Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

```
<!DOCTYPE html><html><body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
foreach($age as $x => $x_value)
{
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?></body></html>
```

OUTPUT:
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

4. Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

```
<!DOCTYPE html><html><body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}?>
</body></html>
```

OUTPUT:
Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35

5. Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

```
<!DOCTYPE html><html><body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
</body></html>
```

OUTPUT:
Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35

6. Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

```
<!DOCTYPE html><html><body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
</body></html>
```

OUTPUT:
Key=Peter, Value=35
Key=Joe, Value=43
Key=Ben, Value=37

❖ PHP loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops. Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

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

1. The PHP while Loop

The while loop will execute block of code repeatedly until the condition is **FALSE**. Once the condition gets FALSE, it exits from the body of loop. It can be used if the number of iterations is not known.

The while loop is also called an Entry control loop, because the condition is checked before entering the loop body. This means that first the condition is checked. If the condition is true, the block of code will be executed.

Syntax:

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

Example :

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

2. The PHP do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true. The do-while loop is also named as exit control loop.
- The PHP do-while loop is used to execute a set of code of the program several times. If you have to execute the loop at least once and the number of iterations is not even fixed, it is recommended to use the do-while loop.
- It executes the code at least one time always because the condition is checked after executing the code.
- The do-while loop is very much similar to the while loop except the condition check. The main difference between both loops is that while loop checks the condition at the beginning, whereas do-while loop checks the condition at the end of the loop.

Syntax:

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

Example:

```
<!DOCTYPE html><html><body>
<?php
$x = 1;
do
{
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?></body></html>
```

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

Syntax:

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

Parameters:

1. init counter: Initialize the loop counter value
2. test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
3. increment counter: Increases the loop counter value

Example:

```
<!DOCTYPE html><html><body>
<?php
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>
</body></html>
```

4. The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax:

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

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Example:

```
<!DOCTYPE html><html><body>
<?php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $value)
{
        echo "$value <br>";
}
?>
</body></html>
```

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

```
<!DOCTYPE html><html><body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $val) {
       echo "$x = $val<br>";
}
?>
</body></html>
```

PHP Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement. The break statement can also be used to jump out of a loop.
This example jumps out of the loop when x is equal to 4:

```
<!DOCTYPE html><html><body>
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
```

```
   break;
  }
  echo "The number is: $x <br>";
}
?>
</body></html>
```

PHP Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

```
<!DOCTYPE html><html><body>
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4)
 {
    continue;
  }
  echo "The number is: $x <br>";
}
?>
</body></html>
```

Practice Programs

1. Arithmetic Operators

```
<html>
   <head>
     <title>Arithmetical Operators</title>
   </head>
   <body>
      <?php
        $a = 42;
        $b = 20;
        $c = $a + $b;
        echo "Addition Operation Result: $c <br/>";
        $c = $a - $b;
        echo "Subtraction Operation Result: $c <br/>";
        $c = $a * $b;
        echo "Multiplication Operation Result: $c <br/>";
        $c = $a / $b;
        echo "Division Operation Result: $c <br/>";
        $c = $a % $b;
        echo "Modulus Operation Result: $c <br/>";
        $c = $a++;
        echo "Increment Operation Result: $c <br/>";
        $c = $a--;
        echo "Decrement Operation Result: $c <br/>";
      ?>
      </body>
</html>
```

OUTPUT:

```
Addition Operation Result: 62
Subtraction Operation Result: 22
Multiplication Operation Result: 840
Division Operation Result: 2.1
Modulus Operation Result: 2
Increment Operation Result: 42
Decrement Operation Result: 43
```

2. Assignment Operators

```
<html>
    <head><title>Assignment Operators</title></head>
    <body>
    <?php
      $a = 42;
      $b = 20;

      $c = $a + $b;
      echo "Addition Operation Result: $c <br/>";
      $c += $a;
      echo "Add AND Assignment Operation Result: $c <br/>";
      $c -= $a;
      echo "Subtract AND Assignment Operation Result: $c <br/>";
      $c *= $a;
      echo "Multiply AND Assignment Operation Result: $c <br/>";
      $c /= $a;
      echo "Division AND Assignment Operation Result: $c <br/>";
      $c %= $a;
      echo "Modulus AND Assignment Operation Result: $c <br/>";
    ?>
  </body>
</html>
```

OUTPUT:

Addition Operation Result: 62
Add AND Assignment Operation Result: 104
Subtract AND Assignment Operation Result: 62
Multiply AND Assignment Operation Result: 2604
Division AND Assignment Operation Result: 62
Modulus AND Assignment Operation Result: 20

3. Incrementing/Decrementing Operators

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be : " . $a++ . "<br />\n";
echo "Should be : " . $a . "<br />\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be : " . ++$a . "<br />\n";
echo "Should be : " . $a . "<br />\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be : " . $a-- . "<br />\n";
echo "Should be : " . $a . "<br />\n";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "Should be : " . --$a . "<br />\n";
echo "Should be : " . $a . "<br />\n";
?>
```

OUTPUT:

Postincrement
Should be : 5
Should be : 6
Preincrement
Should be : 6
Should be : 6
Postdecrement
Should be : 5
Should be : 4
Predecrement
Should be : 4
Should be : 4

4. Array Operators

```
<?php
```

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => "strawberry", "c" => "cherry");

$c = $a + $b;    // Union of $a and $b
echo "Union of \$a and \$b: ";
var_dump($c);

$c = $b + $a;     // Union of $b and $a
echo "<br> Union of \$b and \$a: ";
var_dump($c);

$a += $b;     // Union of $a += $b is $a and $b
echo "<br> Union of \$a += \$b: ";
var_dump($a);
?>
```

OUTPUT:

Union of $a and $b: array(3) { ["a"]=> string(5) "apple" ["b"]=> string(6) "banana" ["c"]=> string(6) "cherry" }
Union of $b and $a: array(3) { ["a"]=> string(4) "pear" ["b"]=> string(10) "strawberry" ["c"]=> string(6) "cherry" }
Union of $a += $b: array(3) { ["a"]=> string(5) "apple" ["b"]=> string(6) "banana" ["c"]=> string(6) "cherry" }

5. PHP program to print sum of digits

```
<?php

$num = 14597;
$sum=0;
$rem=0;

 for($i =0; $i<=strlen($num);$i++)
 {
  $rem=$num%10;
  $sum = $sum + $rem;
  $num=$num/10;
 }
 echo "Sum of digits 14597 is $sum";
?>
```

6. PHP program to check prime number

```
<!DOCTYPE html>
<html><body>
 <?php
        $num = 13;
        $count=0;
        for ( $i=1; $i<=$num; $i++)
        {
                if (($num%$i)==0)
                {
                $count++;
                }
        }
        if ($count<3)
        {
                echo "$num is a prime number.";
```

```
        }
        else
        {
                echo "$num is not a prime number.";
        }
?> </body></html>
```

7. PHP program to print table of a number:

```
<!DOCTYPE html>
<html><body>
<?php
$num = 9;
for($i=1; $i<=10; $i++)
{
        $product = $i*$num;
        echo "$num * $i = $product" ;
        echo '<br>';
}
?>
</body></html>
```

8. PHP program to print factorial of a number:

```
<!DOCTYPE html>
<html>
<body>
<?php
          $n = 5;
          $f = 1;
          for ($i=$n; $i>=1; $i--)
          {
              $f = $f * $i;
          }
          echo "$n! = $f";
?>
</body></html>
```

9. PHP Program To Check For An Armstrong Number

```
Armstrong Number:
abc = (a*a*a) + (b*b*b) + (c*c*c)
Example: 0, 1, 153, 371, 407, 471, etc.
<!DOCTYPE html>
<html>
<body>
<?php
          $n=371;
          $sum=0;
          $i=$n;
          while($i!=0)
          {
              $x=$i%10;
              $sum=$sum+$x*$x*$x;
              $i=$i/10;
          }
          if($n==$sum)
          {
              echo "$n is an Armstrong Number.";
```

```
        }
        else
        {
            echo "$n is not an Armstrong Number.";
        }
?>  </body></html>
```

10. PHP program to check for a Palindrome number.

```
<!DOCTYPE html>
<html><body>
<?php
        $num = 12321;
        $x = 0;
        $n =$num;
        while(floor($num))
        {
                $mod = $num%10;
                $x = $x * 10 + $mod;
                $num = $num/10;
        }
        if($n==$x)
        {
                echo "$n is a Palindrome number.";
        }
        else
        {
                echo "$n is not a Palindrome number.";
        }
?>  </body></html>
```

11. PHP program to print fibonacci series.

```
<!DOCTYPE html>
<html><body>
<?php
        $num = 0;
        $n1 = 0;
        $n2 = 1;
        echo "<h3>Fibonacci series for first 12 numbers: </h3>";
        echo "\n";
        echo $n1.' '.$n2.' ';
        while ($num < 10 )
        {
                $n3 = $n2 + $n1;
                echo $n3.' ';
                $n1 = $n2;
                $n2 = $n3;
                $num = $num + 1;
        }
?>  </body></html>
```

OUTPUT:

Fibonacci series for first 12 numbers:
0 1 1 2 3 5 8 13 21 34 55 89