200 XP

Exercise - Create do and while iteration loops

10 minutes

On the surface, the `do-while` and `while` statements are yet *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 do-while statement?

The `do-while` statement executes a statement or a block of statements while a specified Boolean expression evaluates to true. Because that expression is evaluated after each execution of the loop, a do-while loop executes one or more times.

```
c#

do
{
    // This code executes at least one time
} while (true);
```

The flow of execution starts inside of the curly brace. The code executes at least one time, then the Boolean expression next to the `while` keyword is evaluated. If the Boolean expression returns `true`, the code block is executed again.

By hard coding the Boolean expression to `true`, we've created an infinite loop--a loop that will never end, at least, not as it's currently written. We would need a way to break out of the loop inside of the code block. We'll discuss the exit criteria of a `do-while` in a bit.

Okay, now let's prepare our coding environment and begin our examination of code samples that implement a `do-while` 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
   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 do-while statement to break when a certain random number is generated

Let's write code that will keep generating random numbers between 1 and 10 until we generate the number 7. It could take just one iteration to get a 7, or it could take dozens of iterations.

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#
Random random = new Random();
int current = 0;

do
{
    current = random.Next(1, 11);
    Console.WriteLine(current);
} while (current != 7);
```

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>`

6. Review your output.

   Since the numbers generated are random, your results will be different. However, the value `7` will be the last number to be printed as the Boolean expression will evaluate to `false` when 7 is generated and the flow of execution will exit the code block.

   Output

   ```
   2
   5
   8
   2
   7
   ```

7. Take a minute to review your code.

   A key learning for this first task is that the code block of a `do-while` loop will execute at least once. It could iterate a large number of times, and it is unlikely that we know ahead of time how many iterations there will be.

   It's also important to notice that the code inside of the code block is influencing whether to continue iterating through the code block or not. A code block that influences the exit criteria is a primary reason to select a `do-while` or `while` statements rather than one of the

other iteration statements. Both the `foreach` and `for` rely on factors that are external to the code block to determine the number of code block iterations.

Write a while statement that iterates only while a random number is greater than some value

Now let's take a look at the `while` statement.

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

```c#
Random random = new Random();
int current = random.Next(1, 11);

/*
do
{
    current = random.Next(1, 11);
    Console.WriteLine(current);
} while (current != 7);
*/

while (current >= 3)
{
    Console.WriteLine(current);
    current = random.Next(1, 11);
}
Console.WriteLine($"Last number: {current}");
```

> **Note**
>
> In this case, we position the `while` keyword and the Boolean expression before the code block. This changes the meaning of the code and acts as a "gate" to only allow the flow of execution to enter if the Boolean expression evaluates to true.

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. Review the output values listed.

   Since the numbers are random, so your code will produce a different sequence.

Output

```
9
7
5
Last number: 1
```

4. Take a minute to review your code.

   At the top code, we use `random` to initialize our `int` variable named `current`. Our next active code line is our `while` statement.

   Our `while` statement will iterate based on the Boolean expression `(current >= 3)`. We don't know what value will be assigned to `current`, but there are possibilities that affect our `while` loop:

   - If `current` is initialized to a value greater than or equal to `3`, then the Boolean expression will return `true`, and the flow of execution will enter the code block. Inside the code block, the first thing that we do is write the value of `current` to the console. Next, still inside the code block, we update the value of `current` with a new random value. At this point, control goes back to the top of the `while` statement where the Boolean expression is evaluated. This process continues until the Boolean expression returns `false` and the flow of execution breaks from the code block.
   - If `current` is initialized (at the top of our code) to a value less than `3`, then the Boolean expression will return `false`, and the code block will never execute.

   The final code line writes the value of `current` to the console. This code runs whether the iteration code block was executed or not, and is our chance to write the final value of `current` to the console.

Use the continue statement to step directly to the Boolean expression

In certain cases, we want to short-circuit the remainder of the code in the code block and continue to the next iteration. We can do that using the `continue` statement.

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

```
c#
```

```
Random random = new Random();
int current = random.Next(1, 11);

do
{
    current = random.Next(1, 11);

    if (current >= 8) continue;

    Console.WriteLine(current);
} while (current != 7);

/*
while (current >= 3)
{
    Console.WriteLine(current);
    current = random.Next(1, 11);
}
Console.WriteLine($"Last number: {current}");
*/
```

2. Take a minute to review your code.

   Notice that we've switched back to a `do-while`. A `do-while` ensures that the loop will iterate at least once.

   The first thing that we do inside the code block is to assign a new random value to `current`. Next, we check to see if `current` is greater than or equal to `8`. If this expression returns `true`, the `continue` key word will transfer control to the end of the code block and the `while` will evaluate `(current != 7)`. So, the loop will continue to iterate as long as the value of `current` is not equal to `7`.

   The key to this step of the exercise is the line of code containing the `continue` key word:

   c#

   ```
   if (current >= 8) continue;
   ```

   Since our code that writes the value of `current` to the console is located after the `if (current >= 8) continue;`, our code ensures that a value of `current` that is greater than or equal to `8` will never be written to the output window.

   Let's try it out.

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. Review the output values listed.

   Output

   ```
   5
   1
   6
   7
   ```

   You'll likely see different results than what is displayed below. However, you will not see any values `8` or greater in the output window before the code's execution ends with the value `7`.

5. Consider the difference between the `continue` and `break` statements.

   As you saw in this last step, the `continue` statement transfers execution to the end of the current iteration. This behavior is different than the behavior we saw with the `break` statement. The `break` statement terminates the iteration (or `switch`) and transfers control to the statement that follows the terminated statement. If there is no statement after the terminated statement, then control transfers to the end of the file or method.

Recap

There's a few important ideas you should take away from this unit:

- The `do-while` statement iterates through a code block at least once, and might continue to iterate based on a Boolean expression. The evaluation of the Boolean expression usually depends on some value generated or retrieved inside of the code block.
- The `while` statement evaluates a Boolean expression first, and continues to iterate through the code block as long as the Boolean expression evaluates to `true`.
- The `continue` keyword to step immediately to the Boolean expression.

Knowledge check

1. Which of the following correctly describes a behavior of either a `do-while` or `while` iterative statement? *

- ◯ A `do-while` statement executes a code block zero or more times.
- ◯ A `while` statement executes a code block at least once.
- ◯ A `do-while` statement executes a code block at least once.

2. A developer needs to capture user input inside a loop. The user enters the keyboard combination `ctrl + Esc` to exit the iteration when they're done entering information. Which is the best iteration statement for this purpose? *

- ◯ while
- ◯ for
- ◯ do-while

Check your answers