Iterative WHILE and FOR/FOREACH Loops

by Sophia

WHAT'S COVERED

In this lesson, you will be introduced to the three common programming loop structures in PHP, which are used to repeat sections of code. You will learn about the generic `while`, `for`, and `foreach` loops used for processing collections of data.

Specifically, this lesson will cover the following:

1. Introduction to Loops

You may recall from a previous challenge on JavaScript that we can use loops (also called iterative structures) to repeat a set of commands until the loop's condition statement evaluates to “false.” Furthermore, you were briefly introduced to the `for` loop, which is a common type of loop in most programming and scripting languages. Overall, there are three common types of loops: the `while` loop, the `for` loop, and the `foreach` loop.

Loops use a condition statement that evaluates to a “true” or “false” response and determines if the loop should run or not. Once a loop starts, the body of the loop will execute (called an iteration), the condition will be retested, and the loop will either run again or terminate. The condition statements themselves often use comparison operators to compare the value of a variable against a threshold or target value. Furthermore, condition statements are often designed to create a counter-controlled or sentinel-value-controlled loop.

Counter-controlled loops use a counter variable, typically a whole number initialized to 0, and compare the counter value against a literal threshold. Additionally, developers also often use the “length” property of a collection object as the threshold. Each time the loop runs, the loop's final steps will be to increment the counter value, thus enabling the loop to eventually reach the threshold and end.

Sentinel-value-controlled loops use a condition statement that compares an input variable's value against a specific value (the sentinel value). The goal is to keep the loop running as long as the value of the input variable does NOT equal the sentinel value. This type of condition setup is perfect for when you want to give the user control over when the loop ends.

These two loop-controlling mechanisms are not mutually exclusive and can be combined using the proper logical operator, AND (&&) or OR (||).

Now, let us take a look at the three common types of programming loops.

TERMS TO KNOW

Counter-Controlled Loop
A loop that terminates after the loop has completed a predetermined number of iterations.

Sentinel-Value-Controlled Loop
A loop that terminates when the specific sentinel value is detected.

2. PHP While Loops

`while` loops are the simplest loop and include the keyword `while` followed by a set of parentheses, which holds the condition statement and then the body of code. As long as the condition statement evaluates to a “true” or nonzero output, the loop will continue to execute its body of code. When the executing system reaches a `while` loop, it will first evaluate the condition statement and determine if the loop needs to run at all. If the condition is false, the loop will be skipped altogether. If the condition is true, the loop will run once, and the condition will be retested before deciding to run the loop again or terminate it.

EXAMPLE Sample of a PHP `while` loop

```
<?php
        $threshold = 5;
        $count = 0;
        while ($count < $threshold)
        {
                echo "$count <br />";
                $count = $count + 1;
        }
?>
```

Output:

```
0
1
2
3
4
```

3. PHP For Loops

The `for` loops are designed with built-in control mechanisms. A `for` loop includes the keyword “for”, followed by parentheses containing three sections, and then the body of the code. The three sections within the parentheses are the initializer, condition statement, and modifier.

- The initializer is used to set up a counter variable. This variable only exists within the loop and for the duration of the loop.
- The condition statement determines whether the loop should continue or stop. It’s a test that is performed before each iteration of the loop. If the condition is true, the loop continues; if it’s false, the loop ends.
- The modifier is used to adjust the counter variable, typically by incrementing the current value of the counter by 1.

This structure allows for precise control over the execution of the loop.

EXAMPLE Sample of a PHP `for` loop

```
<?php
        $names = array("John", "Jane", "Able", "Carter");
        for ($count = 0 ; $count < count($names) ; $count = $count + 1)
        {
                echo "$count ";
                echo "$names[$count] <br />";
        }
?>
```

Output:

```
0 John
1 Jane
2 Able
3 Carter
```

KEY CONCEPT

Notice something different about the right side of the condition statement when compared to the JavaScript loop's condition? In PHP, arrays created using just the array() constructor function do not contain internal methods and attributes. Instead, in order to dynamically retrieve the number of elements in an array, we need to pass the array object into the function names count(). Calling count() and passing the array in as an argument will return an integer representing the size of the collection, which we can use in our condition statement.

HINT

Are you running into errors when trying to run your PHP code? Try using the PHP code analyzer web tool to check your code for errors. To use the PHP Code Checker, copy your code, visit phpcodechecker.com, paste your code into the textbox on the page, and then click "Analyze."

REFLECT

`for` loops may sound like they were designed specifically to be a counter-controlled loop and nothing else. However, in practice, `for` loops can be used as sentinel-value-controlled loops as well, as each of the three sections within the parentheses is optional.

4. PHP Foreach Loops

`foreach` loops are specialized loops designed specifically for processing collection objects, so they will only work on arrays and other collection-type objects. The idea behind `foreach` loops is that when they are used to process collections, each element of the collection is extracted and assigned to a temporary variable. The temporary variable is then used in the body of the loop to process each element, one at a time.

EXAMPLE Sample of a PHP `foreach` loop

```
<?php
```

```
        $directory = array("John", "Jane", "Able", "Carter");
        $count = 0;
        foreach($directory as $name)
        {
          echo "$count ";
          echo "$name <br />";
          $count++;
        }
?>
```

Output:
0 John
1 Jane
2 Able
3 Carter

You may recall that in a previous tutorial, we used a JavaScript loop to generate the rows and columns of an HTML table. The same can be accomplished using a PHP loop. Start by echoing the table tag and the first row of headings; then using a PHP loop, interpolate data into a string containing the rows, cells, and data for the table and echo the string. Just below or after the loop, echo another string containing the table's closing tags.

EXAMPLE This is a PHP loop to create a table:

```
<style>
        table.border, th, td { border: 1px solid black;
                               border-collapse: collapse;
                               padding: 2px; }
        td:nth-child(2) { text-align: center }
</style>

<?php
        $fruit = array( "Banana", 0.5 , "Apple", 2.0, "Orange", 1.49, "Jackfruit", 25.89 );

        function createTable($data)
        {
           echo "<table class='border'><tr><th>Product</th><th>Unit Price</th></tr>";
           for ($counter = 0; $counter < count($data) ; $counter = $counter + 2 )
          {
           echo "<tr><td>{$data[$counter]}</td><td>$".number_format($data[$counter+1], 2)."</td></tr>";
          }
          echo "</table>";
        }
        createTable($fruit);
?>
```

Output:

Product	Unit Price
Banana	$0.50
Apple	$2.00
Orange	$1.49
Jackfruit	$25.89

THINK ABOUT IT

Why did we increment the counter variable by 2 in the above example? And why did we use `[$counter+1]` in the second cell?

Since we were accessing two elements during each iteration, if we only incremented by 1 in the next row, we would get the price from the first row as the product name in the second row. As such, we needed the counter to skip over the next element, so we increment by 2.

Furthermore, for each row, we need to access the current counter element value, as well as the next element. Since we only have the counter variable 1, we can use it to access the current counter element value and manually add 1 to access the next. When the loop ends, the counter variable is incremented by 2, thus preparing the counter variable to access the next two values.

Finally, for the second cell, where we add 1 to the value of $count, we need to explicitly interpolate the value by breaking the string and using the dot operator. This is due to the fact that we cannot perform math operations when interpolating data within a string. Instead, we need to break out of the string first in order to add 1 to $count when accessing the collection.

WATCH

View the following video for more on learning PHP.

SUMMARY

In this lesson, you learned about the `while`, `for`, and `foreach` loops and how each type of loop is created and controlled. You learned about the counter-control mechanism versus the sentinel-value control mechanism `for` loops and how they can be used to control when and how a loop terminates. Lastly, you saw how loops can be used in PHP scripts to generate structures of HTML content, specifically a table of information.

TERMS TO KNOW

Counter-Controlled Loop
A loop that terminates after the loop has completed a predetermined number of iterations.

Sentinel-Value-Controlled Loop

A loop that terminates when the specific sentinel value is detected.