MODULE 3

- PHP stands for Hypertext Preprocessor.
- PHP was created by **Rasmus Lerdorf**
- PHP 8.2 is the latest PHP version
- 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 language.

PHP Features

PHP is very popular language because of its simplicity and open source.

Performance:

PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance.

Open Source:

PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

Familiarity with syntax:

PHP has easily understandable syntax. Programmers are comfortable coding with it.

Embedded:

PHP code can be easily embedded within HTML tags and script.

Platform Independent:

PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

Database Support:

HP supports all the leading databases such as MySQL, SQLite, ODBC, etc.

Error Reporting -

PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

loosely Typed Language:

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

Web servers Support:

PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

Security:

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

Control:

Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

A Helpful PHP Community:

It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

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".

```
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Insert PHP code in HTML document using include or require keyword

- PHP is an HTML embedded server-side scripting language.
- We can insert any PHP file into the HTML code by using two keywords that are ‘Include’ and ‘Require’.
- PHP include() function: This function is used to copy all the contents of a file called within the function, text wise into a file from which it is called. This happens before the server executes the code.
- SYNTAX

  include 'php filename';

natural.php

```
<?php
$i = 1;
echo "the first 10 natural numbers are:";
for($i = 1; $i <= 10; $i++) {
        echo $i;
}
?>
```

Insert the above code into an HTML document by using the include keyword

```
<!DOCTYPE html>
<html>
<head>
        <title>inserting PHP code into html</title>
</head>
<body>
<h2>Natural numbers</h2>
<?php include 'natural.php'?>
</body>
</html>
```

- PHP require() function: The require() function performs same as the include() function. It also takes the file that is required and copies the whole code into the file from where the require() function is called
- **Syntax:**

  require 'php filename'

Example 2: We can insert the PHP code into HTML Document by directly writing in the body tag of the HTML document.

```
<!DOCTYPE html>
<html>
<head>
        <title>inserting php code into html </title>
</head>
<body>
<h2>natural numbers</h2>
<?php
$i = 1;
echo"the first 10 natural numbers are:";
for($i = 1; $i <= 10; $i++) {
        echo $i;
}
?>
</body>
</html>
```

To insert the above code into an HTML document by using the ‘require’ keyword

```
<!DOCTYPE html>
<html>
<head>
<title>inserting PHP code into html</title>
</head>
<body>
<h2>Natural numbers</h2>
<?php require 'natural.php'?>
</body>
</html>
```

PHP Comments

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Syntax for single-line comments:

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

</body>
</html>
```

Syntax for multiple-line comments:

```
<!DOCTYPE html>
<html>
<body>
<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>
</body>
</html>
```

PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

```
<?php
$txt = "Hello world!";
$x = 5;
```

```
$y = 10.5;
?>
```

after the execution of the statements above, the variable $txt will hold the value Hello world!, the variable $x will hold the value 5, and the variable $y will hold the value 10.5.

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)

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

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

```
<?php
$x = 5; // global scope

function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
```

```
echo "<p>Variable x outside function is: $x</p>";
?>
```

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

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>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

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

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

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

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

```
<?php
$x = 5;
$y = 10;
function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

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

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

```
<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();
?>
```

```
0
1
2
```

Difference between echo and print

echo

echo is a statement, which is used to display the output.

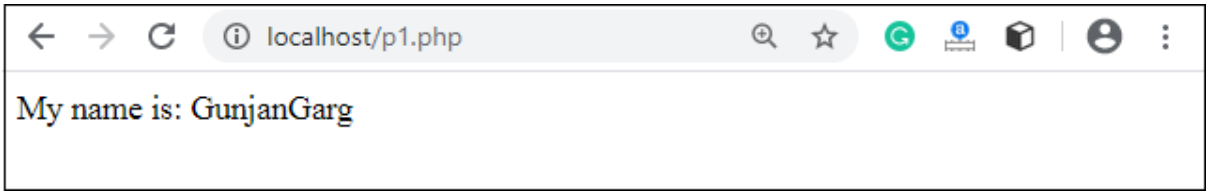
- echo can be used with or without parentheses.
- echo does not return any value.

- We can pass multiple strings separated by comma (,) in echo.
- echo is faster than print statement.

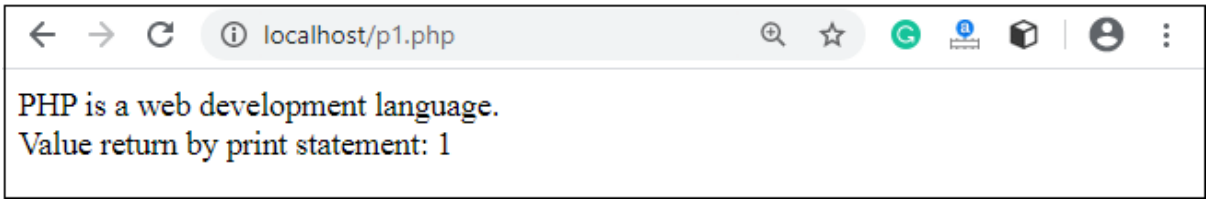
print

- print is also 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 echo statement.

```
<?php
    $fname = "Gunjan";
    $lname = "Garg";
    Echo  "My name is: ".$fname,$lname;
?>
```




```
<?php
    $lang = "PHP";
    $ret = print $lang." is a web development language.";
    print "</br>";
print "Value return by print statement: ".$ret;
?>
```



PHP 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 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 works like 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:

```
<?php
  if (TRUE)
    echo "This condition is TRUE.";
  if (FALSE)
    echo "This condition is FALSE.";
?>
```

Output:

```
This condition is TRUE.
```

PHP Integer

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

Rules for integer:

- An integer can be either positive or negative.
- An integer must not contain decimal point.

- Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
- The range of an integer must be 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 integer, 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 = "Javatpoint";
   //both single and double quote statements will treat different
   echo "Hello $company";
   echo "</br>";
   echo 'Hello $company';
?>
```

Output:

```
Hello Javatpoint
Hello $company
```

PHP Array

An array is a compound data type. It can store multiple values of 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
```

This is an advanced topic of PHP, which we will discuss later in detail.

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 defined a variable with no value.

Example:

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

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:

$num=10+20;//+ 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

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

PHP Operators Precedence

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 conditional statements

PHP conditional 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

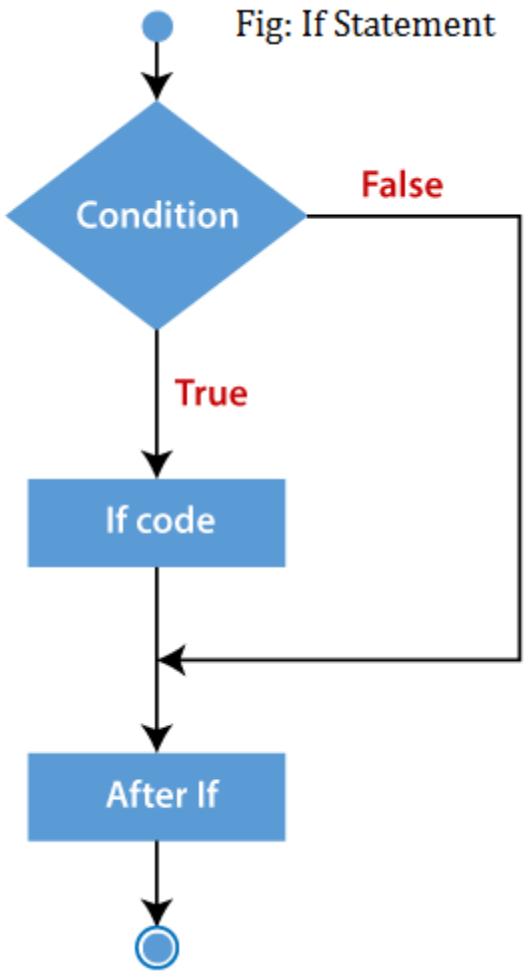
PHP if statement allows conditional execution of code. It is executed if condition is true.

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 Conditional Statements

PHP conditional statement is executed whether condition is true or false.

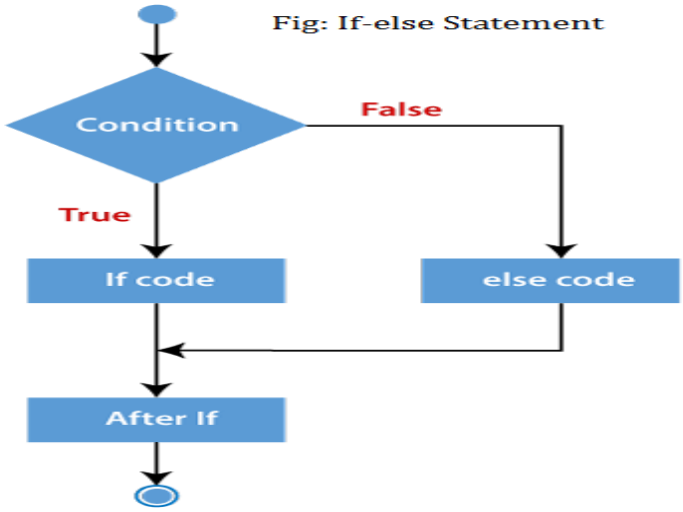
If-else statement is slightly different from if statement. It executes one block of code if the specified condition is **true** and another block of code if the condition is **false**.

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

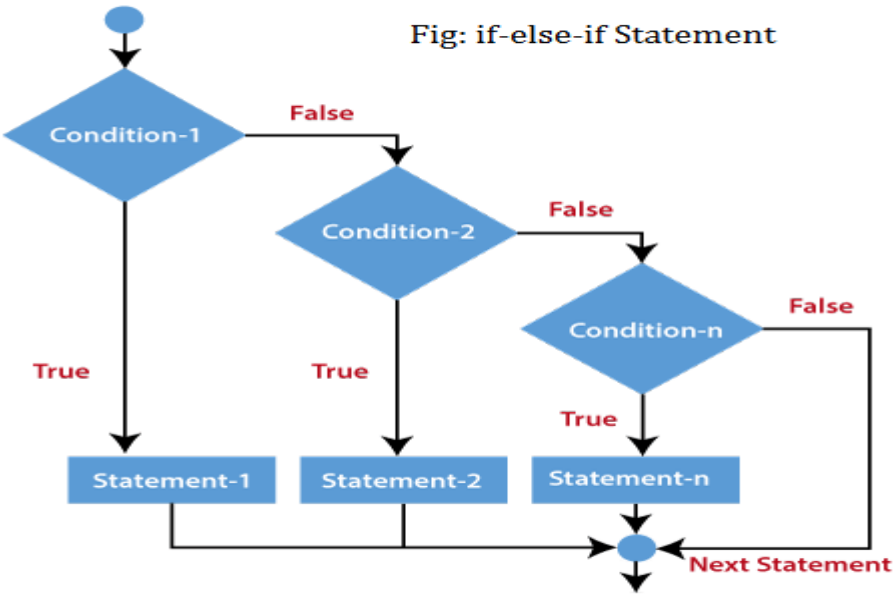


Fig: if-else-if Statement

Example

```
<?php
   $marks=69;
   if ($marks<33){
       echo "fail";
   }
   else if ($marks>=34 && $marks<50) {
       echo "D grade";
   }
   else if ($marks>=50 && $marks<65) {
      echo "C grade";
   }
   else if ($marks>=65 && $marks<80) {
       echo "B grade";
   }
   else if ($marks>=80 && $marks<90) {
```

```
    echo "A grade";
  }
  else if ($marks>=90 && $marks<100) {
    echo "A+ grade";
  }
 else {
    echo "Invalid input";
  }
?>
```

Output:

```
B Grade
```

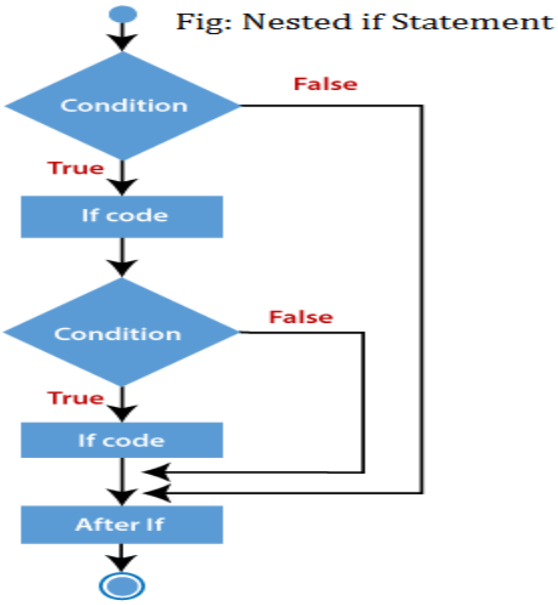
PHP nested if Statement

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

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

Flowchart



Example

```
<?php
        $age = 23;
   $nationality = "Indian";
   //applying conditions on nationality and age
   if ($nationality == "Indian")
   {
      if ($age >= 18) {
         echo "Eligible to give vote";
      }
      else {
         echo "Not eligible to give vote";
      }
   }
?>
```

Output:

```
Eligible to give vote
```

PHP Switch Example

```
<?php
        $a = 34; $b = 56; $c = 45;
   if ($a < $b) {
      if ($a < $c) {
         echo "$a is smaller than $b and $c";
      }
   }
?>
```

Output:

```
34 is smaller than 56 and 45
```

PHP Switch statement

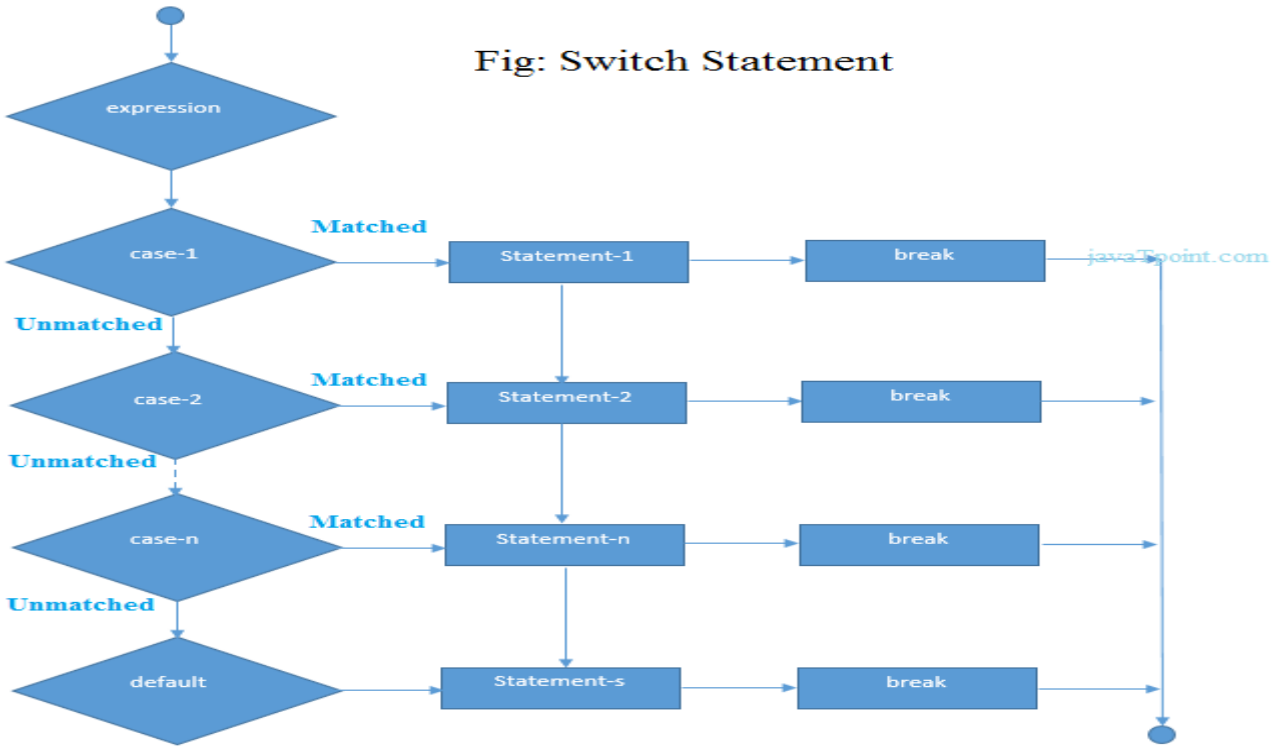
PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

```
switch(expression){
```

```
case value1:
 //code to be executed
 break;
case value2:
 //code to be executed
 break;
.....
default:
 code to be executed if all cases are not matched;
}
```

PHP Switch Flowchart



PHP Switch Example

```
<?php
$num=20;
switch($num){
case 10:
echo("number is equals to 10");
break;
case 20:
echo("number is equal to 20");
```

```
break;
case 30:
echo("number is equal to 30");
break;
default:
echo("number is not equal to 10, 20 or 30");
}
?>
```

Output:

```
number is equal to 20
```

PHP Looping statement

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;

- for loop
- while loop
- do-while loop
- foreach loop

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.

1. **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

// code to illustrate for loop
for ($num = 1; $num <= 10; $num += 2) {
    echo "$num \n";
}

?>
```

Output:

1

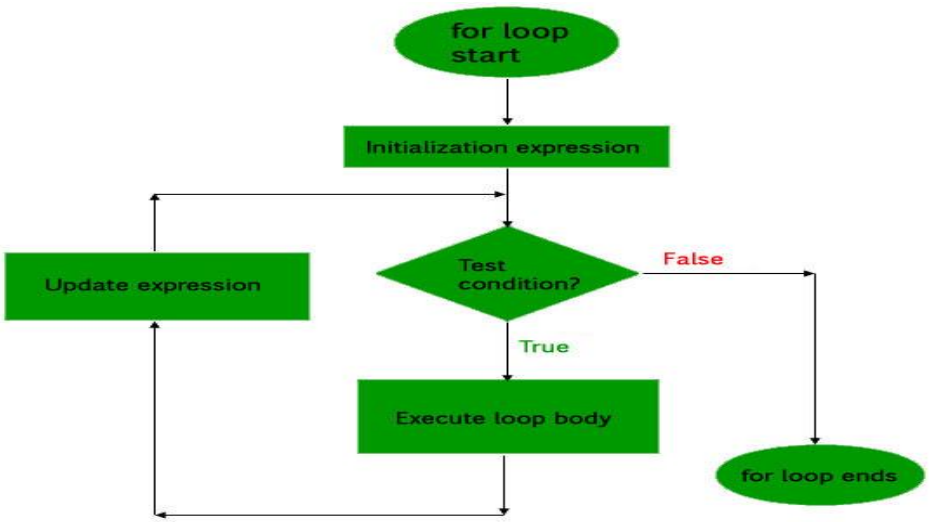
3

5

7

9

Flow Diagram:



5. **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
$num = 2;
while ($num < 12) {
   $num += 2;
   echo $num, "\n";
}
?>
```

   Output:

   4

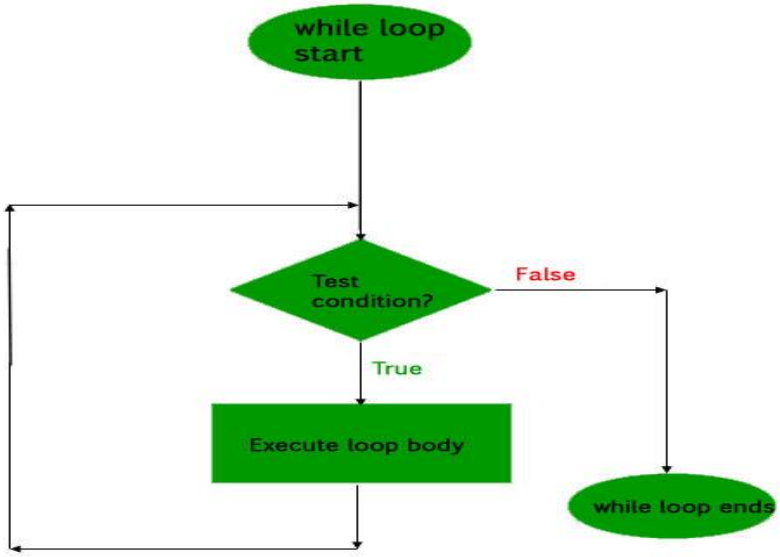
   6

   8

   10

   12

   Flowchart:



6. **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
$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.

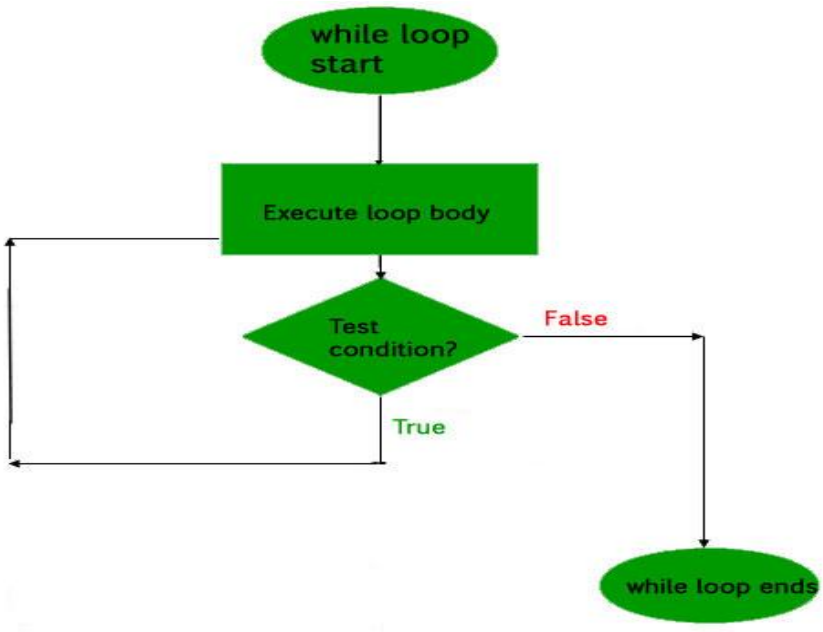
```
<?php

// 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:



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 Break

PHP break statement breaks the execution of the current for, while, do-while, switch, and for-each loop. If you use break inside inner loop, it breaks the execution of inner loop only.

The **break** keyword immediately ends the execution of the loop or switch structure. It breaks the current flow of the program at the specified condition and program control resumes at the next statements outside the loop.

The break statement can be used in all types of loops such as while, do-while, for, foreach loop, and also with switch case.

Syntax

```
jump statement;
break;
```

Flowchart

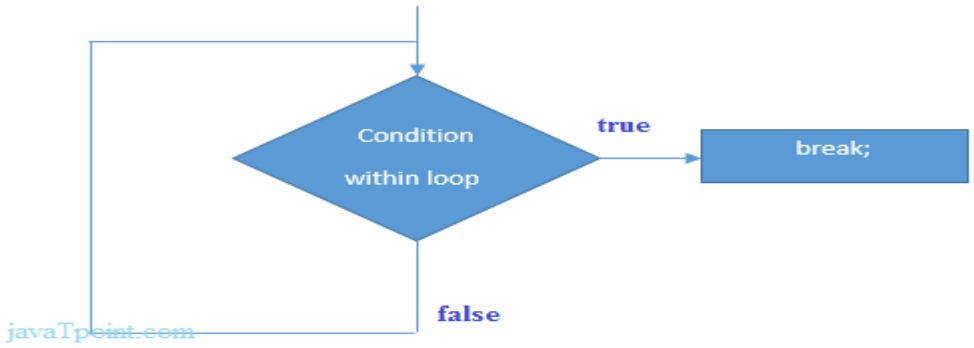


Figure: Flowchart of break statement

PHP Break: inside loop

Let's see a simple example to break the execution of for loop if value of i is equal to 5.

```
1. <?php
2. for($i=1;$i<=10;$i++){
3. echo "$i <br/>";
4. if($i==5){
5. break;
6. }
7. }
8. ?>
```

Output:

```
1
2
3
4
5
```

PHP continue statement

The PHP continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

The continue statement is used within looping and switch control structure when you immediately jump to the next iteration.

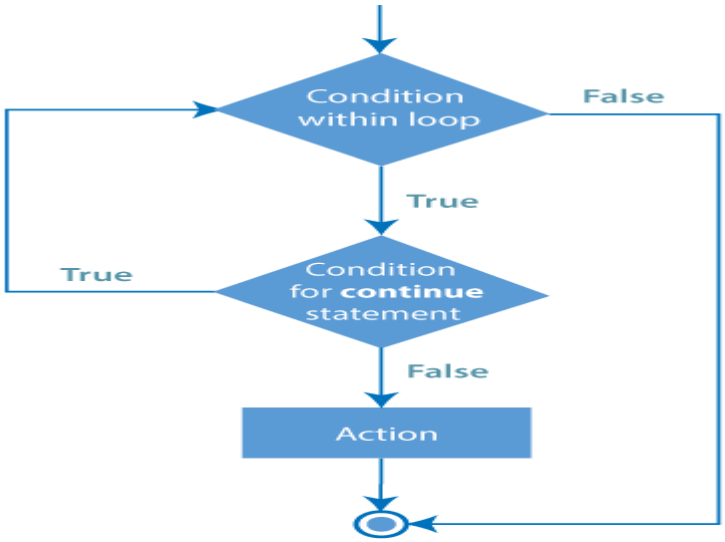
The continue statement can be used with all types of loops such as - for, while, do-while, and foreach loop. The continue statement allows the user to skip the execution of the code for the specified condition.

Syntax

The syntax for the continue statement is given below:

```
jump-statement;
continue;
```

Flowchart:



PHP Continue Example with for loop

Example

In the following example, we will print only those values of i and j that are same and skip others.

```
<?php
  //outer loop
  for ($i =1; $i<=3; $i++) {
    //inner loop
    for ($j=1; $j<=3; $j++) {
      if (!($i == $j) ) {
        continue;      //skip when i and j does not have same values
      }
      echo $i.$j;
      echo "</br>";
    }
```

```
    }
?>
```

Output:

```
11
22
33
```

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

```
function functionname(){
//code to be executed
}
```

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

```
<?php
function sayHello(){
echo "Hello PHP Function";
```

```
}
sayHello();//calling function
?>
```

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 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

File: functiondefaultarg.php

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

Output:

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