100 XP

Exercise - Create arrays and foreach loops

25 minutes

In this exercise, you'll review the Starter project code and then begin updating the application. Your first coding task will be creating the arrays that hold student exam scores. Once your application data is available in arrays, you'll begin working on a `foreach` loop that can be used to sum student grades. The detailed tasks that you'll complete during this exercise are:

1. Code review: review the contents of the Program.cs file.
2. Arrays: Create the arrays that store each student's assignment scores.
3. Iteration: Create a `foreach` loop that can be used to sum Sophia's assignment grades.
4. Calculate and display Sophia's average assignment grade.
5. Verification test: perform a verification test for the code that you've developed in this exercise.

> **Important**
>
> You need to have completed the Setup instructions in the previous unit, Prepare, before you begin this Exercise.

Review the contents of the Program.cs file

In this task, you'll review the code that's provided as a Starter project for this module. The Program.cs file contains the initial version of the student grading application that you'll be updating.

1. Ensure that you have the **GuidedProject** folder open in Visual Studio Code.

   The Prepare unit (the previous unit in this module) includes a Setup section for this Guided project module. The Setup section describes the process for downloading your initial code and opening the project folder Visual Studio Code. If necessary, go back and follow the Setup instructions.

2. In the Visual Studio Code EXPLORER panel, expand the **Starter** folder, then select **Program.cs**.

   When you select the Program.cs file, the file contents will open in the main Editor area to the right of the EXPLORER panel.

   If the EXPLORER panel isn't open, you can open it using the Activity Bar on the far-left side of the Visual Studio Code window. The EXPLORER button is the topmost icon on the Activity Bar.

3. Take a few minutes to review the code in the Program.cs file.

4. Notice that the top portion of your code begins with a `Using` statement and a list of variable declarations.

```c#
// initialize variables - graded assignments
int currentAssignments = 5;

int sophia1 = 90;
int sophia2 = 86;
int sophia3 = 87;
int sophia4 = 98;
int sophia5 = 100;

int andrew1 = 92;
int andrew2 = 89;
int andrew3 = 81;
int andrew4 = 96;
int andrew5 = 90;

int emma1 = 90;
int emma2 = 85;
int emma3 = 87;
int emma4 = 98;
int emma5 = 68;

int logan1 = 90;
int logan2 = 95;
int logan3 = 87;
int logan4 = 88;
int logan5 = 96;
```

   The `using` statement enables you to write code that implements members of the `System` namespace without requiring you to specify `System`. For example, your code can use the

`Console.WriteLine()` method without having to specify `System.Console.WriteLine()`. Among other things, the `using` statement makes your code easier to read.

Next, you see a comment line followed by a list of variables that are used to hold the scores of graded assignments for each student. Your first variable, `currentAssignments` is used to hold the number of exams that have been scored.

The assignment score variables represent a great opportunity to create and use arrays!

5. Scroll down and review the two groups of variable declaration code lines.

```c#
int sophiaSum = 0;
int andrewSum = 0;
int emmaSum = 0;
int loganSum = 0;

decimal sophiaScore;
decimal andrewScore;
decimal emmaScore;
decimal loganScore;
```

The first group of variables are integers that are being used to hold the sum of the exam scores.

The second group of variables are decimals that are used to hold the calculated average score. The code uses decimals here because an integer calculation rounds off the fractional portion of the calculated value.

Notice that you were using a unique variable for each student. This may provide another opportunity to slim down the number of code lines in your updated application.

It looks like the starter code begins score calculations next.

6. Scroll down a bit further and take a minute to review the following code:

```c#
sophiaSum = sophia1 + sophia2 + sophia3 + sophia4 + sophia5;
andrewSum = andrew1 + andrew2 + andrew3 + andrew4 + andrew5;
emmaSum = emma1 + emma2 + emma3 + emma4 + emma5;
loganSum = logan1 + logan2 + logan3 + logan4 + logan5;

sophiaScore = (decimal)sophiaSum / currentAssignments;
```

```
andrewScore = (decimal)andrewSum / currentAssignments;
emmaScore = (decimal)emmaSum / currentAssignments;
loganScore = (decimal)loganSum / currentAssignments;
```

The first group of equations is used to calculate the sum of the assignment scores for each student.

The second group of equations calculates the average score. Notice that the numerator is cast as a decimal to ensure the division retains the fractional component.

7. Take a minute to review the final code section:

```
c#

Console.WriteLine("Student\t\tGrade\n");
Console.WriteLine("Sophia:\t\t" + sophiaScore + "\tA-");
Console.WriteLine("Andrew:\t\t" + andrewScore + "\tB+");
Console.WriteLine("Emma:\t\t" + emmaScore + "\tB");
Console.WriteLine("Logan:\t\t" + loganScore + "\tA-");

Console.WriteLine("Press the Enter key to continue");
Console.ReadLine();
```

This section prints the formatted output in accordance with the teacher's guidelines. The first line is a header line with column titles, followed by the names and scores for each student.

The `Console.ReadLine()` statement will pause the application so that the application user can review the output.

8. In the Visual Studio Code EXPLORER panel, right-click **Starter**, and then select **Open in Integrated Terminal**.

You will be using the TERMINAL panel to run .NET Command Line Interface (CLI) commands, such as `dotnet build` and `dotnet run`. The `dotnet build` command will compile your code and display error and warning messages related to your code syntax.

> **ⓘ Important**
>
> Ensure that terminal command prompt is open to the root of your project workspace. In this case, the root of your project workspace is the Starter folder, where your Starter.csproj and Program.cs files are located. When you run .NET CLI commands in the

terminal, the commands will try to perform actions using the current folder location. If you try to run the `dotnet build` or `dotnet run` commands from a folder location that does not contain your files, the commands will generate error messages.

9. At the TERMINAL command prompt, to build your project code, enter the following command: `dotnet build`

   After a couple seconds, you should see a message telling you that your build succeeded and that you have 0 Warning(s) and 0 Error(s).

   txt

   ```
   Determining projects to restore...
   All projects are up-to-date for restore.
   Starter ->
   C:\Users\someuser\Desktop\GuidedProject\Starter\bin\Debug\net7.0\Starter.dll

   Build succeeded.
       0 Warning(s)
       0 Error(s)
   ```

   Using .NET CLI commands is an easy way to build and run your applications in Visual Studio Code.

10. At the Terminal command prompt, type **dotnet run** and then press Enter.

    The `dotnet run` command will instruct the compiler to build your application and then, as long as there were no build errors, it will run your compiled code.

    > **ⓘ Important**
    >
    > The Starter project targets .NET 7 (in the Starter.csproj file). If you don't have .NET 7 installed, the `dotnet run` command will generate an error. You can either install the .NET 7 SDK (recommended), or you can change the target framework in the Starter.csproj file to align with the version of .NET that you have installed in your environment.

11. Verify that your application produced the following output:

    Output

```
Student         Grade

Sophia:         92.2    A-
Andrew:         89.6    B+
Emma:           85.6    B
Logan:          91.2    A-
Press the Enter key to continue
```

12. In the TERMINAL panel, press the Enter key.

    Your application was essentially paused after writing "Press the Enter key to continue" to the console. This behavior is caused by the `Console.ReadLine()` statement, which is used to collect user input in a console application. Your application will stop running once you press Enter.

13. Close the Terminal panel.

That completes your code review. This application looks like the perfect opportunity to apply arrays, iterations, and selection statements.

It's time to get started on your updates!

Create the assignment score arrays

In this task, you'll replace the variables that hold the individual scores with arrays that hold the graded assignment scores for each student.

1. In the Visual Studio Code Editor, scroll to the top of the Program.cs file.
2. Create a blank code line below the line used to declare the `currentAssignments` variable.
3. On the new code line, to create an integer array that will be used for Sophia's assignment scores, enter the following code:

    C#

    ```
    int[] sophiaScores = new int[5];
    ```

    Notice that this code uses the `new` operator to specify that you're creating a new instance of an array. The set of square brackets in `int[]` tells the compiler that `sophiaScores` will be an

integer array, while the set of square brackets in `int[5]` is used to specify the number of elements in the array.

You may recall that you have the option to assign array values as part of the declaration.

> **Note**
>
> In situations where you know the values of array elements ahead of time and when the values are unlikely to change, it makes sense to assign the array values when the array is declared. A good example would be an array that's used to hold the days of the week. Obviously, if the array values are unknown, you can't specify them when the array is declared, and you'll use the syntax that you entered above to declare your array.

4. To specify Sophia's assignment scores within the declaration, update the declaration for `sophiaScores` as follows:

```
C#

int[] sophiaScores = new int[] { 90, 86, 87, 98, 100 };
```

Notice that the `5` that was used to specify the number of elements has been removed, and instead, you've included the scores of the five graded assignments.

5. Verify that the scores listed inside the `{}` match the individual scores for Sophia's assignments.

The five variables that are used to hold individual scores are as follows:

```
C#

int sophia1 = 90;
int sophia2 = 86;
int sophia3 = 87;
int sophia4 = 98;
int sophia5 = 100;
```

6. Delete the code lines that declare the variables holding Sophia's individual scores.

Since you will be using the `sophiaScores` array to access Sophia's scores going forward, these variables are no longer needed.

7. To create an integer array that will be used for Andrew's assignment scores, enter the following code:

```
C#

int[] andrewScores = new int[] {92, 89, 81, 96, 90};
```

8. Create the array declarations for the other students.

   Be sure to name the arrays using the student's name, and then copy the values of their individual scores into `{}` on the array declaration line.

9. Verify that you've copied the individual scores into the array declarations accurately, and then delete the variables used to hold the individual scores for Andrew, Emma, and Logan.

   The code at the top of your Program.cs file should now be similar to the following:

```
C#

// initialize variables - graded assignments
int currentAssignments = 5;

int[] sophiaScores = new int[] { 90, 86, 87, 98, 100 };
int[] andrewScores = new int[] { 92, 89, 81, 96, 90 };
int[] emmaScores = new int[] { 90, 85, 87, 98, 68 };
int[] loganScores = new int[] { 90, 95, 87, 88, 96 };

int sophiaSum = 0;
```

   Good job, the scores arrays are declared and ready to use.

Create a foreach iteration to calculate Sophia's grade

In this task, you'll replace the existing code that's used to perform calculations with a `foreach` statement that iterates through Sophia's assignment scores. You will use the code block of the `foreach` loop to calculate the sum of Sophia's scores, and then calculate and display Sophia's grade.

1. Locate the code lines that are used to declare variables and perform calculations for the sum and average score values.

The code should look similar to the following:

```
C#

int sophiaSum = 0;
int andrewSum = 0;
int emmaSum = 0;
int loganSum = 0;

decimal sophiaScore;
decimal andrewScore;
decimal emmaScore;
decimal loganScore;

sophiaSum = sophia1 + sophia2 + sophia3 + sophia4 + sophia5;
andrewSum = andrew1 + andrew2 + andrew3 + andrew4 + andrew5;
emmaSum = emma1 + emma2 + emma3 + emma4 + emma5;
loganSum = logan1 + logan2 + logan3 + logan4 + logan5;

sophiaScore = (decimal)sophiaSum / currentAssignments;
andrewScore = (decimal)andrewSum / currentAssignments;
emmaScore = (decimal)emmaSum / currentAssignments;
loganScore = (decimal)loganSum / currentAssignments;
```

2. Delete the code lines that are used to perform the sum calculations.

   You'll be writing the code to calculate the sum inside a `foreach` loop once you're finished cleaning up.

3. Delete the code lines that declare `int` and `decimal` variables for Andrew, Emma, and Logan.

   > ⓘ **Note**
   >
   > Leave the code lines that declare variables for Sophia.

4. Delete the code lines that are used to calculate the average score for Andrew, Emma, and Logan.

   > ⓘ **Note**
   >
   > Leave the code line containing the average score calculation for Sophia.

5. Scroll down to the bottom of your code, and then delete the code lines used to report grades for Andrew, Emma, and Logan.

6. Verify that your updated Program.cs file contains the following code:

> **Note**
>
> Add or remove blank code lines so that your code matches the code shown below.

C#

```
// initialize variables - graded assignments
int currentAssignments = 5;

int[] sophiaScores = new int[] { 90, 86, 87, 98, 100 };
int[] andrewScores = new int[] { 92, 89, 81, 96, 90 };
int[] emmaScores = new int[] { 90, 85, 87, 98, 68 };
int[] loganScores = new int[] { 90, 95, 87, 88, 96 };

int sophiaSum = 0;

decimal sophiaScore;

sophiaScore = (decimal)sophiaSum / currentAssignments;

Console.WriteLine("Student\t\tGrade\n");
Console.WriteLine("Sophia:\t\t" + sophiaScore + "\tA-");

Console.WriteLine("Press the Enter key to continue");
Console.ReadLine();
```

7. Create a blank code line after the line used to declare `sophiaScore`.

8. To create a `foreach` statement that you will use to iterate through Sophia's scores, enter the following code:

C#

```
foreach (int score in sophiaScores)
{
}
```

Notice that this code instantiates an integer variable named `score` as part of the `foreach` statement. You will be using `score` inside the code block of your `foreach` to access the

values of the `sophiaScores` array.

9. To create the equation that sums Sophia's score, update the `foreach` code block as follows:

```
C#

foreach (int score in sophiaScores)
{
    // add the exam score to the sum
    sophiaSum += score;
}
```

Notice that this code is using the `+=` operator to add the value of `score` to `sophiaSum` inside your `foreach` loop. Developers often use `+=` as a shortcut when calculating a sum. This equation is equivalent to the following:

```
C#

sophiaSum = sophiaSum + score;
```

Once your `foreach` loop has iterated through all of the values in the `sophiaScores` array, `sophiaSum` will contain the sum of her scores.

10. Take a minute to review your code.

Your code should now look similar to the following:

```
C#

// initialize variables - graded assignments
int currentAssignments = 5;

int[] sophiaScores = new int[] { 90, 86, 87, 98, 100 };
int[] andrewScores = new int[] { 92, 89, 81, 96, 90 };
int[] emmaScores = new int[] { 90, 85, 87, 98, 68 };
int[] loganScores = new int[] { 90, 95, 87, 88, 96 };

int sophiaSum = 0;

decimal sophiaScore;

foreach (int score in sophiaScores)
{
    // add the exam score to the sum
    sophiaSum += score;
```

```
}

sophiaScore = (decimal)sophiaSum / currentAssignments;

Console.WriteLine("Student\t\tGrade\n");
Console.WriteLine("Sophia:\t\t" + sophiaScore + "\tA-");

Console.WriteLine("Press the Enter key to continue");
Console.ReadLine();
```

At this point, you've created the scores arrays for all students and you're calculating the sum of Sophia's score inside a `foreach` loop that iterates through her scores. You still have a long way to go to complete all of the planned updates, but this is a good point to check your progress.

11. On the Visual Studio Code **File** menu, click **Save**.

Check your work

In this task, you'll run the application to verify that your code logic is working as expected.

1. Ensure that you've saved your changes to the Program.cs file.
2. In the Visual Studio Code EXPLORER panel, right-click **Starter**, and then select **Open in Integrated Terminal**.

   You will be using the Terminal panel to enter .NET CLI commands that build and run your applications.
3. Verify that the Terminal command prompt lists the Starter folder as the current folder location.

   You should see a command prompt that appears similar to the following:

   Output

   ```
   C:\Users\someuser\Desktop\GuidedProject\Starter>
   ```
4. At the TERMINAL command prompt, to build your project code, enter the following command: `dotnet build`

After a couple seconds, you should see a message telling you that your build succeeded and that you have 0 Warning(s) and 0 Error(s).

```txt
Determining projects to restore...
All projects are up-to-date for restore.
Starter ->
C:\Users\someuser\Desktop\GuidedProject\Starter\bin\Debug\net6.0\Starter.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)
```

5. If you see Error or Warning messages, you need to fix them before continuing.

   Error and Warning messages list the code line where the issue can be found. The following is an example of a `Build FAILED` error message:

   `C:\Users\someuser\Desktop\GuidedProject\Starter\Program.cs(53,18): error CS1002: ; expected [C:\Users\someuser\Desktop\GuidedProject\Starter\Starter.csproj]`

   This message tells you the type of error that was detected and where to find it. In this case, the message tells you that the Program.cs file contains an error - `error CS1002: ; expected`. The `; expected` suggests that the developer forgot to include a `;` at the end of a statement. The `Program.cs(53,18)` portion of the message tells you that the error is located on code line 53, at a position 18 characters in from the left.

   A syntax error like this prevents the Build task from succeeding (Build FAILED). Some Build messages provide a "Warning" instead of an "Error", which means there is something to be concerned with, but you can try running the program anyway (Build succeeded).

   Once you have fixed the issues and saved your updates, you can run the `dotnet build` command again. Continue updating and saving your code until you have 0 Warning(s) and 0 Error(s).

   > **Note**
   >
   > If you have trouble resolving an issue on your own, you can examine the Program.cs code in the Final folder that's included as part of the download that you completed during Setup. The Program.cs code in the Final folder represents the conclusion of all

exercises in this module, so it will include code that you have not created yet. It may look considerably different than the Program.cs code that you have developed at this point in the Guided project. However, you can try examining the Program.cs code in Final to help you isolate and fix an issue in your code if you need to. Try not to use the code in the Final folder as a guide if you can avoid it. Remember that you learn from your mistakes and that every developer spends time finding and fixing errors.

6. At the Terminal command prompt, type **dotnet run** and then press Enter.

   The `dotnet run` command instructs the compiler to build your application and then, as long as no errors were detected, will run the compiled code.

7. Verify that your code produced the following output:

   Output

   ```
   Student         Grade

   Sophia:         92.2    A-
   Press the Enter key to continue
   ```

8. In the TERMINAL panel, to stop your running application, press the Enter key.

9. Close the Terminal panel.

Congratulations, you've built an application that uses a `foreach` loop to iterate through the elements of an array and perform calculations based on the contents of the array. You're making great progress on the required app updates!

Next unit: Exercise - Construct a nested loop structure for student grade calculations

Continue >