Loops and Collections

The objective of this exercise is to consolidate your understanding of loops and collections.

1	Create a new **console** project called **Averages** in: **{installedFolder}\Labs\05_Loops_and_Collections\Begin**

2	In **Program.cs**, delete the comment and the line of code and replace with the following code:
	
	`AverageCalculator calculator = newAverageCalculator();`
	
	`calculator.AveragesWithWhile();`
	`Console.WriteLine("==========");`
	`calculator.AveragesWithDoWhile();`
	`Console.WriteLine("==========");`
	`calculator.AveragesWithFor();`

3	Resolve the issues as follows: • Ctrl-dot on AverageCalculator, and select **Generate class in new file** • Ctrl-dot on each of the three methods and select **Generate method** • In each of the three generated methods in the AverageCalculator class, remove the line with the NotImplementedException

4	In the **AveragesWithWhile** method, put the following code:
	
	`double total = 0.0;`
	`int count = 0;`
	
	`Console.Write("Enter the first number, or Q to quit: ");`
	`string input = Console.ReadLine();`

5	On the following line, type **'while'** then press **tab twice** to insert a code snippet. As the snippet is inserted, the word 'true' is highlighted. Overtype that with: **input.ToUpper() != "Q"** The full method should now look like the following:

```
internal void AveragesWithWhile()
{
    double total = 0.0;
    int count = 0;

    Console.Write("Enter the first number, or Q to quit: ");
    string input = Console.ReadLine();
    while (input.ToUpper() != "Q")
    {

    }

}
```

6 Inside the while loop, put the following code:

```
total += double.Parse(input);
            count++;

Console.Write("Enter another number, or Q to quit: ");
            input = Console.ReadLine();
```

7 After the end of the while loop, put the following code:

```
Console.WriteLine($"The average of those numbers is {total / count}");
```

8 Test **AveragesWithWhile** by running the program.

Press F5, enter some numbers when prompted, and when you're ready to see the average, enter **Q** to quit.

9 Take a moment to look at the code you've just written. Note the following points:

- In the condition for the while loop, you use **input.ToUpper** rather than just **input**. This means that the loop will end when the user types either 'q' or 'Q', because you turn the user's input to upper-case before comparing it to the letter 'Q'.
- You have to read from the console immediately before the start of the while loop to decide whether to enter it for the first time. You do this again at the end of the loop to decide whether to repeat the loop. This pattern, where you set a variable before the loop and then change it at the very end of the loop, is quite common.
- Note how you use **double.Parse** to parse the string input from the user. **Console.ReadLine** will only ever return a string, so if you want to treat the input data as any other data type, it will always need to be parsed first.
- If the user types **'Q'** in place of the first number, the loop will not execute at all. This is a key feature of while loops; they execute zero or more times.

	• What do you think the outputted result will be if you type 'Q' in place of the first number? Try it and see if you are right.

10	In the previous step, when you tried entered **'Q'** to quit without performing any average, the program displayed the output **'NaN'**. This stands for 'Not a Number', because when you divide by zero there is no valid numeric answer.
	
	(As an aside, if you divide an *integer* by zero, you will get an *exception*, but if you divide a *double* by zero the answer is *NaN* and there is no exception.)
	
	Let's fix that problem. Replace the final **Console.WriteLine** with the following:
	
	`if (count == 0)`
	`  {`
	`Console.WriteLine("You didn't enter any numbers");`
	`  }`
	`else`
	`  {`
	`Console.WriteLine($"The average of those numbers is {total / count}");`
	`  }`

11	Confirm the behaviour of the program by running it and immediately entering 'Q' versus running it and entering some valid numbers before quitting.
12	Now see if you can achieve the same behaviour with a **do…while** loop. Write code within **AverageWithDoWhile**. Use the 'do' code snippet to get started.
13	When you have completed your code, run the program. When you press F5, the first sequence of numbers you enter is controlled with a **while** loop. After you press Q, it will prompt you for a second set of numbers; this is what is being controlled by the new code that you have written using a **do…while** loop. Check that both types of loops give correct averages.
14	Now see if you can achieve the same behaviour with a **for** loop. Write code within **AverageWithFor**.

	Use the **'for'** code snippet to get started. Because **for** loops run a set number of times, start off by asking the user how many numbers they have before entering the loop.
15	When you have completed your code, run the program. When you press F5, the first sequence of numbers you enter is controlled with a **while** loop and the second set are controlled by the **do…while** loop. You should then be prompted for how many numbers you have before the **for** loop calculates and displays the average. Check that all three loops give correct averages.
16	A suggested solution is provided in the **End** folder for your reference.

Collections

1	Create a new **console** project called **Collections** in: **{installedFolder}\Labs\05_Loops_and_Collections\Begin**
2	Delete the contents of **Program.cs**.
3	Create an *array of strings* called **muppets** to store the following values: 'Kermit the Frog', 'Miss Piggy', 'Fozzie Bear', 'Gonzo', 'Rowlf the Dog', 'Scooter', 'Animal', 'Rizzo the Rat', 'Pepe the Prawn', 'Walter', 'Clifford'
4	Use a **foreach** loop to write the strings to the console.
5	Create a strongly typed *list of strings* called **muppetList**.
6	Add the contents of the muppets array to the list using a single method call.
7	Output the following values to the console: • The first muppet

	• The last muppet
8	Add a new string to the list: **'Beaker'**
9	Output the value of the last muppet in the list to the console and confirm it is now 'Beaker'.
10	You will now confirm the list is stronglytyped by attempting to add non-string types to the list: • Attempt to add the Boolean value *true* to the list • Attempt to add the int value *3* to the list Comment out any code that will prevent a successful build.
11	Your next task is to sort the list. Firstly, use a loop to display all the strings on one line separated by commas. Then, sort the list and re-display the strings to confirm they are sorted alphabetically.
12	You will now use different operators to extract specific strings from the list. Use an *indexer*, the *index from end* and *range* operators, to extract the following strings: • The first string • The last string • The second to last string • A slice of the strings in position 5 and 6
13	You will now create a *dictionary* to store strings for the keys and values. Call this dictionary **muppetdict**.
14	Add the following items to the dictionary: • 'Beaker', 'Meep' • 'Miss Piggy', 'Hi-ya!' • 'Kermit', 'Hi-ho!' • 'Cookie Monster', 'Om nom nom'
15	Create a variable **catchphrase** and extract the value for **Miss**

	Piggy Output this to the console.
16	You will now write three loops to iterate over the **KeyValue** pairs, **Keys**, and **Values** respectively. Within each loop, output a string containing the iterated item.
17	A suggested solution is provided in the **End** folder for your reference.