UNIT 5:

PHP (Hypertext Preprocessor)

PHP: Hypertext Preprocessor (earlier called, Personal Home Page)

PHP is an HTML-embedded, server-side scripting language designed for web development. It is also used as a general purpose programming language. It was created by Rasmus Lerdorf in 1994 and appeared in the market in 1995. Much of its syntax is borrowed from C, C++, and Java.

PHP codes are simply mixed with HTML codes and can be used in combination with various web frameworks. Its scripts are executed on the server. PHP code is processed by a PHP interpreter. The main goal of PHP is to allow web developer to create dynamically generated pages quickly.

A PHP file consists of texts, HTML tags and scripts with a file extension of .php, .php3, or .phtml. You can create a login page, design a form, create forums, dynamic and static websites and many more with PHP.

Key features of PHP

- PHP stands for Hypertext Preprocessor.
- PHP is a server-side scripting language like ASP.
- PHP supports various databases like MySQL, Oracle, Sybase, Solid, PostgreSQL, Informix etc.

- PHP is an open source software and it is free to download and use.

Advantages of PHP

1) **Free of Cost**: PHP is open source and all its components are free to use and distribute.

2) **Platform independent**: PHP is platform independent and can be run on all major operating systems.

3) **Compatible with almost all servers**: PHP is compatible with almost all servers used today.

4) **Secure**: PHP has multiple layers of security to prevent threats and other malicious attacks.

5) **Easy to learn**: PHP has a very easy and understandable syntax. Its codes are based on C, C++ and embedded with HTML so it is very easy to learn for a programmer.

PHP | Basic Syntax

The structure which defines PHP computer language is called **PHP syntax**.
The PHP script is executed on the server and the HTML result is sent to the browser. It can normally have HTML and PHP tags. PHP or Hypertext Preprocessor is a widely used open-source general-purpose scripting language and can be embedded with HTML. PHP files are saved with the ".php" extension. PHP scripts can be written anywhere in the document within PHP tags along with normal HTML.

Escaping To PHP:
Writing the PHP code inside *<?php ….?>* is called **Escaping to PHP**.
The mechanism of separating a normal HTML from PHP code is called the mechanism of Escaping To PHP. There are various ways in which this can be done. Few methods are already set by default but in order to use few others like Short-open or ASP-style tags, we need to change the configuration of the *php.ini* file. These tags are also used for embedding PHP within HTML. There are 4 such tags available for this purpose.
Canonical PHP Tags: The script starts with **<?php** and ends with **?>**. These tags are also called 'Canonical PHP tags'. Everything outside of a pair of opening and closing tags is ignored by the PHP parser. The open and closing tags are called delimiters. Every PHP command ends with a semi-colon (**;**). Let's look at the *hello world* program in PHP.

- PHP

```
<?php

# Here echo command is used to print

echo "Hello, world!";

?>
```

Output:
Hello, world!

SGML or Short HTML Tags: These are the shortest option to initialize a PHP code. The script starts with **<?** and ends with **?>**. This will only work by setting the *short_open_tag* setting in the *php.ini* file to 'on'.
Example:

- PHP

```
<?

# Here echo command will only work if

# setting is done as said before

echo "Hello, world!";

?>
```

Output:
Hello, world!

HTML Script Tags: These are implemented using script tags. This syntax is removed in PHP 7.0.0. So its no more used.
Example:

- PHP

```
<script language="php">

echo "hello world!";

</script>
```

Output:
hello world!

ASP Style Tags: To use this we need to set the configuration of the *php.ini* file. These are used by Active Server Pages to describe code blocks. These tags start with **<%** and end with **%>**.
Example:

- PHP

```
<%

# Can only be written if setting is turned on

# to allow %

echo "hello world";

%>
```

Output:
hello world
Constants:
Constants can be defined using the *const* keyword or define() function.
There is some difference between constants and variables.

- Constants do not have $ in front of them like variables have.
- Constants can be accessed from anywhere without regard to variable scoping rules.

Comments in PHP:

Comments help in reminding the developer about the code if it’s re-visited after a period of time.

A comment is something that is ignored and not read or executed by the PHP engine or the language as part of a program and is written to make the code more readable and understandable. These are used to help other users and developers to describe the code and what it is trying to do. It can also be used in documenting a set of codes or parts of a program. You must have noticed this in the above sample programs. PHP supports two types of comment:

- **Single Line Comment**: As the name suggests these are single line or short relevant explanations that one can add to their code. To add this, we need to begin the line with (**//**) or (**#**).

Example:

- PHP

```
<?php

// This is a single line comment

// These cannot be extended to more lines



echo "hello world!!!";



# This is also a single line comment

?>
```

Output:

hello world!!!

- **Multi-line or Multiple line Comment**: These are used to accommodate multiple lines with a single tag and can be extended to many lines as required by the user. To add this, we need to begin and end the line with (**/*…*/**)

- PHP

```
<?php

/* This is a multi line comment

   In PHP variables are written

   by adding a $ sign at the beginning.*/



$geek = "hello world!";

echo $geek;

?>
```

Output:
hello world!

Case Sensitivity in PHP:

- **PHP is insensitive to whitespace.** This includes all types of spaces that are invisible on the screen including tabs, spaces, and carriage returns. Even one space is equal to any number of spaces or carriage returns. This means that PHP will ignore all the spaces or tabs in a single row or carriage return in multiple rows. Unless a semi-colon is encountered, PHP treats multiple lines as a single command.

Example:

- PHP

```
<?php

// PHP code illustrate the whitespace insensitivity

$var1        =    15;

$var2 =

30;

$sum = $var1

+

$var2;



// "\n" for new line

echo $sum, "\n";



$sum1 = $var1 + $var2;

echo $sum1;

?>
```

Output:

45

45

Both of them show the same results without any errors.

- **PHP is case-sensitive**. All the keywords, functions, and class names in PHP (while, if, echo, else, etc) are NOT case-sensitive except variables. Only variables with

different cases are treated differently. Let’s look at this example:

- PHP

```
<?php

// Here we can see that all echo

// statements are executed in the same manner



$variable = 25;

echo $variable;

ECHO $variable;

EcHo $variable;



// but this line will show RUNTIME ERROR as

// "Undefined Variable"

echo $VARIABLE

?>
```

Output:

25

25

25

Blocks in PHP:

In PHP, multiple statements can be executed simultaneously (under a single condition or loop) by using curly-braces (**{}**). This forms a block of statements that gets executed simultaneously.

- PHP

```
<?php
$var = 50;
if ($var>0){
   echo ("Positive as \n");
   echo ("greater than 0");
}
?>
```

Output:

```
Positive as
greater than 0
```

PHP Variables

<=”” p=”” style="color: rgb(51, 51, 51); font-family: inter-regular, system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Helvetica, Arial, sans-serif; font-size: 16px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; text-align: justify; text-indent: 0px; text-transform: none; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; white-space: normal; background-color: rgb(255, 255, 255); text-decoration-thickness: initial; text-decoration-style: initial; text-decoration-color: initial;">

In PHP, a variable is declared using a **$ sign** followed by the variable name. Here, some important points to know about variables:

- As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
- After declaring a variable, it can be reused throughout the code.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

```
1. $variablename=value;
```

Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

PHP Variable: Declaring string, integer, and float

Let's see the example to store string, integer, and float values in PHP variables.

File: variable1.php

```
1. <?php
2. $str="hello string";
3. $x=200;
4. $y=44.6;
5. echo "string is: $str <br/>";
6. echo "integer is: $x <br/>";
7. echo "float is: $y <br/>";
8. ?>
```

Output:

```
string is: hello string
integer is: 200
float is: 44.6
```

PHP Variable: Sum of two variables

File: variable2.php

```php
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

Output:

```
11
```

PHP Variable: case sensitive

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

File: variable3.php

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

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

File: variablevalid.php

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)

echo "$a <br/> $_b";
?>
```

Output:

```
hello
hello
```

File: variableinvalid.php

```
<?php
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$4c <br/> $*d";
?>
```

Output:

```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable
(T_VARIABLE)
 or '$' in C:\wamp\www\variableinvalid.php on line 2
```

PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

PHP String

PHP string is a sequence of characters i.e., used to store and manipulate text. PHP supports only 256-character set and so that it does not offer native Unicode support. There are 4 ways to specify a string literal in PHP.

1. single quoted

2. double quoted
3. heredoc syntax
4. newdoc syntax (since PHP 5.3)

Single Quoted

We can create a string in PHP by enclosing the text in a single-quote. It is the easiest way to specify string in PHP.

For specifying a literal single quote, escape it with a backslash (\) and to specify a literal backslash (\) use double backslash (\\). All the other instances with backslash such as \r or \n, will be output same as they specified instead of having any special meaning.

For Example

Following some examples are given to understand the single quoted PHP String in a better way:

Example 1

```
<?php
    $str='Hello text within single quote';
    echo $str;
?>
```

Output:

```
Hello text within single quote
```

We can store multiple line text, special characters, and escape sequences in a single-quoted PHP string.

Example 2

```
<?php
$str1='Hello text
multiple line
text within single quoted string';
```

```
$str2='Using double "quote" directly inside single quoted string';
$str3='Using escape sequences \n in single quoted string';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
Hello text multiple line text within single quoted string
Using double "quote" directly inside single quoted string
Using escape sequences \n in single quoted string
```

Example 3

```
<?php
$num1=10;
$str1='trying variable $num1';
$str2='trying backslash n and backslash t inside single quoted string \n \t';
$str3='Using single quote \'my quote\' and \\backslash';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

Output:

```
trying variable $num1
trying backslash n and backslash t inside single quoted string \n \t
Using single quote 'my quote' and \backslash
```

Note: ***In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \' and backslash through \\ inside single quoted PHP strings.***

Double Quoted

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

Example 1

```
<?php
$str="Hello text within double quote";
echo $str;
```

```
4. ?>
```

Output:

```
Hello text within double quote
```

Now, you **can't use double quote directly** inside double quoted string.

Example 2

```
1. <?php
2. $str1="Using double "quote" directly inside double quoted string";
3. echo $str1;
4. ?>
```

Output:

```
Parse error: syntax error, unexpected 'quote' (T_STRING) in
C:\wamp\www\string1.php on line 2
```

We **can store multiple line text, special characters and escape sequences** in a double quoted PHP string.

Example 3

```
1. <?php
2. $str1="Hello text
3. multiple line
4. text within double quoted string";
5. $str2="Using double \"quote\" with backslash inside double quoted string";
6. $str3="Using escape sequences \n in double quoted string";
7. echo "$str1 <br/> $str2 <br/> $str3";
8. ?>
```

Output:

```
Hello text multiple line text within double quoted string
Using double "quote" with backslash inside double quoted string
Using escape sequences in double quoted string
```

In double quoted strings, **variable will be interpreted**.

Example 4

```
1. <?php
2. $num1=10;
3. echo "Number is: $num1";
4. ?>
```

Output:

```
Number is: 10
```

Heredoc

Heredoc syntax (<<<) is the third way to delimit strings. In Heredoc syntax, an identifier is provided after this heredoc <<< operator, and immediately a new line is started to write any text. To close the quotation, the string follows itself and then again that same identifier is provided. That closing identifier must begin from the new line without any whitespace or tab.

Naming Rules

The identifier should follow the naming rule that it must contain only alphanumeric characters and underscores, and must start with an underscore or a non-digit character.

For Example

Valid Example

```
1. <?php
2.    $str = <<<Demo
3. It is a valid example
4. Demo;    //Valid code as whitespace or tab is not valid before closing identifier
5. echo $str;
6. ?>
```

Output:

```
It is a valid example
```

Invalid Example

We cannot use any whitespace or tab before and after the identifier and semicolon, which means identifier must not be indented. The identifier must begin from the new line.

```
<?php
    $str = <<<Demo
It is Invalid example
        Demo;    //Invalid code as whitespace or tab is not valid before closing identifier
echo $str;
?>
```

This code will generate an error.

Output:

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\xampp\PMA\heredoc.php on line 7
```

Heredoc is similar to the double-quoted string, without the double quote, means that quote in a heredoc are not required. It can also print the variable's value.

Example

```
<?php
    $city = 'Delhi';
    $str = <<<DEMO
Hello! My name is Misthi, and I live in $city.
DEMO;
    echo $str;
 ?>
```

Output:

```
Hello! My name is Misthi, and I live in Delhi.
```

Example

We can add multiple lines of text here between heredoc syntax.

```php
<?php
    $str = <<<DEMO
It is the example
of multiple
lines of text.
DEMO;
    echo $str;

echo '</br>';

echo <<<DEMO    // Here we are not storing string content in variable str.
It is the example
of multiple
lines of text.
DEMO;
 ?>
```

Output:

```
It is the example of multiple lines of text.
It is the example of multiple lines of text.
```

Below are the example with class and their variable

Example

```php
<?php
class heredocExample{
        var $demo;
        var $example;
        function __construct()
        {
                $this->demo = 'DEMO';
                $this->example = array('Example1', 'Example2', 'Example3');
        }
    }
    $heredocExample = new heredocExample();
```

```
12.  $name =  'Gunjan';
13.
14.  echo <<<ECO
15.  My name is "$name". I am printing some $heredocExample->demo example.
16.  Now, I am printing {$heredocExample->example[1]}.
17.  It will print a capital 'A': \x41
18. ECO;
19.  ?>
```

Output:

```
My name is "Gunjan". I am printing some DEMO example.
Now, I am printing Example2.
It will print a capital 'A': A
```

Newdoc

Newdoc is similar to the heredoc, but in newdoc parsing is not done. It is also identified with three less than symbols <<< followed by an identifier. But here identifier is enclosed in single-quote, **e.g. <<<'EXP'**. Newdoc follows the same rule as heredocs.

The difference between newdoc and heredoc is that - Newdoc is a **single-quoted string** whereas heredoc is a **double-quoted string**.

> *Note:* ***Newdoc works as single quotes.***

Example-1:

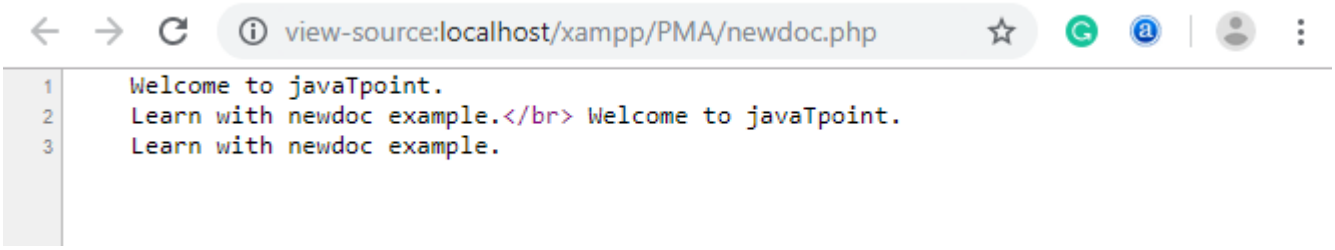
```
1.  <?php
2.     $str = <<<'DEMO'
3.     Welcome to javaTpoint.
4.           Learn with newdoc example.
5.  DEMO;
6.  echo $str;
7.  echo '</br>';
8.
9.  echo <<< 'Demo'    // Here we are not storing string content in variable str.
10.    Welcome to javaTpoint.
```

11. Learn with newdoc example.
12. Demo;
13. ?>

Output:

```
Welcome to javaTpoint. Learn with newdoc example.
Welcome to javaTpoint. Learn with newdoc example.
```

Go to view page source and see the source of the program.



Example

The below example shows that newdoc does not print the variable's value.

```
1. <?php
2. class heredocExample{
3.         var $demo;
4.         var $example;
5.         function __construct()
6.         {
7.                 $this->demo = 'DEMO';
8.                 $this->example = array('Example1', 'Example2', 'Example3');
9.         }
10.    }
11.    $heredocExample = new heredocExample();
12.    $name =  'Gunjan';
13.
14.    echo <<<ECO
15.    My name is "$name". I am printing some $heredocExample->demo example.
16.    Now, I am printing {$heredocExample->example[1]}.
```

```
17.     It will print a capital 'A': \x41
18. ECO;
19.  ?>
```

Output:

The output of the above program will be like:

```
My name is "$name". I am printing some $heredocExample->demo example.
Now, I am printing {$heredocExample->example[1]}.
It will print a capital 'A': \x41
```

Note: ***newdoc supported by PHP 5.3.0+ versions.***

Invalid Example

We cannot use any whitespace or tab before and after the identifier and semicolon, means identifier must not be indented. The identifier must begin from the new line. It is also invalid in newdoc same as heredoc.

```
1. <?php
2.     $str = <<<'Demo'
3. It is Invalid example
4. Demo;  //Invalid code as whitespace or tab is not valid before closing identifier
5. echo $str;
6. ?>
```

This code will generate an error.

Output:

```
Parse error: syntax error, unexpected end of file in
C:\xampp\htdocs\xampp\PMA\newdoc.php on line 7
```

PHP Operators

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

```
1. $num=10+20;//+ is the operator and 10,20 are operands
```

In the above example, + is the binary + operator, 10 and 20 are operands and $num is variable.

PHP Operators can be categorized in following forms:

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Comparison Operators
- Incrementing/Decrementing Operators
- Logical Operators
- String Operators
- Array Operators
- Type Operators
- Execution Operators
- Error Control Operators

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

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

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

The exponentiation (**) operator has been introduced in PHP 5.6.

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 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

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

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

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

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

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

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

Type Operators

The type operator **instanceof** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to. It is used in object-oriented programming.

```
1. <?php
2.     //class declaration
```

```
3.  class Developer
4.  {}
5.  class Programmer
6.  {}
7.  //creating an object of type Developer
8.  $charu = new Developer();
9.
10. //testing the type of object
11. if( $charu instanceof Developer)
12. {
13.     echo "Charu is a developer.";
14. }
15. else
16. {
17.     echo "Charu is a programmer.";
18. }
19. echo "</br>";
20. var_dump($charu instanceof Developer);        //It will return true.
21. var_dump($charu instanceof Programmer);      //It will return false.
22. ?>
```

Output:

```
Charu is a developer.
bool(true) bool(false)
```

Execution Operators

PHP has an execution operator **backticks (``)**. PHP executes the content of backticks as a shell command. Execution operator and **shell_exec()** give the same result.

Operator	Name	Example	Explanation
``	backticks	echo `dir`;	Execute the shell command and return the result. Here, it will show the directories available in current folder.

Note: Note that backticks (``) are not single-quotes.

Error Control Operators

PHP has one error control operator, i.e., **at (@) symbol**. Whenever it is used with an expression, any error message will be ignored that might be generated by that expression.

Operator	Name	Example	Explanation
@	at	@file ('non_existent_file')	Intentional file error

PHP Operators Precedence

Let's see the precedence of PHP operators with associativity.

Operators	Additional Information	Associativity
clone new	clone and new	non-associative
[	array()	left
**	arithmetic	right
++ -- ~ (int) (float) (string) (array) (object) (bool) @	increment/decrement and types	right
instanceof	types	non-associative
!	logical (negation)	right
* / %	arithmetic	left

+ - .	arithmetic and string concatenation	left
<< >>	bitwise (shift)	left
< <= > >=	comparison	non-associative
== != === !== <>	comparison	non-associative
&	bitwise AND	left
^	bitwise XOR	left
|	bitwise OR	left
&&	logical AND	left
||	logical OR	left
?:	ternary	left
= += -= *= **= /= .= %= &= |= ^= <<= >>= =>	assignment	right
and	logical	left
xor	logical	left
or	logical	left
,	many uses (comma)	left

PHP | Loops

Like any other language, loop in PHP is used to execute a statement or a block of statements, multiple times until and unless a specific condition is met. This helps the user to save both time and effort of writing the same code multiple times.

PHP supports four types of looping techniques;

1. for loop
2. while loop
3. do-while loop
4. foreach loop

Let us now learn about each of the above mentioned loops in details:

1. **for loop**: This type of loops is used when the user knows in advance, how many times the block needs to execute. That is, the number of iterations is known beforehand. These type of loops are also known as entry-controlled loops. There are three main parameters to the code, namely the initialization, the test condition and the counter.
   Syntax:

```
for (initialization expression; test condition; update expression)
{

    // code to be executed

}
```

   In for loop, a loop variable is used to control the loop. First initialize this loop variable to some value, then check whether this variable is less than or greater than counter value. If statement is true, then loop body is executed and loop variable gets updated . Steps are repeated till exit condition comes.

   - **Initialization Expression**: In this expression we have to initialize the loop counter to some value. for example: $num = 1;
   - **Test Expression**: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: $num <= 10;
   - **Update Expression**: After executing loop body this expression increments/decrements the loop variable by some value. for example: $num += 2;

   Example:

```php
<?php



// code to illustrate for loop

for ($num = 1; $num <= 10; $num += 2) {

    echo "$num \n";

}



?>
```

Output:

1

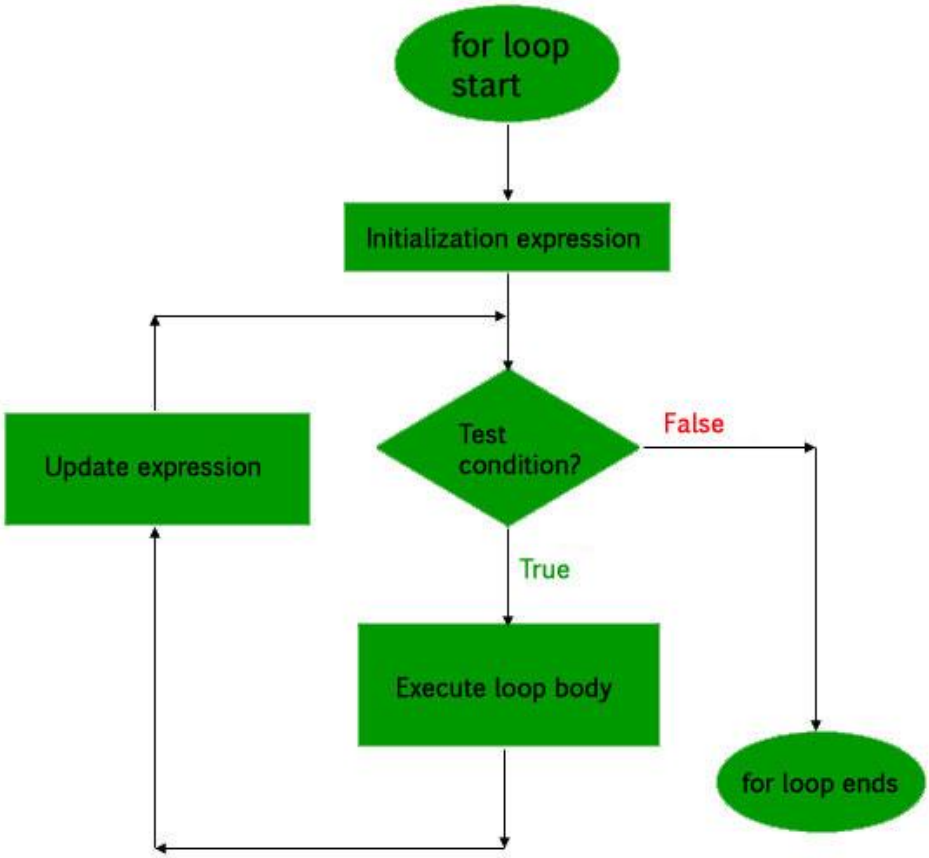
3

5

7

9

Flow Diagram:



2. **while loop**: The while loop is also an entry control loop like for loops i.e., it first checks the condition at the start of the loop and if its true then it enters the loop and executes the block of statements, and goes on executing it as long as the condition holds true.
   Syntax:

```
while (if the condition is true) {
    // code is executed
}
```

   Example:

```
<?php


// PHP code to illustrate while loops
```

```
$num = 2;



while ($num < 12) {

    $num += 2;

    echo $num, "\n";

}



?>
```

Output:

4

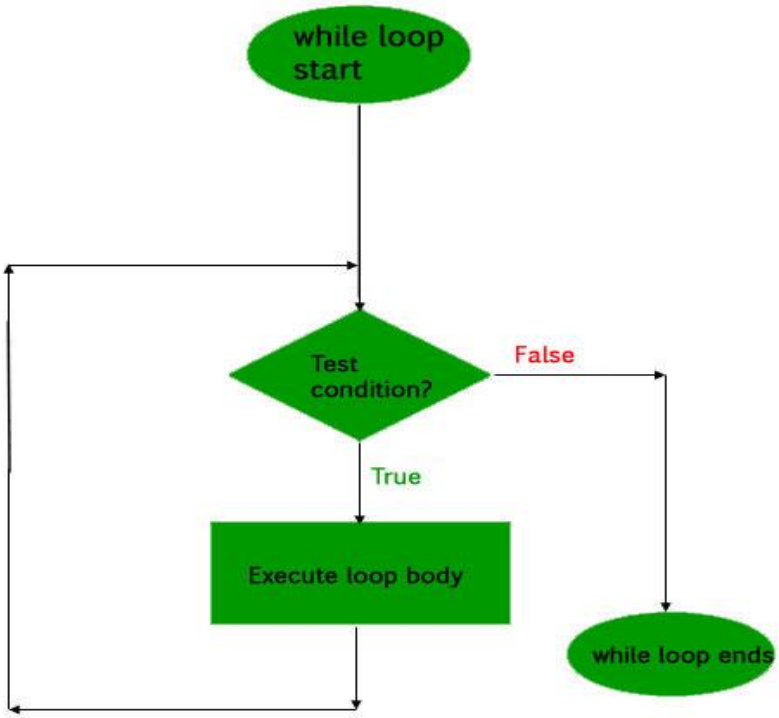
6

8

10

12

Flowchart:



3. **do-while loop**: This is an exit control loop which means that it first enters the loop, executes the statements, and then checks the condition. Therefore, a statement is executed at least once on using the do…while loop. After executing once, the program is executed as long as the condition holds true.
   Syntax:

```
do {


    //code is executed


} while (if condition is true);
```

   Example:

```
<?php


// PHP code to illustrate do...while loops
```

```
$num = 2;

do {

    $num += 2;

    echo $num, "\n";

} while ($num < 12);



?>
```

Output:

4

6

8

10

12

This code would show the difference between while and do…while loop.

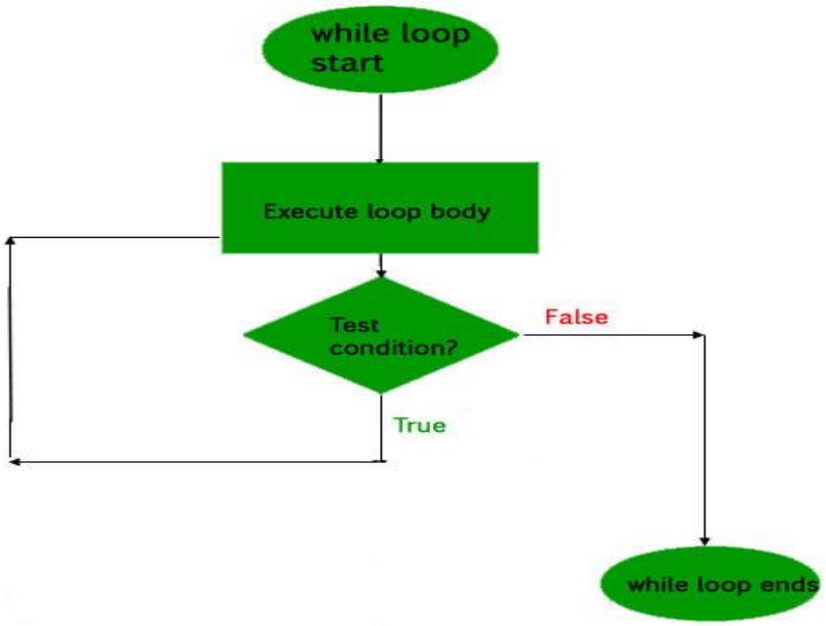
```
<?php



// PHP code to illustrate the difference of two loops

$num = 2;



// In case of while

while ($num != 2) {
```

```
    echo "In case of while the code is skipped";
    echo $num, "\n";


}
// In case of do...while
do {


    $num++;
    echo "The do...while code is executed atleast once ";


} while($num == 2);


?>
```

Output:

```
The code is executed at least once
```

Flowchart:



4. **foreach loop**: This loop is used to iterate over arrays. For every counter of loop, an array element is assigned and the next counter is shifted to the next element.
   Syntax:

```
foreach (array_element as value) {
   //code to be executed
}
```

Example:

```
<?php


    $arr = array (10, 20, 30, 40, 50, 60);
    foreach ($arr as $val) {
        echo "$val \n";
    }
```

```
    $arr = array ("Ram", "Laxman", "Sita");
    foreach ($arr as $val) {
        echo "$val \n";
    }

?>
```

Output:

```
10
20
30
40
50
60
Ram
Laxman
Sita
```

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

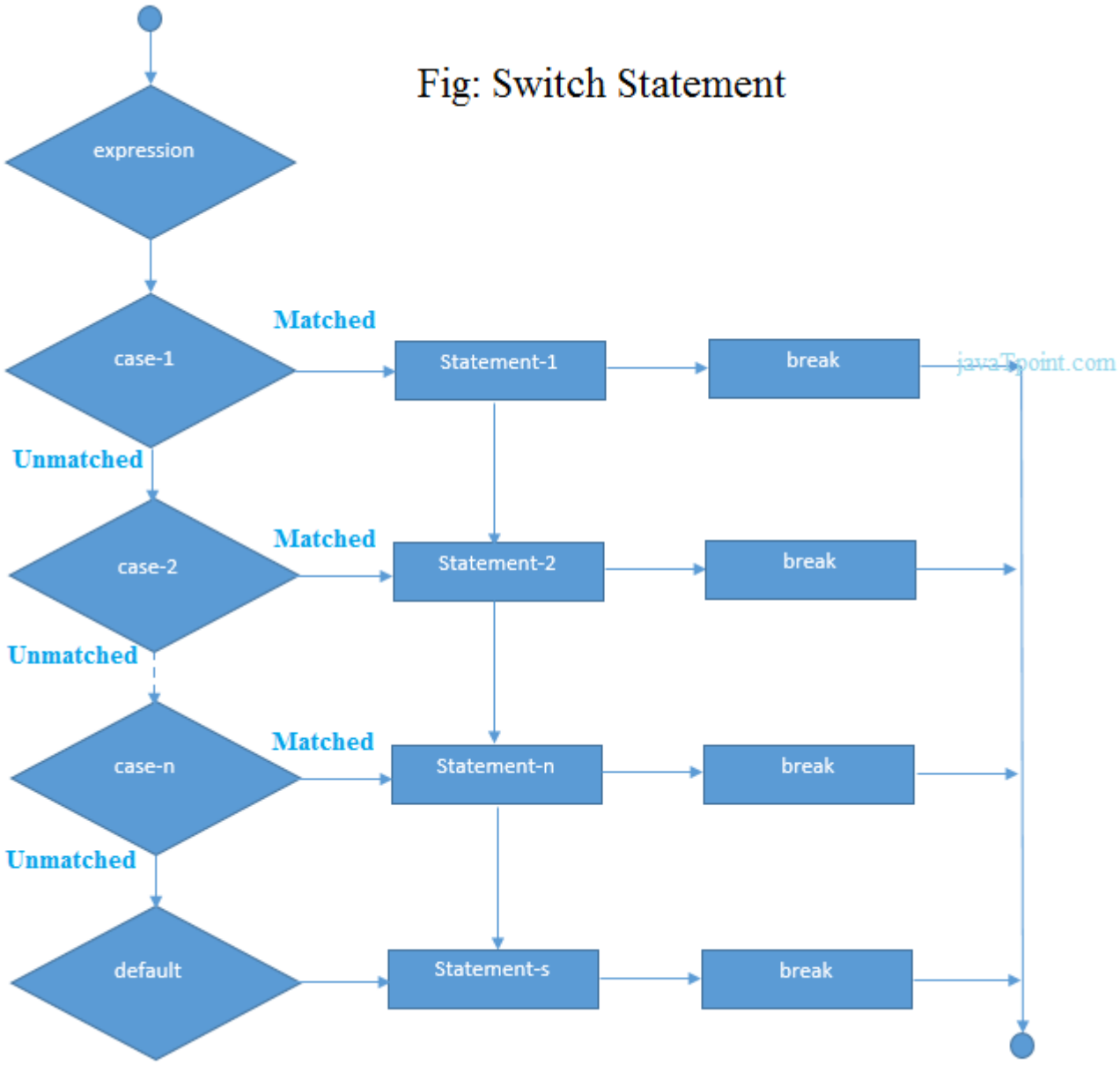
1. **switch**(expression){
2. **case** value1:
3. //code to be executed
4. **break**;

```
5.  case value2:
6.   //code to be executed
7.   break;
8.  ......
9.  default:
10.  code to be executed if all cases are not matched;
11. }
```

Important points to be noticed about switch case:

1. The **default** is an optional statement. Even 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 statement.
4. The **break** statement is optional to use in switch. If break is not used, all the statements will execute after finding matched case value.
5. PHP allows you to use number, character, string, as well as functions in switch expression.
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 error.

PHP Switch Flowchart



PHP Switch Example

```
1. <?php
2. $num=20;
3. switch($num){
4. case 10:
5. echo("number is equals to 10");
6. break;
```

```
7.  case 20:
8.  echo("number is equal to 20");
9.  break;
10. case 30:
11. echo("number is equal to 30");
12. break;
13. default:
14. echo("number is not equal to 10, 20 or 30");
15. }
16. ?>
```

Output:

```
number is equal to 20
```

PHP If Else

PHP if else statement is used to test condition. There are various ways to use if statement in PHP.

- if
- if-else
- if-else-if
- nested if

PHP If Statement

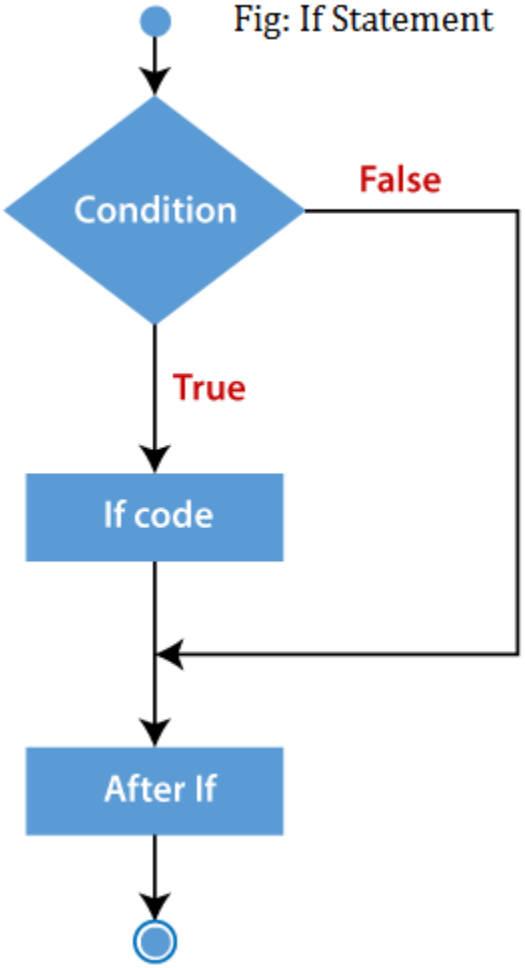
PHP if statement allows conditional execution of code. It is executed if condition is true.

If statement is used to executes the block of code exist inside the if statement only if the specified condition is true.

Syntax

```
1. if(condition){
2. //code to be executed
3. }
```

Flowchart



Example

```
1. <?php
2. $num=12;
3. if($num<100){
4. echo "$num is less than 100";
5. }
6. ?>
```

Output:

```
12 is less than 100
```

PHP If-else Statement

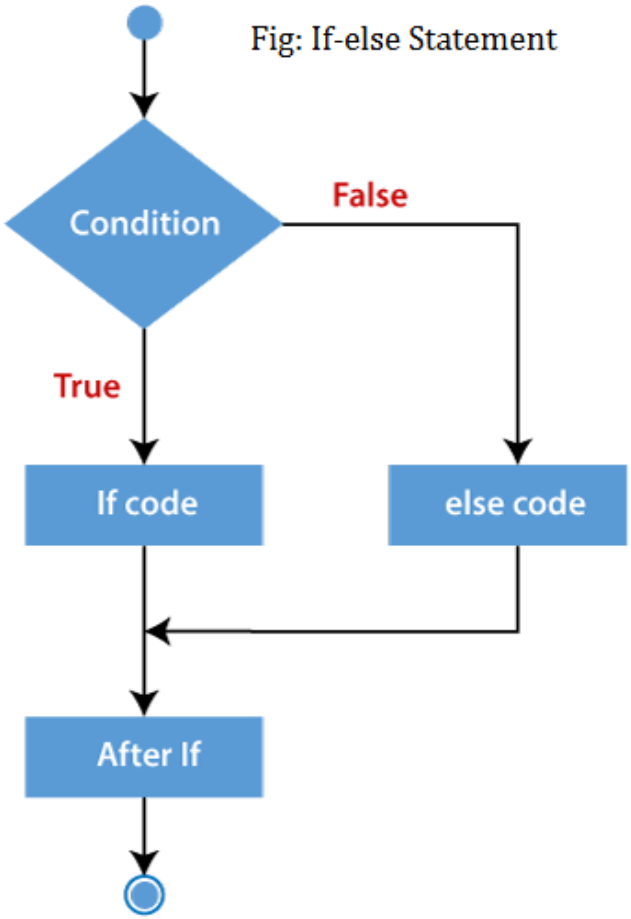
PHP if-else statement is executed whether condition is true or false.

If-else statement is slightly different from if statement. It executes one block of code if the specified condition is **true** and another block of code if the condition is **false**.

Syntax

```
1. if(condition){
2. //code to be executed if true
3. }else{
4. //code to be executed if false
5. }
```

Flowchart



Example

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

Output:

```
12 is even number
```

PHP If-else-if Statement

The PHP if-else-if is a special statement used to combine multiple if?.else statements. So, we can check multiple conditions using this statement.

Syntax

```
if (condition1){
//code to be executed if condition1 is true
} elseif (condition2){
//code to be executed if condition2 is true
} elseif (condition3){
//code to be executed if condition3 is true
....
}  else{
//code to be executed if all given conditions are false
}
```

Flowchart

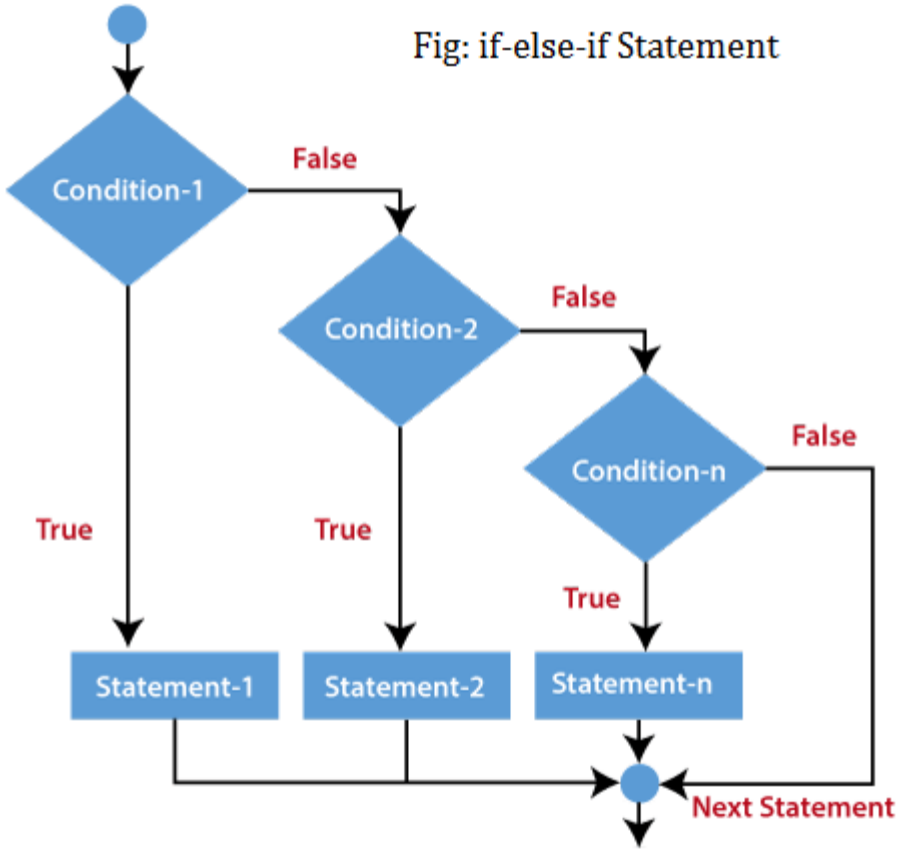


Fig: if-else-if Statement

Example

```
1.  <?php
2.      $marks=69;
3.      if ($marks<33){
4.          echo "fail";
5.      }
6.      else if ($marks>=34 && $marks<50) {
7.          echo "D grade";
8.      }
9.      else if ($marks>=50 && $marks<65) {
10.         echo "C grade";
11.     }
12.     else if ($marks>=65 && $marks<80) {
13.         echo "B grade";
14.     }
```

```
15.    else if ($marks>=80 && $marks<90) {
16.        echo "A grade";
17.    }
18.    else if ($marks>=90 && $marks<100) {
19.        echo "A+ grade";
20.    }
21.   else {
22.        echo "Invalid input";
23.    }
24. ?>
```

Output:

```
B Grade
```

PHP nested if Statement

The nested if statement contains the if block inside another if block. The inner if statement executes only when specified condition in outer if statement is **true**.

Syntax

```
1. if (condition) {
2. //code to be executed if condition is true
3. if (condition) {
4. //code to be executed if condition is true
5. }
6. }
```

Flowchart

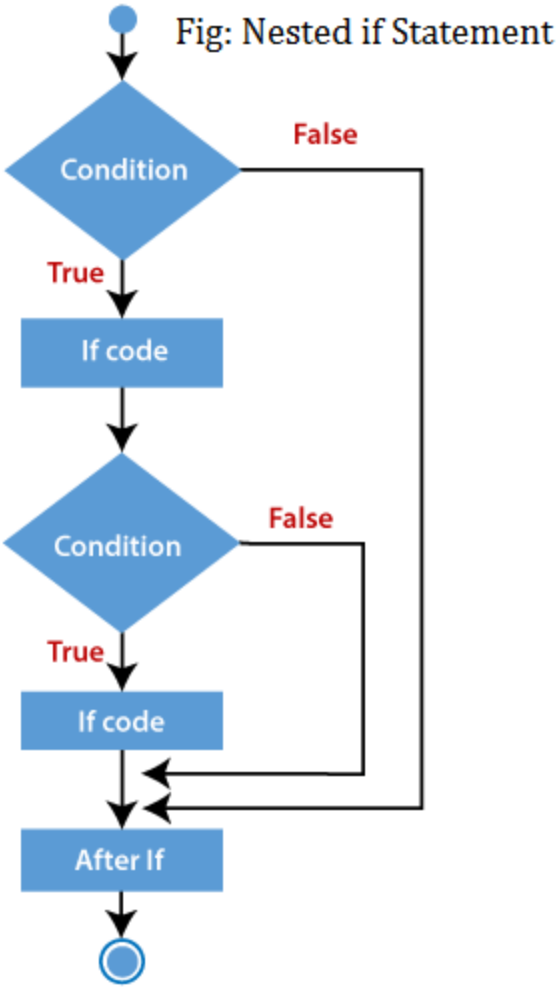


Fig: Nested if Statement

Example

```
1.  <?php
2.              $age = 23;
3.      $nationality = "Indian";
4.      //applying conditions on nationality and age
5.      if ($nationality == "Indian")
6.      {
7.          if ($age >= 18) {
8.              echo "Eligible to give vote";
9.          }
10.         else {
11.             echo "Not eligible to give vote";
12.         }
```

```
13.     }
14. ?>
```

Output:

```
Eligible to give vote
```

PHP Arrays

PHP array is an ordered map (contains value on the basis of key). It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

Less Code: We don't need to define multiple variables.

Easy to traverse: By the help of single loop, we can traverse all the elements of an array.

Sorting: We can sort the elements of array.

PHP Array Types

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

PHP Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

1st way:

```
$season=array("summer","winter","spring","autumn");
```

2nd way:

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

Example

File: array1.php

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

File: array2.php

```
<?php
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

Click me for more details...

PHP Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

1st way:

```
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
```

2nd way:

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
```

Example

File: arrayassociative1.php

```
<?php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

File: arrayassociative2.php

```
<?php
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
```

8. ?>

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

PHP Multidimensional Array

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.

Definition

```
1. $emp = array
2.  (
3.  array(1,"sonoo",400000),
4.  array(2,"john",500000),
5.  array(3,"rahul",300000)
6.  );
```

PHP Multidimensional Array Example

Let's see a simple example of PHP multidimensional array to display following tabular data. In this example, we are displaying 3 rows and 3 columns.

Id	Name	Salary
1	sonoo	400000
2	john	500000
3	rahul	300000

File: multiarray.php

```
1. <?php
2. $emp = array
```

```
3.    (
4.    array(1,"sonoo",400000),
5.    array(2,"john",500000),
6.    array(3,"rahul",300000)
7.    );
8.
9.  for ($row = 0; $row < 3; $row++) {
10.   for ($col = 0; $col < 3; $col++) {
11.     echo $emp[$row][$col]."  ";
12.   }
13.   echo "<br/>";
14. }
15. ?>
```

Output:

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.

In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

Advantage of PHP Functions

Code Reusability: PHP functions are defined only once and can be invoked many times, like in other programming languages.

Less Code: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

Easy to understand: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare user-defined functions.

Syntax

```
1. function functionname(){
2. //code to be executed
3. }
```

Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.

PHP Functions Example

File: function1.php

```
1. <?php
2. function sayHello(){
3. echo "Hello PHP Function";
4. }
5. sayHello();//calling function
6. ?>
```

Output:

```
Hello PHP Function
```

PHP Function Arguments

We can pass the information in PHP function through arguments which is separated by comma.

PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and **Variable-length argument list**.

Let's see the example to pass single argument in PHP function.

File: functionarg.php

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
sayHello("Sonoo");
sayHello("Vimal");
sayHello("John");
?>
```

Output:

```
Hello Sonoo
Hello Vimal
Hello John
```

Let's see the example to pass two argument in PHP function.

File: functionarg2.php

```
<?php
function sayHello($name,$age){
echo "Hello $name, you are $age years old<br/>";
}
sayHello("Sonoo",27);
sayHello("Vimal",29);
sayHello("John",23);
?>
```

Output:

```
Hello Sonoo, you are 27 years old
Hello Vimal, you are 29 years old
Hello John, you are 23 years old
```

PHP Call By Reference

Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Let's see a simple example of call by reference in PHP.

File: functionref.php

```
<?php
function adder(&$str2)
{
    $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

Output:

```
Hello Call By Reference
```

PHP Function: Default Argument Value

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

File: functiondefaultarg.php

```
<?php
function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

Output:

```
Hello Rajesh
Hello Sonoo
Hello John
```

PHP Function: Returning Value

Let's see an example of PHP function that returns value.

File: functiondefaultarg.php

```
1. <?php
2. function cube($n){
3. return $n*$n*$n;
4. }
5. echo "Cube of 3 is: ".cube(3);
6. ?>
```

Output:

```
Cube of 3 is: 27
```

PHP Form Handling

We can create and use forms in PHP. To get form data, we need to use PHP superglobals $_GET and $_POST.

The form request may be get or post. To retrieve data from get request, we need to use $_GET, for post request $_POST.

PHP Get Form

Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.

Let's see a simple example to receive data from get request in PHP.

File: form1.html

```
1. <form action="welcome.php" method="get">
```

```
2. Name: <input type="text" name="name"/>
3. <input type="submit" value="visit"/>
4. </form>
```

File: welcome.php

```
1. <?php
2. $name=$_GET["name"];//receiving name field value in $name variable
3. echo "Welcome, $name";
4. ?>
```

PHP Post Form

Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.

The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

Let's see a simple example to receive data from post request in PHP.

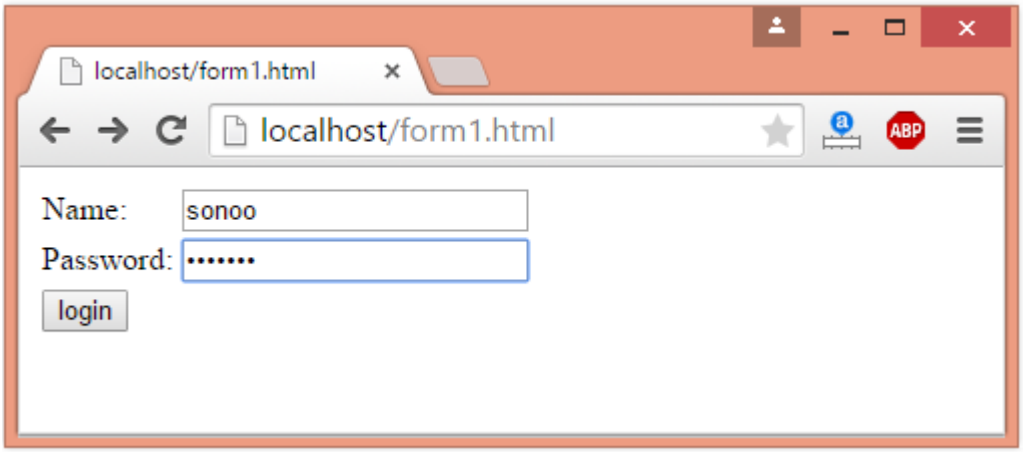
File: form1.html

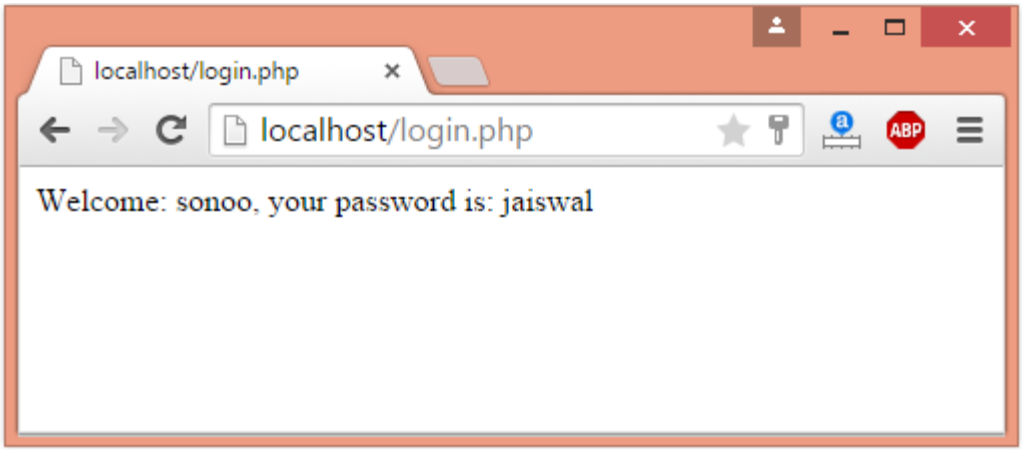
```
1. <form action="login.php" method="post">
2. <table>
3. <tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
4. <tr><td>Password:</td><td> <input type="password" name="password"/></td></tr>
5. <tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>
6. </table>
7. </form>
```

File: login.php

```
1. <?php
2. $name=$_POST["name"];//receiving name field value in $name variable
3. $password=$_POST["password"];//receiving password field value in $password variable
4.
5. echo "Welcome: $name, your password is: $password";
6. ?>
```

Output:





PHP Mail

PHP mail() function is used to send email in PHP. You can send text message, html message and attachment with message using PHP mail() function.

PHP mail() function

Syntax

1. bool mail (string $to , string $subject , string $message [, string $additional_headers [, string $additional_parameters]])

   $to: specifies receiver or receivers of the mail. The receiver must be specified one of the following forms.

   - user@example.com
   - user@example.com, anotheruser@example.com

- User <user@example.com>
- User <user@example.com>, Another User <anotheruser@example.com>

$subject: represents subject of the mail.

$message: represents message of the mail to be sent.

> ***Note: Each line of the message should be separated with a CRLF (\r\n) and lines should not be larger than 70 characters.***

$additional_headers (optional): specifies the additional headers such as From, CC, BCC etc. Extra additional headers should also be separated with CRLF (\r\n).

PHP Mail Example

File: mailer.php

```
<?php
  ini_set("sendmail_from", "sonoojaiswal@javatpoint.com");
  $to = "sonoojaiswal1987@gmail.com";//change receiver address
  $subject = "This is subject";
  $message = "This is simple text message.";
  $header = "From:sonoojaiswal@javatpoint.com \r\n";

  $result = mail ($to,$subject,$message,$header);

  if( $result == true ){
    echo "Message sent successfully...";
  }else{
    echo "Sorry, unable to send mail...";
  }
?>
```

If you run this code on the live server, it will send an email to the specified receiver.

PHP Mail: Send HTML Message

To send HTML message, you need to mention Content-type **text/html** in the message header.

```php
<?php
  $to = "abc@example.com";//change receiver address
  $subject = "This is subject";
  $message = "<h1>This is HTML heading</h1>";

  $header = "From:xyz@example.com \r\n";
  $header .= "MIME-Version: 1.0 \r\n";
  $header .= "Content-type: text/html;charset=UTF-8 \r\n";

  $result = mail ($to,$subject,$message,$header);

  if( $result == true ){
    echo "Message sent successfully...";
  }else{
    echo "Sorry, unable to send mail...";
  }
?>
```

PHP Mail: Send Mail with Attachment

To send message with attachment, you need to mention many header information which is used in the example given below.

```php
<?php
  $to = "abc@example.com";
  $subject = "This is subject";
  $message = "This is a text message.";
  # Open a file
  $file = fopen("/tmp/test.txt", "r" );//change your file location
  if( $file == false )
```

```
{
   echo "Error in opening file";
   exit();
}
# Read the file into a variable
$size = filesize("/tmp/test.txt");
$content = fread( $file, $size);

# encode the data for safe transit
# and insert \r\n after every 76 chars.
$encoded_content = chunk_split( base64_encode($content));

# Get a random 32 bit number using time() as seed.
$num = md5( time() );

# Define the main headers.
$header = "From:xyz@example.com\r\n";
$header .= "MIME-Version: 1.0\r\n";
$header .= "Content-Type: multipart/mixed; ";
$header .= "boundary=$num\r\n";
$header .= "--$num\r\n";

# Define the message section
$header .= "Content-Type: text/plain\r\n";
$header .= "Content-Transfer-Encoding:8bit\r\n\n";
$header .= "$message\r\n";
$header .= "--$num\r\n";

# Define the attachment section
$header .= "Content-Type:  multipart/mixed; ";
$header .= "name=\"test.txt\"\r\n";
$header .= "Content-Transfer-Encoding:base64\r\n";
$header .= "Content-Disposition:attachment; ";
$header .= "filename=\"test.txt\"\r\n\n";
```

```
42.  $header .= "$encoded_content\r\n";
43.  $header .= "--$num--";
44.
45.  # Send email now
46.  $result = mail ( $to, $subject, "", $header );
47.  if( $result == true ){
48.     echo "Message sent successfully...";
49.   }else{
50.     echo "Sorry, unable to send mail...";
51.   }
52. ?>
```

How to connect ODBC Database with PHP?

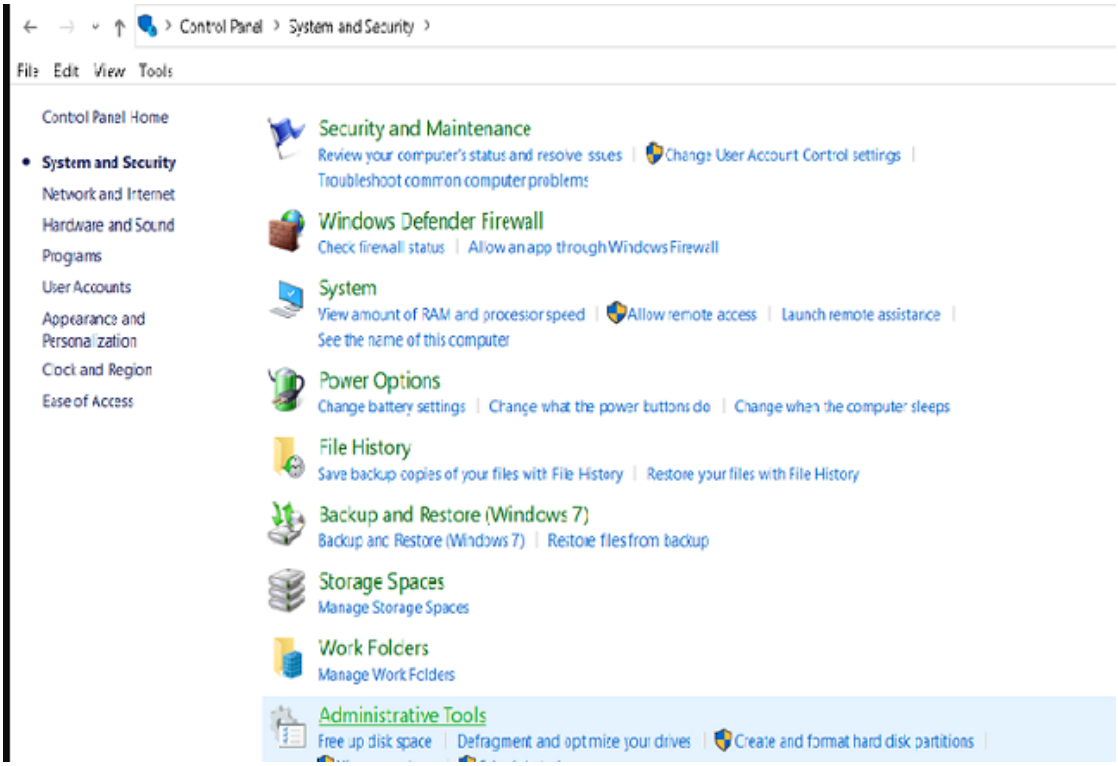
The Microsoft Open Database Connectivity (ODBC) is a C application programming language interface. So the applications can access data from different database management systems. The designers of ODBC aimed to make it independent of database systems and operating systems. In this article, we are using the Microsoft Access database to establish a connection.

Requirements: If ODBC Drivers are installed or the old version is installed, then update to the latest version or install the driver first

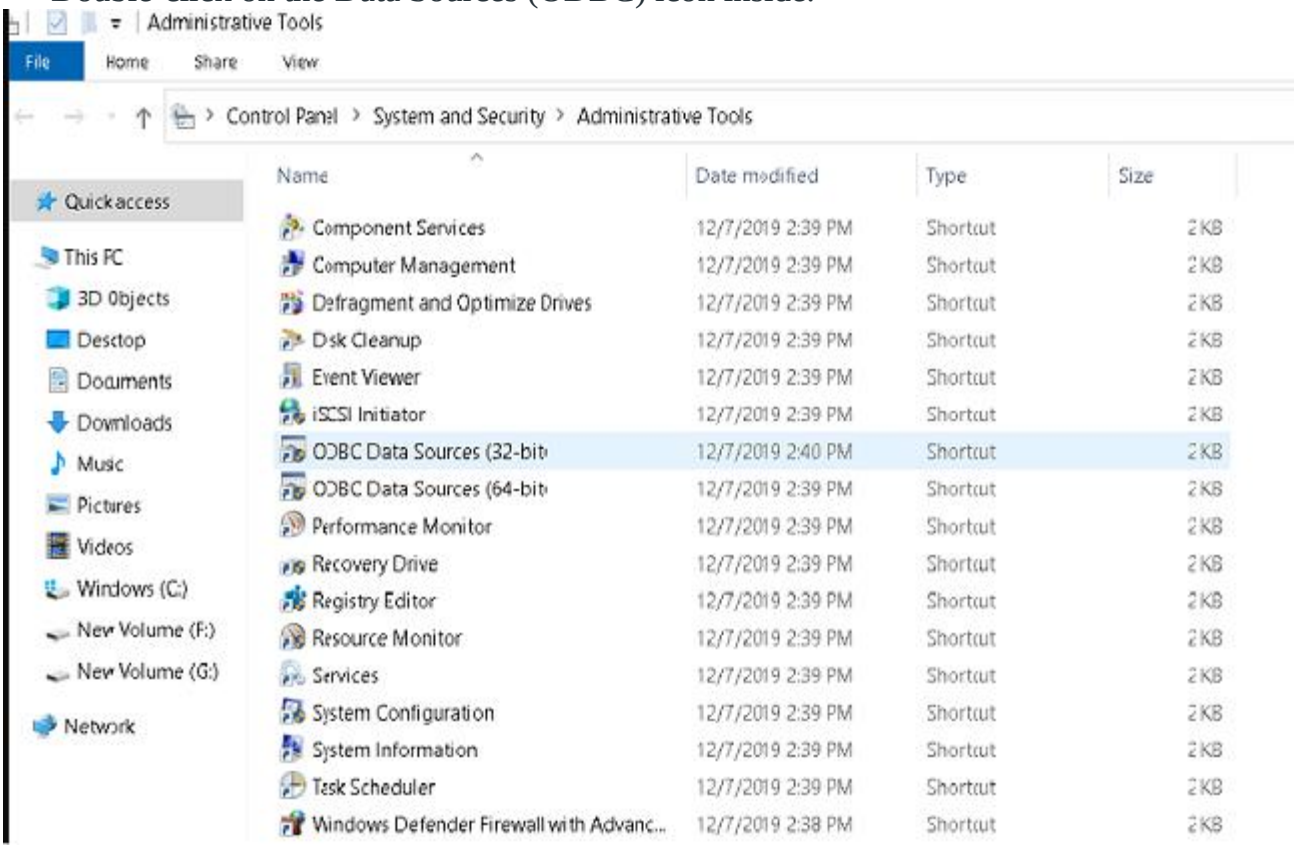
- Start XAMPP server by starting Apache
- Write PHP script for Access database connection.
- Run it in a local browser.
- The database is successfully connected.

Steps to follow for ODBC connection to an MS Access Database:

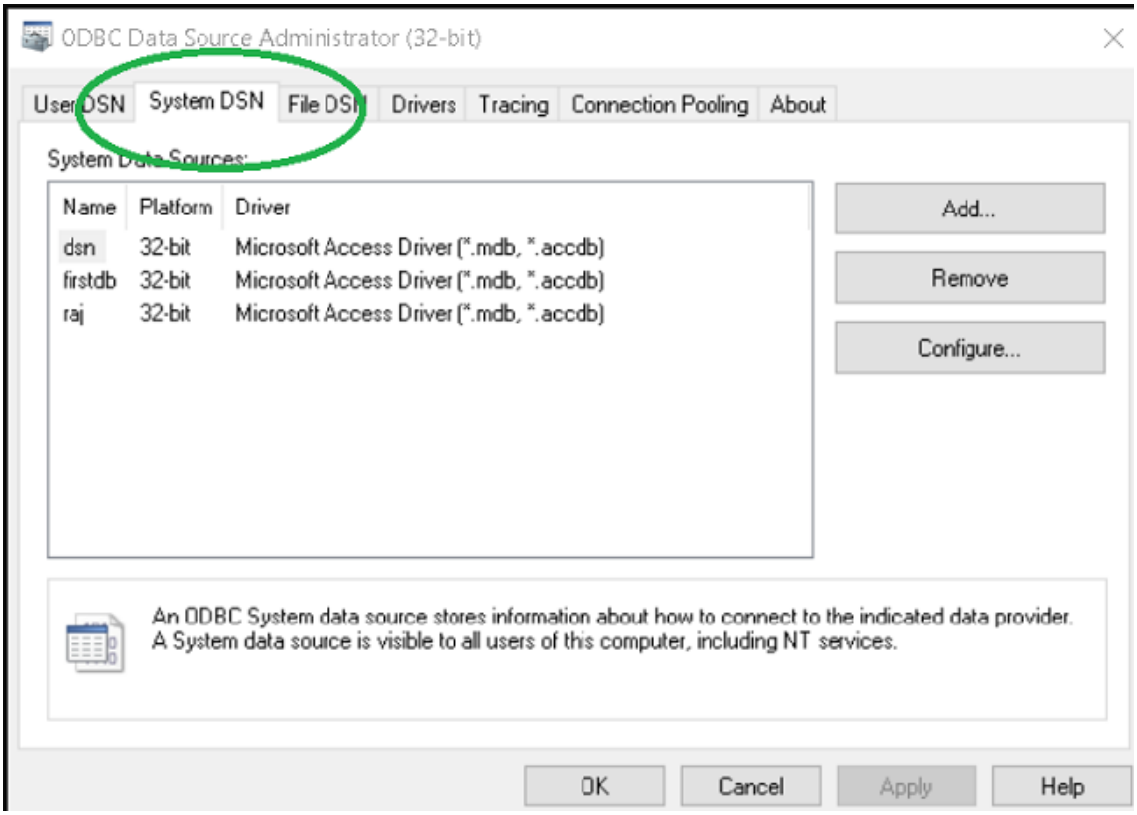
- Open the Administrative tools icon in your control panel



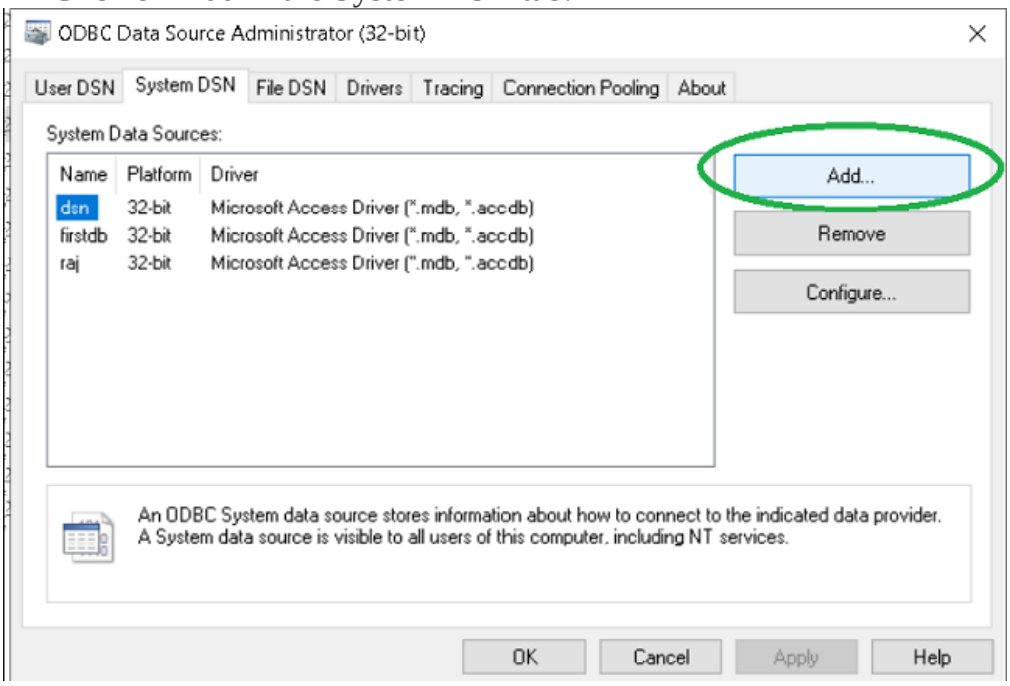
- Double-click on the Data Sources (ODBC) icon inside.



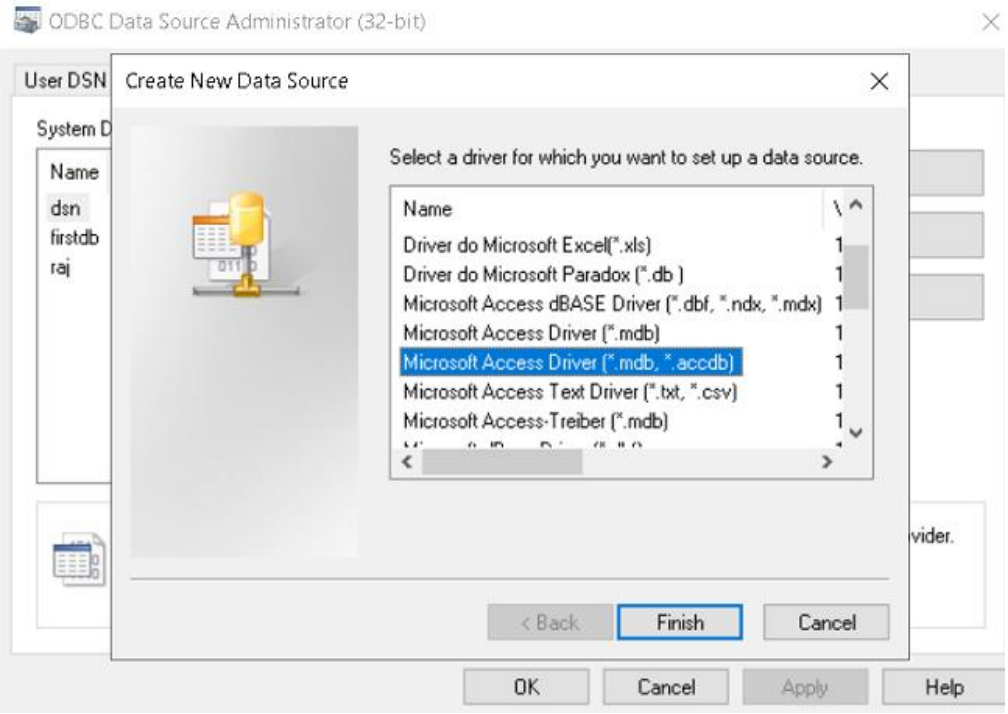
- Choose the System DSN tab.



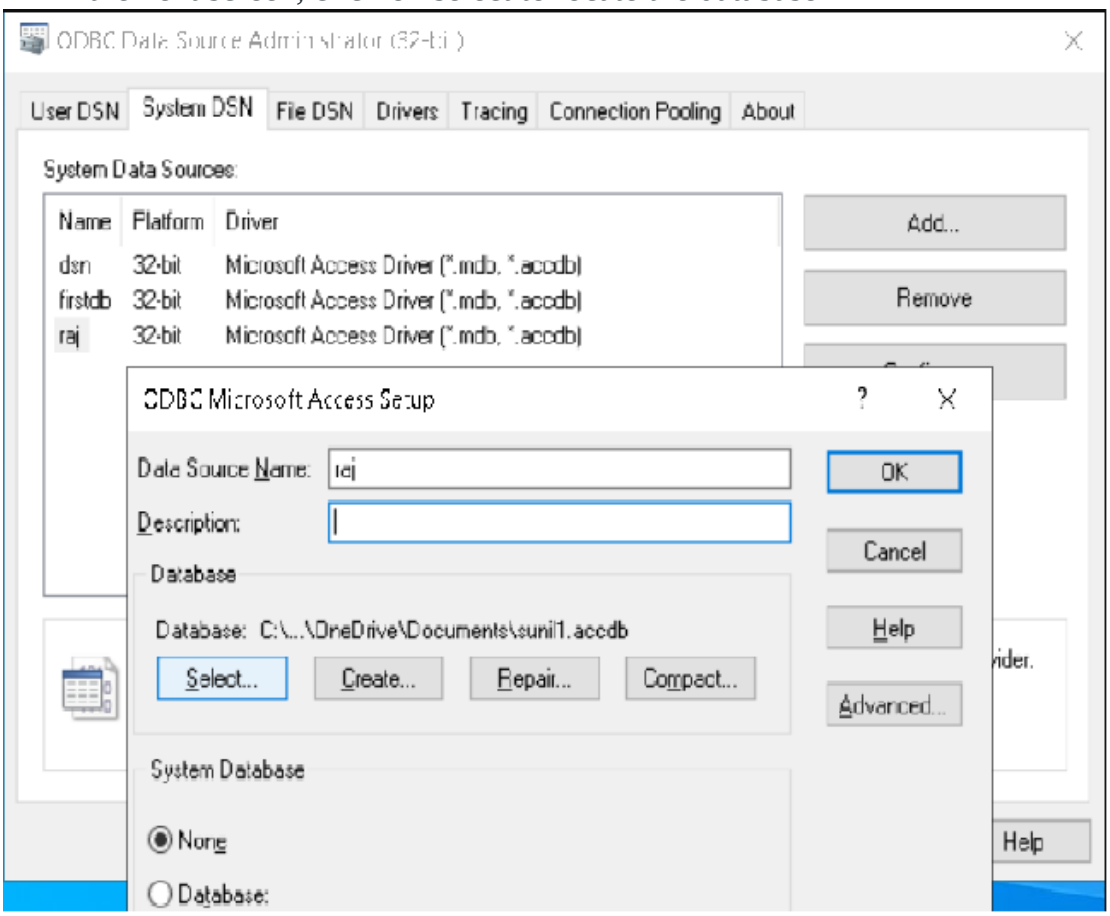
- Click on Add in the System DSN tab.



- Select the Microsoft Access Driver, Click Finish



- In the next screen, click on select to locate the database



- Give the database a Data Source Name (DSN) as shown in the above image. Click ok.

Example: The following example demonstrates the simple connection script.

- PHP

```
<!DOCTYPE html>

<html>



<body>

 <?php

   //Storing DSN(Data Source Name created)

  $dsn="raj";

  $user="root";

  $password="";



   //storing connection id in $conn

  $conn=odbc_connect($dsn,$user, $password);



  //Checking connection id or reference

  if (!$conn)

   {

   echo (die(odbc_error()));
```

```
    }

    else

   {

        echo "Connection Successful !";

   }

   //Resource releasing

   odbc_close($conn);

   ?>



</body>

</html>
```

Below output will be shown on the browser

Output:

```
Connection Successful !
```