Strings in PHP

- Strings can be seen as a stream of characters. For example ‘J’ is a character and ‘Javascript’ is a string.
- Everything inside single quotes(‘’) and double quotes(“”) in PHP treated as a string.

<u>Creating Strings in PHP:</u>
There are four ways to create a string in PHP.
1. Single-quote strings: This type of string does not process special characters inside quotes.

```
<?php
$site = 'Welcome to PHP';

echo $site;

?>
```

Output:

```
Welcome to PHP
```

The above program compiles correctly. We have created a string ‘Welcome to GeeksforGeeks’ and stored it in variable and printing it using *echo* statement.
Let us now look at the below program:

```
<?php

$site = 'strings in
PHP';

echo 'Welcome to $site';
?>
```

Output:

```
Welcome to $site
```

In the above program the *echo* statement prints the variable name rather than printing the contents of the variables. This is because single-quotes strings in PHP do not process special characters. Hence, the string is unable to identify the ‘$’ sign as the start of a variable name.

2.Double-quote strings : Unlike single-quote strings, double-quote strings in PHP are capable of processing special characters.

```
<?php

echo "Welcome to javascript \n";

$site = "Javascript";

echo "Welcome to $site";
```

```
?>
```

Output:

```
Welcome to javascript
Welcome to javascript
```

In the above program, we can see that the double-quote strings are processing the special characters according to their properties. The '\n' character is not printed and is considered as a new line. Also instead of the variable name $site, "GeeksforGeeks" is printed.

PHP treats everything inside double quotes(" ") as Strings.

In this article, we will learn about the working of the various string functions and how to implement them along with some special properties of strings. Unlike other data types like integers, doubles, etc. Strings do not have any fixed limits or ranges. It can extend to any length as long as it is within the quotes.
It has been discussed earlier that string with single and double quotes are treated differently. Strings within a single quote ignore the special characters, but double-quoted strings recognize the special characters and treat them differently.

Example:

```
<?php

$name = "Krishna";

echo "The name of the geek is $name \n";

echo 'The name of the geek is $name';

?>
```

Output:

```
The name of the geek is Krishna
The name of the geek is $name
```

Some important and frequently used special characters that are used with double-quoted strings are explained below:

The character begins with a backslash("\") is treated as escape sequences and is replaced with special characters. Here are few important escape sequences.

1. "\n" is replaced by a new line
2. "\t" is replaced by a tab space
3. "\$" is replaced by a dollar sign
4. "\r" is replaced by a carriage return

5. “\\” is replaced by a backslash
6. “\”” is replaced by a double quote
7. “\'” is replaced by a single quote

- The string starting with a dollar sign(“$”) are treated as variables and are replaced with the content of the variables.

3. Heredoc: The syntax of Heredoc (<<<) is another way to delimit PHP strings. An identifier is given after the heredoc (<<<) operator, after which any text can be written as a new line is started. To close the syntax, the same identifier is given without any tab or space.

Note: Heredoc syntax is similar to the double-quoted string, without the quotes.

Example:

```
<?php
$input = <<<testHeredoc

Heredoc syntax is similar to the double-
quoted string, with out the quotes.

testHeredoc

echo $input;

?>
```

Output:

```
Heredoc syntax is similar to the double-
quoted string, without the quotes.
```

4. Nowdoc: Nowdoc is very much similar to the heredoc other than the parsing done in heredoc. The syntax is similar to the heredoc syntax with symbol <<< followed by an identifier enclosed in single-quote. The rule for nowdoc is the same as heredoc.

Note: Nowdoc syntax is similar to the single-quoted string.

Example:

```
<?php

$input = <<<'testNowdoc'

Nowdoc syntax is similar to the single-quoted
string.

testNowdoc;

echo $input;
echo <<<'Nowdoc'
\n
Welcome to PHP.
Learning PHP is fun.

Nowdoc;

?>
```

Output:

```
Nowdoc syntax is similar to the single-quoted string.\n
Welcome to PHP .
Learning PHP is fun.
```

Built-in String functions

Built-in functions in PHP are some existing library functions that can be used directly in our programs making an appropriate call to them. Below are some important built-in string functions that we use in our daily and regular programs:

1. strlen() function: This function is used to find the length of a string. This function accepts the string as an argument and returns the length or number of characters in the string.

Example:

```
<?php

echo strlen("Hello Programmers!");

?>
```

Output:

```
17
```

2. strrev() function: This function is used to reverse a string. This function accepts a string as an argument and returns its reversed string.

Example:

```
<?php

echo strrev("Hello Programmers!");

?>
```

Output:

```
!sremmargorp olleH
```

3. str_replace() function: This function takes three strings as arguments. The third argument is the original string and the first argument is replaced by the second one. In other words, we can say that it replaces all occurrences of the first argument in the original string with the second argument.

Example:

```
<?php

echo str_replace("Programmers", "World", "Hello Programmers!"),
"\n";
echo str_replace("function", "world", "Hello replace function!"),
"\n";

?>
```

Output:

```
Hello World!
Hello replace world!
```

In the first example, we can see that all occurrences of the word “Geeks” are replaced by “World” in “Hello GeeksforGeeks!”.

4. strpos() function: This function takes two string arguments and if the second string is present in the first one, it will return the starting position of the string otherwise returns FALSE.

Example:

```
<?php

echo strpos("Hello Programmers!", "Pro"), "\n";

echo strpos("Hello Programmers!", "grammers"), "\
n";

var_dump(strpos("Hello Programmers!", "Peek"));

?>
```

Output:

```
6
9
bool(false)
```

We can see in the above program, in the third example the string “Peek” is not present in the first string, hence this function returns a boolean value false indicating that string is not present.

5. trim() function: This function allows us to remove whitespaces or strings from both sides of a string.

Example:

```
<?php

echo trim("Hello World!", "Hed!");

?>
```

Output:

```
llo Worl
```

6. explode() function: This function converts a string into an array.

Example:

```
<?php


$input = "Welcome to programmers";

print_r(explode(" ",$input));

?>
```

Output:

```
Array ( [0] => Welcome [1] => to [2] => programmers )
```

7. strtolower() function: This function converts a string into the lowercase string.

Example:

```
<?php

$input = "WELCOME TO Programmers";

echo strtolower($input);

?>
```

Output:

```
welcome to programmers
```

8. strtoupper() function: This function converts a string into the uppercase string.

Example:

```
<?php

$input = "Welcome to Programmers";

echo strtoupper($input);

?>
```

Output:

```
WELCOME TO PROGRAMMERS
```

9. strwordcount() function: This function counts total words in a string.

Example:

```
<?php

$input = "Welcome to Programmers";

echo str_word_count($input);

?>
```

Output:

```
3
```

10. substr() function: This function gives the substring of a given string from a given index.

Example:

```
<?php

$input = "Welcome to programmers";

echo(substr($input,3));

?>
```

Output:

come to programmers

Conditional Statements in PHP

PHP allows us to perform actions based on some type of conditions that may be logical or comparative. Based on the result of these conditions i.e., either TRUE or FALSE, an action would be performed as asked by the user. It's just like a two- way path. If you want something then go this way or else turn that way. To use this feature, PHP provides us with four conditional statements:

- **if** statement
- **if…else** statement
- **if…elseif…else** statement
- **switch** statement

Let us now look at each one of these in details:

if Statement: This statement allows us to set a condition. On being TRUE, the following block of code enclosed within the if clause will be executed.

Syntax :

```
if (condition){
    // if TRUE then execute this code
}
```

```
Example:

<?php
$x = 12;

if ($x > 0) {
    echo "The number is positive";
}
?>
```

Output:

```
The number is positive
```

1. **if…else Statement**: We understood that if a condition will hold i.e., TRUE, then the block of code within if will be executed. But what if the condition is not TRUE and we want to perform an action? This is where else comes into play. If a condition is TRUE then if block gets executed, otherwise else block gets executed.

   Syntax:

   ```
   if (condition) {
       // if TRUE then execute this code
   }
   else{
       // if FALSE then execute this code
   }
   ```

Example:

```
<?php
$x = -12;

if ($x > 0) {
    echo "The number is positive";
}

else{
    echo "The number is negative";
}
?>
```

Output:

```
The number is negative
```

2. **if…elseif…else Statement**: This allows us to use multiple if…else statements. We use this when there are multiple conditions of TRUE cases.
   Syntax:

```
if (condition) {
    // if TRUE then execute this code
}
elseif {
    // if TRUE then execute this code
}
elseif {
    // if TRUE then execute this code
}
else {
    // if FALSE then execute this code
}
```

Example:

```
<?php
$x = "August";

if ($x == "January") {
    echo "Happy Republic Day";
}

elseif ($x == "August") {
    echo "Happy Independence Day!!!";
}

else{
    echo "Nothing to show";
}
?>
```

Output:

```
Happy Independence Day!!!
```

3. **switch Statement**: The “switch” performs in various cases i.e., it has various cases to which it matches the condition and appropriately executes a particular case block. It first evaluates an expression and then compares with the values of each case. If a case matches then the same case is executed. To use switch, we need to get familiar with two different keywords namely, **break** and **default**.
    1. The **break** statement is used to stop the automatic control flow into the next cases and exit from the switch case.
    2. The **default** statement contains the code that would execute if none of the cases match.

Syntax:

```
switch(n) {
    case statement1:
        code to be executed if n==statement1;
        break;
    case statement2:
        code to be executed if n==statement2;
        break;
    case statement3:
        code to be executed if n==statement3;
        break;
    case statement4:
        code to be executed if n==statement4;
        break;
    ......
    default:
        code to be executed if n != any case;
```

Example:

```
<?php
$n = "February";

switch($n) {
    case "January":
        echo "Its January";
        break;
    case "February":
        echo "Its February";
        break;
    case "March":
        echo "Its March";
        break;
    case "April":
        echo "Its April";
        break;
    case "May":
        echo "Its May";
        break;
    case "June":
        echo "Its June";
        break;
    case "July":
        echo "Its July";
        break;
    case "August":
        echo "Its August";
        break;
    case "September":
        echo "Its September";
        break;
    case "October":
        echo "Its October";
        break;
    case "November":
        echo "Its November";
        break;
    case "December":
        echo "Its December";
        break;
    default:
        echo "Doesn't exist";
}
?>
```

Output:

```
Its February
```

Ternary Operators in PHP:

In addition to all this conditional statements, PHP provides a shorthand way of writing if…else, called Ternary Operators. The statement uses a question mark (?) and a colon (:) and takes three operands: a condition to check, a result for TRUE and a result for FALSE.

Syntax:

```
(condition) ? if TRUE execute this : otherwise execute this;
```

Example:

```
<?php
$x = -12;

if ($x > 0) {
    echo "The number is positive \n";
}
else {
    echo "The number is negative \n";
}
```

```
// This whole lot can be written in a
// single line using ternary operator
echo ($x > 0) ? 'The number is positive' :
                'The number is negative';
?>
```

Output:

```
The number is negative
The number is negative
```

Control Statements in PHP

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;

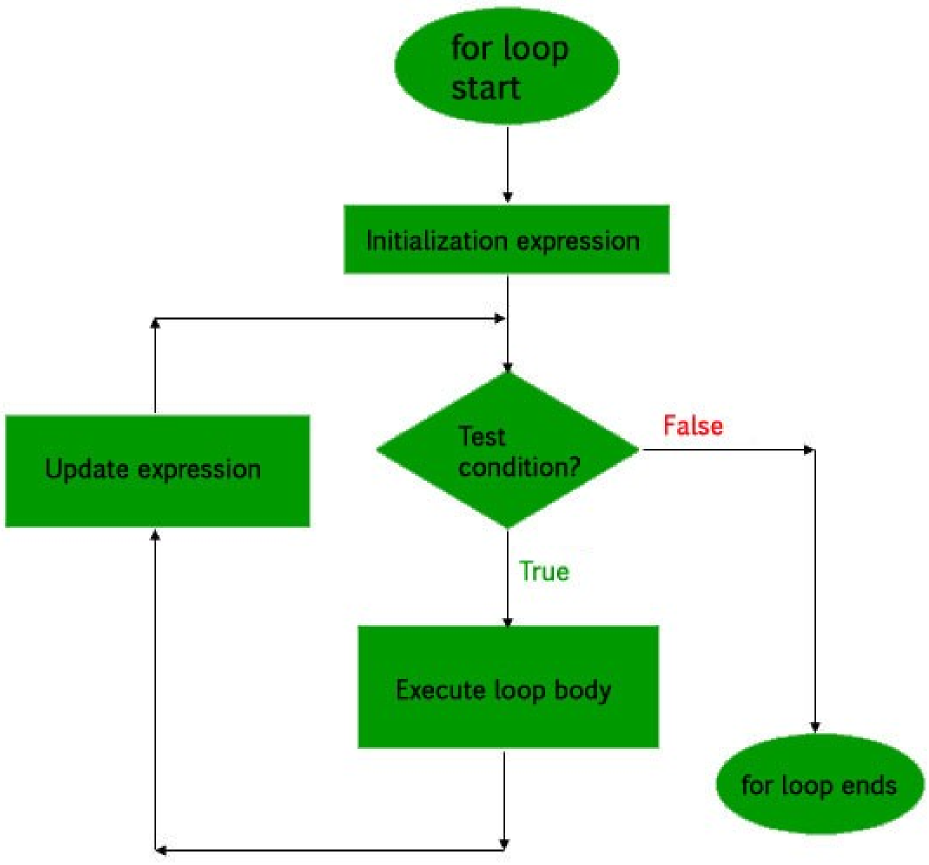
Example:

```
<?php

// 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
}
```

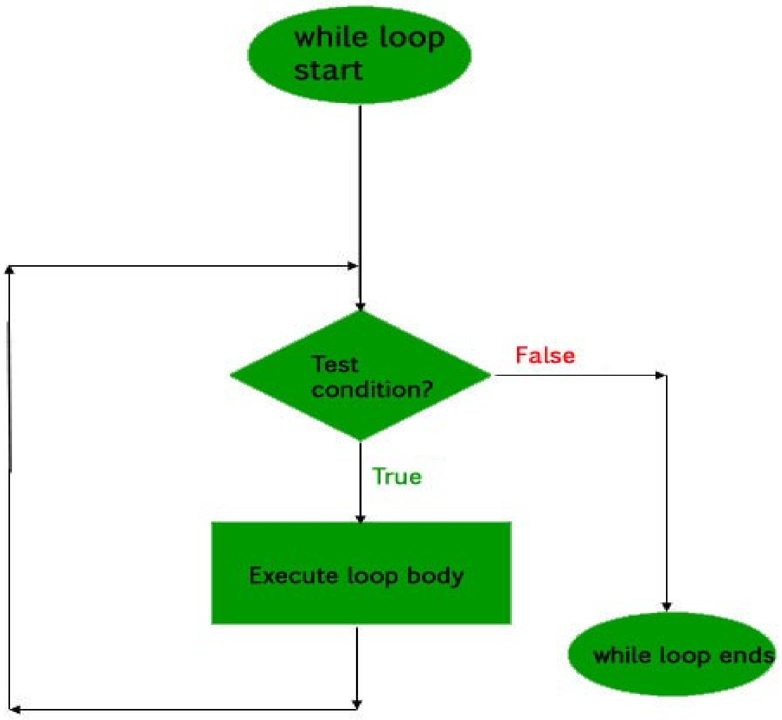
   Example:

```
<?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
```

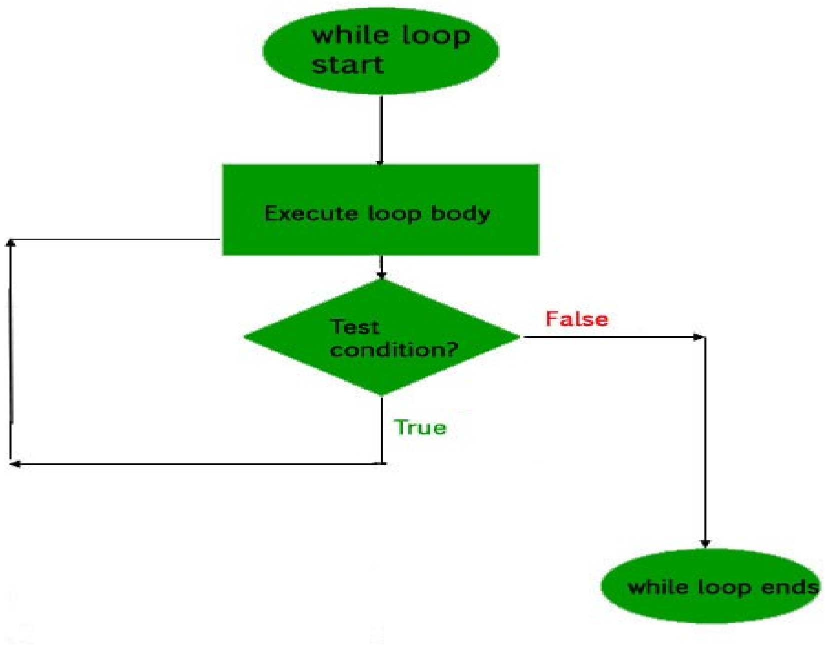
   This code would show the difference between while and do…while loop.

```
<?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 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;
```

Example with for loop:

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
```

Example with while loop:

In the following example, we will print the even numbers between 1 to 20.

```php
<?php
  //php program to demonstrate the use of continue statement

  echo "Even numbers between 1 to 20: </br>";
  $i = 1;
  while ($i<=20) {
    if ($i %2 == 1) {
      $i++;
      continue;   //here it will skip rest of statements
    }
    echo $i;
    echo "</br>";
    $i++;
  }
?>
```

Output:

```
Even numbers between 1 to 20:
2
4
6
8
10
12
14
16
18
20
```

Example with array

The following example prints the value of array elements except those for which the specified condition is true and continue statement is used.

```php
<?php
  $number = array ("One", "Two", "Three", "Stop", "Four");
  foreach ($number as $element) {
    if ($element == "Stop") {
      continue;
    }
```

```
        echo "$element </br>";
    }
?>
```

Output:

```
One
Two
Three
Four
```

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

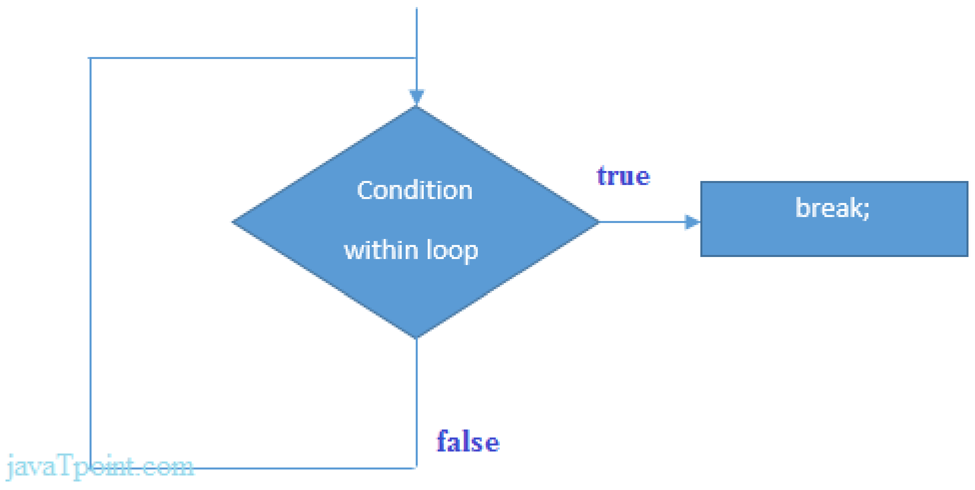


Figure: Flowchart of break statement

Example: Break inside loop

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

```
<?php
for($i=1;$i<=10;$i++){
echo "$i <br/>";
if($i==5){
break;
}
}
?>
```

Output:

```
1
2
3
4
5
```

Example: Break inside inner loop

The PHP break statement breaks the execution of inner loop only.

```
<?php
for($i=1;$i<=3;$i++){
 for($j=1;$j<=3;$j++){
  echo "$i   $j<br/>";
  if($i==2 && $j==2){
   break;
  }
 }
}
?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
3 1
3 2
3 3
```

Example: Break inside of Switch statement

The PHP break statement breaks the flow of switch case also.

```
<?php
$num=200;
switch($num){
```

```
case 100:
echo("number is equals to 100");
break;
case 200:
echo("number is equal to 200");
break;
case 50:
echo("number is equal to 300");
break;
default:
echo("number is not equal to 100, 200 or 500");
}
?>
```

Output:

```
number is equal to 200
```

Example Break with array of string :

```
<?php
//declare an array of string
$number = array ("One", "Two", "Three", "Stop", "Four");
foreach ($number as $element) {
if ($element == "Stop") {
break;
}
echo "$element </br>";
}
?>
```

Output:

```
One
Two
Three
```

You can see in the above output, after getting the specified condition true, break statement immediately ends the loop and control is came out from the loop.

Example switch statement without break:

It is not essential to break out of all cases of a switch statement. But if you want that only one case to be executed, you have to use break statement.

```
<?php

$car = 'Mercedes Benz';
switch ($car) {
```

```
default:
echo '$car is not Mercedes Benz<br>';
case 'Orange':
echo '$car is Mercedes Benz';
}
?>
```

Output:

```
$car is not Mercedes Benz
$car is Mercedes Benz
```