200 XP

Exercise - Create and configure for iteration loops

14 minutes

On the surface, the `for` statement is another iteration statement that allows you to iterate through a code block and thereby change the flow of execution of your code. However, once we examine how each works, we can better identify the nuances of each iteration statement and when to use them.

What is the `for` statement?

The `for` statement iterates through a code block a specific number of times. This level of control makes the `for` statement unique among the other iteration statements. The `foreach` statement iterates through a block of code once for each item in a sequence of data like an array or collection. The `while` statement iterates through a block of code until a condition is met.

Furthermore, the `for` statement gives you much more control over the process of iteration by exposing the conditions for iteration.

In this exercise, you'll use the `for` statement, learning how to control the iteration's pre-condition, completion condition, its iteration pattern and more. Also, you'll learn of common use cases for the `for` statement.

Okay, now let's prepare our coding environment and begin our examination of code samples that implement a `for` statement.

Prepare your coding environment

This module includes hands-on activities that guide you through the process of building and running demonstration code. We encourage you to complete these activities using Visual Studio Code as your development environment. Using Visual Studio Code for these activities will help you to become more comfortable writing and running code in a developer environment that's used by professionals worldwide.

1. Open Visual Studio Code.

   You can use the Windows Start menu (or equivalent resource for another OS) to open Visual Studio Code.

2. On the Visual Studio Code **File** menu, select **Open Folder**.

3. In the **Open Folder** dialog, navigate to the Windows Desktop folder.

   If you have different folder location where you keep code projects, you can use that folder location instead. For this training, the important thing is to have a location that's easy locate and remember.

4. In the **Open Folder** dialog, select **Select Folder**.

   If you see a security dialog asking if you trust the authors, select **Yes**.

5. On the Visual Studio Code **Terminal** menu, select **New Terminal**.

   Notice that a command prompt in the Terminal panel displays the folder path for the current folder. For example:

   dos

   ```dos
   C:\Users\someuser\Desktop>
   ```

   ⓘ **Note**

   If you are working on your own PC rather than in a sandbox or hosted environment and you have completed other Microsoft Learn modules in this C# series, you may have already created a project folder for code samples. If that's the case, you can skip over the next step, which is used to create a console app in the TestProject folder.

6. At the Terminal command prompt, to create a new console application in a specified folder, type **dotnet new console -o ./CsharpProjects/TestProject** and then press Enter.

   This .NET CLI command uses a .NET program template to create a new C# console application project in the specified folder location. The command creates the CsharpProjects and TestProject folders for us, and uses TestProject as the name of our `.csproj` file.

7. In the EXPLORER panel, expand the **CsharpProjects** folder.

You should see the TestProject folder and two files, a C# program file named Program.cs and a C# project file named TestProject.csproj.

8. In the EXPLORER panel, to view your code file in the Editor panel, select **Program.cs**.

9. Delete the existing code lines.

   You'll be using this C# console project to create, build, and run code samples during this module.

10. Close the Terminal panel.

Write a basic for statement

1. Ensure that you have Visual Studio Code open and Program.cs displayed in the Editor panel.

   Note

   Program.cs should be empty. If if isn't, select and delete all code lines.

2. Type the following code into the Visual Studio Code Editor.

   C#

   ```
   for (int i = 0; i < 10; i++)
   {
       Console.WriteLine(i);
   }
   ```

   This code presents a simple `for` statement that loops through its code block 10 times, printing the current value of `i`.

3. On the Visual Studio Code **File** menu, select **Save**.

   The Program.cs file must be saved before building or running the code.

4. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**.

   A Terminal panel will open. The Terminal should include a command prompt showing that the Terminal is open to your TestProject folder location.

5. At the Terminal command prompt, to run your code, type **dotnet run** and then press Enter.

   > **Note**
   >
   > If you see a message saying "Couldn't find a project to run", ensure that the Terminal command prompt displays the expected TestProject folder location. For example: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

   You should see the following output.

   Output

   ```
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   ```

6. Take a minute to identify the six parts of the `for` statement.

   The `for` statement includes the following six parts:

   a. The `for` keyword.
   b. A set of parenthesis that defines the conditions of `for` iteration. The parentheses contain three distinct parts, separated by the end of statement operator, a semi-colon.
   c. The first part defines and initializes the iterator variable. In this example: `int i = 0`. This section is referred to as the **initializer**.
   d. The second part defines the completion condition. In this example: `i < 10`. In other words, the runtime will continue to iterate over the code in the code block below the `for` statement while `i` is less than `10`. When `i` becomes equal to `10`, the runtime stops executing the `for` statement's code block. The docs refer to this section as the **condition**.
   e. The third part defines the action to take after each iteration. In this case, after each iteration, `i++` will increment the value of `i` by 1. The docs refer to this section as the **iterator**.
   f. Finally, the code block. The code block contains the code that will be executed for each iteration. Notice that the value of `i` is referenced inside of the code block. The docs refer

to this section as the **body**.

Given our rules for naming variables, you may wonder if `i` is a valid name for the variable that holds the current iteration. In this case, `i` is considered by most to be valid. Other popular choices are `x` and `counter`. The name `j` is also used in those situations when you have an outer `for` statement that uses `i`, and need to create an iteration variable for an inner `for` statement.

ⓘ **Note**

All three sections (initializer, condition, and iterator) are optional. However, in practice, typically all three sections are used.

Change the iteration conditions

As we stated at the outset, the `for` statement has two unique qualities among iteration statements.

- The `for` statement should be used when you know the number of times you need to iterate through a block of code ahead of time.
- The `for` statement allows you to control the way in which each iteration is handled.

What if we needed to iterate through a block of code, but want to count down instead of counting up?

1. Use the Visual Studio Code Editor to update your code as follows:

c#

```c#
for (int i = 10; i >= 0; i--)
{
    Console.WriteLine(i);
}
```

2. Take a minute to review your updated code.

   By changing the three parts of the `for` statement, we change its behavior.

   - We initialize the iteration variable to 10.
   - We change the completion condition to exit the `for` statement when `i` is less than `0`.

- We change the pattern of the iterator to subtract `1` from `i` each time we complete an iteration.

3. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

4. Notice that the output has changed.

   When you run the code, you'll see the following output.

   Output

   ```
   10
   9
   8
   7
   6
   5
   4
   3
   2
   1
   0
   ```

Experiment with the iterator's pattern

What if we wanted to skip past certain values in the iterator variable?

1. Use the Visual Studio Code Editor to update your code as follows:

   C#

   ```c#
   for (int i = 0; i < 10; i += 3)
   {
       Console.WriteLine(i);
   }
   ```

2. Take a minute to review your updated code.

   Instead of incrementing or decrementing the value of the iterator variable by `1`, we use `i += 3` to skip two values after each iteration.

3. Save your code file, and then use Visual Studio Code to run your code.

Enter `dotnet run` from the Terminal command prompt to run your code.

4. Notice how the output has changed.

   When you run the code, you'll see the following output.

   Output

   ```
   0
   3
   6
   9
   ```

   Admittedly, you won't do this sort of thing often, but hopefully you can appreciate that you have a fine grained level of control over the iterations should you ever need it.

Use the break keyword to break the iteration statement

What if we need to exit the iteration statement prematurely based on some condition? We can use the `break` keyword.

1. Use the Visual Studio Code Editor to update your code as follows:

   C#

   ```csharp
   for (int i = 0; i < 10; i++)
   {
       Console.WriteLine(i);
       if (i == 7) break;
   }
   ```

2. Take a minute to review the use of the `break` keyword in your updated code.

   We first saw the `break` keyword in the module "Branch the flow of code using the switch-case construct in C#". As it turns out, we can use the `break` keyword to exit out of iteration statements as well.

3. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

4. Notice how the output has changed.

When you run the code, you'll see the following output.

Output

```
0
1
2
3
4
5
6
7
```

Loop through each element of an array

A common usage for the `for` statement is to iterate through an array of elements, especially if you need some control over the manner in which the iteration happens. While the `foreach` iterates through every element of the array, the `for` statement can be tweaked to provide more customization.

1. Use the Visual Studio Code Editor to update your code as follows:

   C#

   ```
   string[] names = { "Alex", "Eddie", "David", "Michael" };
   for (int i = names.Length - 1; i >= 0; i--)
   {
       Console.WriteLine(names[i]);
   }
   ```

2. Take a minute to review your updated code.

   First off, notice that we have instantiated a string array named `names` that contains four names.

   Next, notice that we are using the `Array.Length` property to get the number of elements in the array, and that we are using this value to initialize our iterator variable (`int i = names.Length - 1`). We subtract 1 from the value because the index number for array elements is zero-based (the index numbers of the four elements are 0-3).

   Finally, notice we have chosen iterate through the array backwards--something that we are unable to do with the `foreach` statement. We use the value of the iteration variable inside the code block to specify the index number of the array elements (`names[i]`).

3. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

4. Notice that the array elements are listed in reverse order (as we intended).

   When you run the code, you'll see the following output.

   Output

   ```
   Michael
   David
   Eddie
   Alex
   ```

   Note

   We could have iterated forward through the array elements by constructing the `for` statement as follows: `for (int i = 0; i < names.Length; i++)`.

Examine the limitation of the foreach statement

What if you want to update a value in the array during a `foreach` iteration?

1. Use the Visual Studio Code Editor to update your code as follows:

   C#

   ```c#
   string[] names = { "Alex", "Eddie", "David", "Michael" };
   foreach (var name in names)
   {
       // Can't do this:
       if (name == "David") name = "Sammy";
   }
   ```

2. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

3. Notice the error message that is displayed.

If you attempt to compile and run this code, you will see an exception.

Output

```
Cannot assign to name because it is a 'foreach iteration variable'
```

In other words, you can't reassign the value of `name` because it is part of the `foreach` iteration's inner implementation.

Overcoming the limitation of the foreach statement using the for statement

Let's try using a `for` statement to modify the contents of an array inside the iteration code block.

1. Use the Visual Studio Code Editor to update your code as follows:

   c#

   ```c#
   string[] names = { "Alex", "Eddie", "David", "Michael" };
   for (int i = 0; i < names.Length; i++)
       if (names[i] == "David") names[i] = "Sammy";

   foreach (var name in names) Console.WriteLine(name);
   ```

2. Take a minute to review your updated code.

   Notice that we removed the curly braces from the code blocks that contained only a single line of code. This revision uses the same technique that we talked about in the module "Control variable scope and logic using code blocks in C#". Many developers find this style difficult to read, while others prefer this abbreviated style because it helps them write more succinctly and more expressively. If you find this code difficult to read, or if you just don't prefer this style, rest assured that the curly braces can always be used in your code blocks. If you want, update the code in the Editor panel with the following code:

   c#

   ```c#
   string[] names = { "Alex", "Eddie", "David", "Michael" };

   for (int i = 0; i < names.Length; i++)
   {
       if (names[i] == "David") names[i] = "Sammy";
   }
   ```

```
foreach (var name in names)
{
    Console.WriteLine(name);
}
```

3. Save your code file, and then use Visual Studio Code to run your code.

   Enter `dotnet run` from the Terminal command prompt to run your code.

4. Notice that the code runs without error and generates the desired output.

   When you run the code, you'll see the following output.

   Output

   ```
   Alex
   Eddie
   Sammy
   Michael
   ```

   Since the array isn't directly part of the iteration statement's implementation, you can change values inside of the array.

Recap

Here are a few of the takeaways from this unit:

- The `for` iteration statement allows you to iterate through a block of code a specific number of times.
- The `for` iteration statement allows you to control every aspect of the iteration's mechanics by altering the three conditions inside the parenthesis: the initializer, condition, and iterator.
- It's common to use the `for` statement when you need to control how you want to iterate through each item in an array.
- If your code block has only one line of code, you can eliminate the curly braces and white space if you wish.

Knowledge check

1. Which of the following `for` statements is correct? *

- ○ `for (int x = 20: x < 80: x++)`
- ○ `for (int j = 0; j < 80; j + 1)`
- ○ `for (int counter = 20; counter < 80; counter++)`

2. Which of the following statements should be used to exit out of a `for` loop before the iteration has completed? *

- ○ `exit;`
- ○ `break;`
- ○ `return;`

Check your answers