PHP
Structured Programming Concepts

Primary source: https://www.w3schools.com/php/

Internet Programming 2, Lesson 3

April 2024

PHP Conditional Statements
PHP Loops

PHP Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.
In PHP we have the following conditional statements:

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

PHP - The if Statement

The if statement executes some code if one condition is true.
Syntax

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

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

```php
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
}
?>
```

Output

```
Have a good day!
```

PHP - The if...else Statement

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

Syntax

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

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```php
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

Output - @ 05h04 at the location of the server

```
Have a good day!
```

PHP - The if...elseif....else Statement

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

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

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```php
<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

Output - @ 05h04 at the location of the server

```
The hour (of the server) is 05, and will give the following message:
Have a good morning!
```

The PHP switch Statement

The switch statement is used to perform different actions based on different conditions.
Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed.
Use **break** to prevent the code from running into the next case automatically.
The **default** statement is used if no match is found.

```php
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

Output

```
Your favorite color is red!
```

PHP Loops

PHP while loops execute a block of code while the specified condition is true.

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

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

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

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

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

```php
<?php
$x = 1;

while($x <= 5) {
  echo "The number is: $x <br>";
  $x++;
}
?>
```

Output

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

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

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

```php
<?php
$x = 1;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

Output

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.
The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

```php
<?php
$x = 6;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

Output

```
The number is: 6
```

The PHP for Loop

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

Syntax

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

Parameters:

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

The example below displays the numbers from 0 to 10:

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

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10
```

The PHP foreach Loop

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

Syntax

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

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

The following example demonstrates a loop that will output the values of the given array ($colors):

```
<?php
$colors
= array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
  echo "$value <br>";
}
?>
```

Output

```
red
green
blue
yellow
```

PHP Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

This example jumps out of the loop when **x** is equal to **4**:

Example

```
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
}
?>
```

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
```

NOTE: Although possible, above (exiting a for-look in this manner) must be used with care and should be avoided where possible…

PHP Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
This example skips the value of **4**:

Example

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

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
```

Break and Continue in While Loop

You can also use break and continue in while loops:

Break Example

```php
<?php
$x = 0;

while($x < 10) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
  $x++;
}
?>
```

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
```

Continue Example

```php
<?php
$x = 0;

while($x < 10) {
  if ($x == 4) {
    $x++;
    continue;
  }
  echo "The number is: $x <br>";
  $x++;
}
?>
```

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
```

Exercise 2:

EX2-1.php: Write a PHP Script to display:

- the first 10 odd numbers, starting from 1, using a WHILE-loop.
- the first 10 even numbers, starting from 2, using a DO…WHILE-loop.
- the squares of the first 10 odd numbers, starting from 1, using a WHILE-loop.
- the first 10 multiples of 3, starting from 3, using a FOR-loop.
- the first 10 Fibonacci sequence, starting from 1. The sequence is: 1,1,2,3,5,8,13,21,…
- the letters of the alphabet in the following format: rotate the case of each element in the list between uppercase and lowercase as follows: A, b, C, d, E, f, G…
- the first 20 prime numbers, starting at 1, including 1