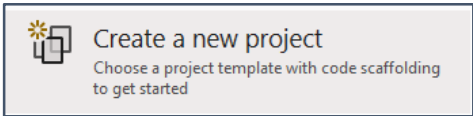
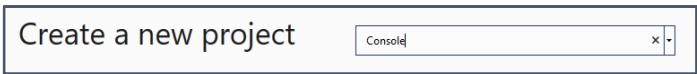
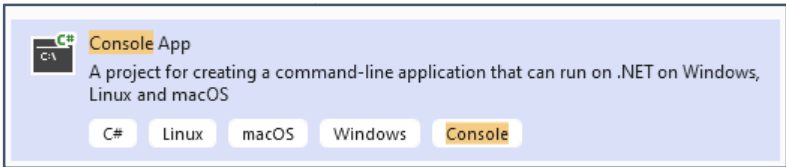
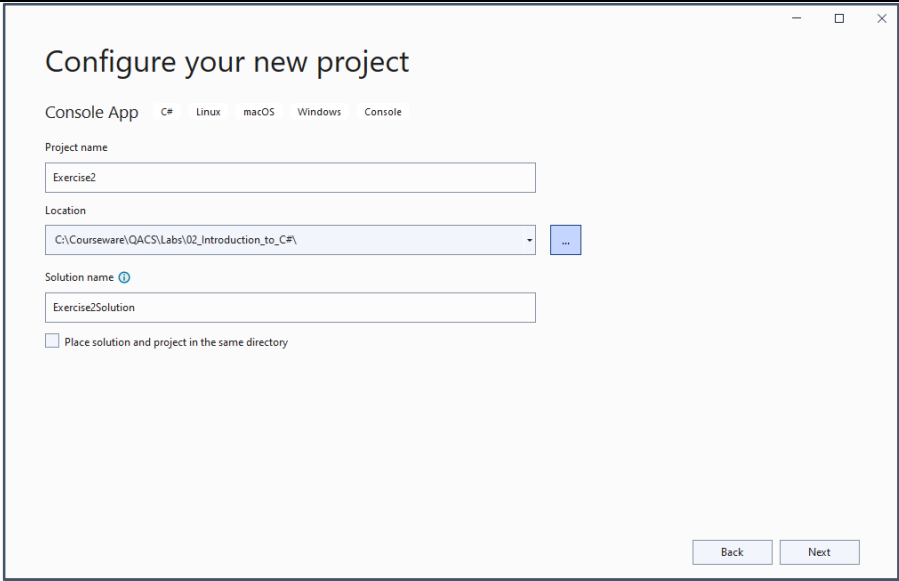
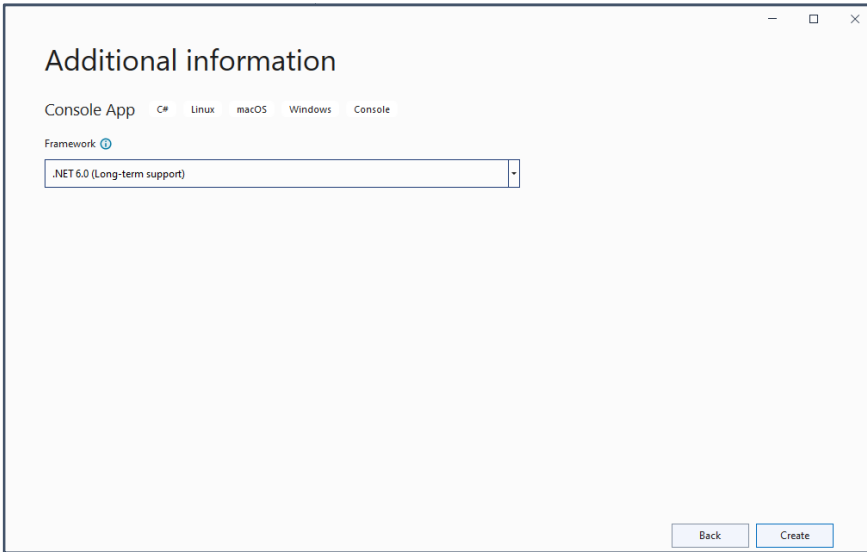

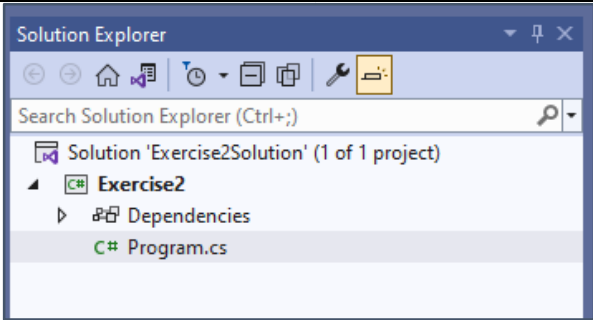
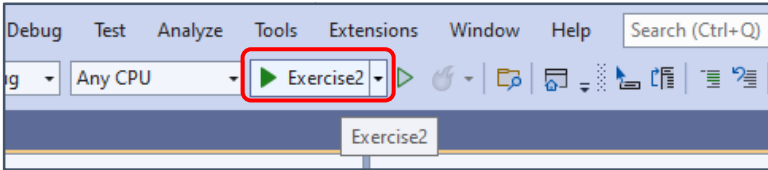
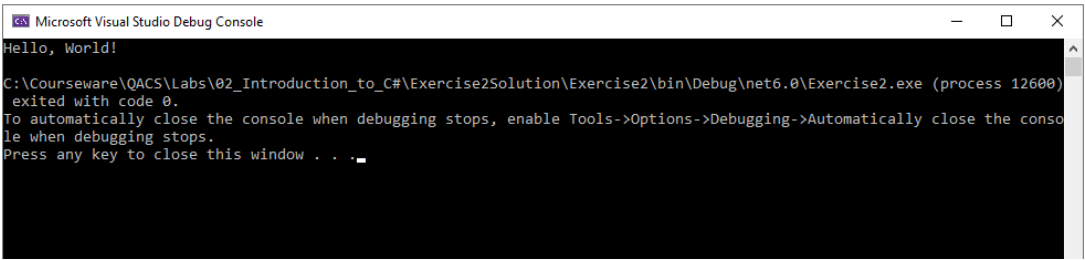
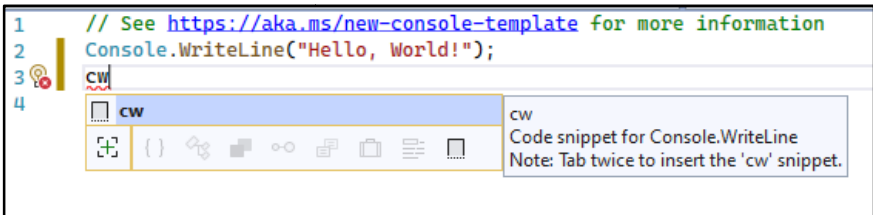


Introduction to C#

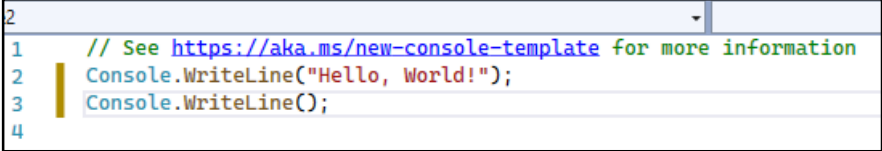
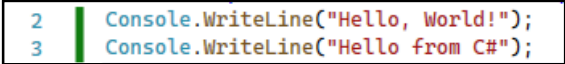
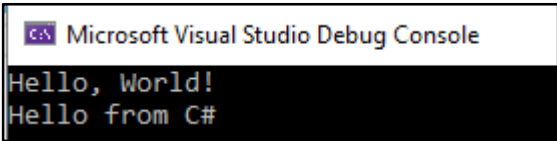
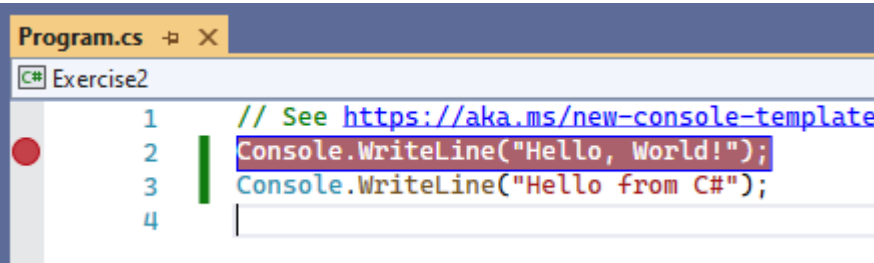
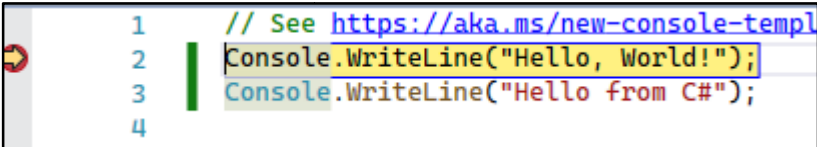
The objective of this exercise is to get you started using C# and the Visual Studio IDE and introduce you to simple debugging. You will also work with a test project and write some simple tests.

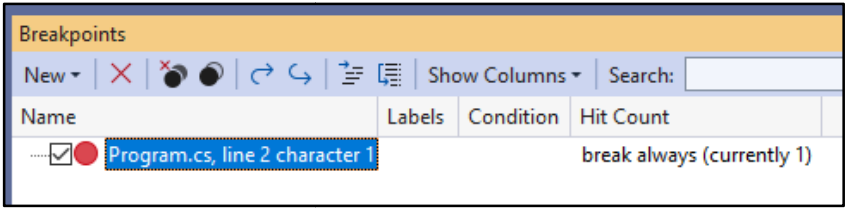
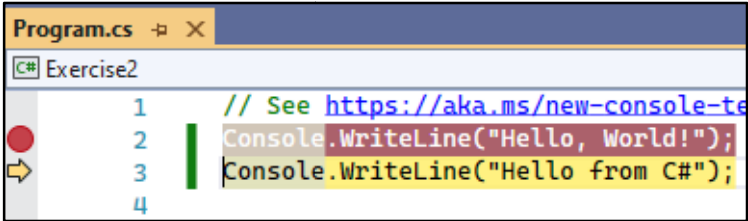
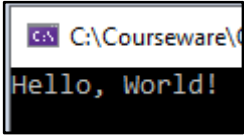
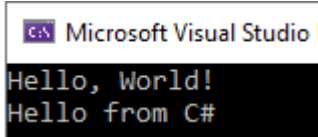
1	Start Microsoft Visual Studio 2022
2	Choose ' Create a new project ' 
3	In the Search box, type Console 
4	Select Console App and choose Next 
5	Name the <i>Project</i> Exercise2 and the <i>Solution</i> Exercise2Solution . Save the files in {installFolder}\Labs\02_Introduction_to_C#

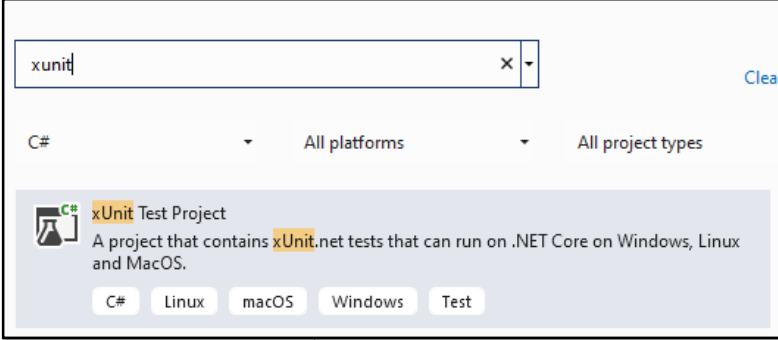
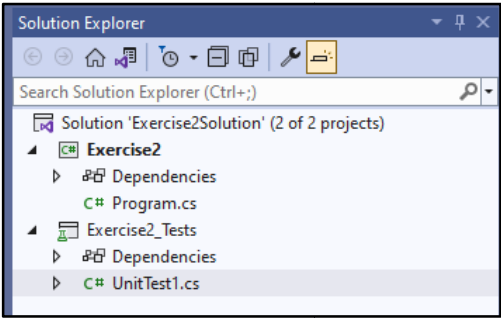
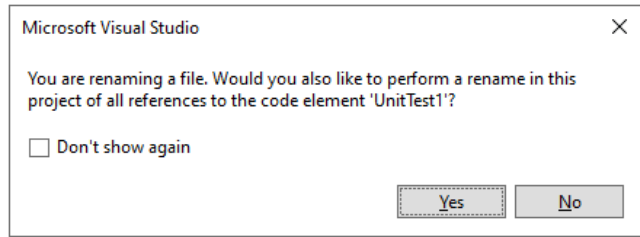
	
6	<p>Ensure .NET 6.0 (Long-term support) is selected in Additional information and click Create</p> 
7	<p>You will see a code editor window for a file called Program.cs containing the following code:</p> 
8	<p>You can see the Solution Explorer window with Program.cs being tracked as the active file.</p>

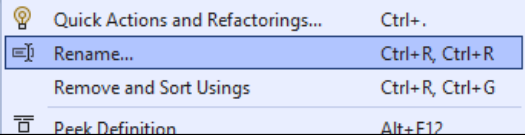
	
9	<p>From the toolbar, click the green arrow to Start with Debugging:</p> 
10	<p>Observe the output of the program and press any key to close the console window.</p> 
11	<p>Add a line of code at line 3 as follows. Type:</p> <p>CW</p> <p>Notice it is a recognised code snippet for Console.WriteLine:</p> 

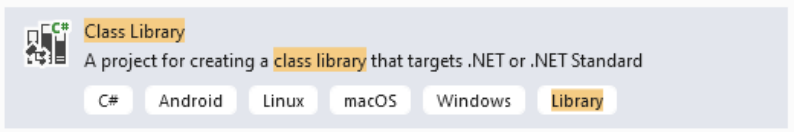
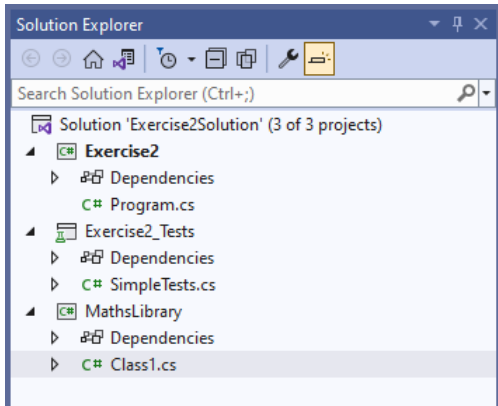
Tab twice to insert the code:

	 <pre> 2 1 // See https://aka.ms/new-console-template for more information 2 Console.WriteLine("Hello, World!"); 3 Console.WriteLine(); 4 </pre>
12	<p>Add the text string "Hello from C#" as a parameter to the WriteLine method.</p>  <pre> 2 Console.WriteLine("Hello, World!"); 3 Console.WriteLine("Hello from C#"); </pre>
13	<p>Run the program again, this time using the keyboard shortcut F5</p>  <p>Press any key to quit the program.</p>
14	<p>You will now do some very simple debugging.</p> <p>Click in the margin well to the left of line 2 to set a breakpoint:</p> 
15	<p>Debug the program with F5 and notice how the application is now paused on your breakpoint:</p>  <p>Observe the changes to the Visual Studio layout.</p> <p>Numerous debug windows are now open including Autos, Locals, Watch 1, Call Stack, and Breakpoints.</p>
16	<p>Open the Breakpoints window and observe the Hit Count value is</p>

	<p>set to "break always (currently 1)"</p> 
17	<p>You will Step Into the next line of code which will run the line that is currently highlighted.</p> <p>Press F11.</p> 
	<p>Look at the Console output window. You can see line 2 has run:</p> 
	<p>Press F11 again.</p> <p>The program ends because the last line of code has been run successfully. You can see the complete result in the output console:</p> 
18	<p>Some of the exercises on the course use tests to validate code behaviour so you will now add a test project to your solution.</p> <p>Right-click the Exercise2Solution and choose Add -> New Project</p> <p>In the search box type xunit.</p>

	<div data-bbox="261 197 1043 535">  </div> <p>Select xUnit Test Project and click Next.</p> <p>Name the project Exercise2_Tests and click Next.</p> <p>Ensure .NET 6.0 (Long-term support) is selected and click Create.</p>
19	<p>Your Solution Explorer window now contains one solution with two projects:</p> <div data-bbox="261 936 764 1252">  </div>
20	<p>Rename UnitTest1.cs (by right-clicking on the file name) to SimpleTests.cs and select YES when the following prompt displays:</p> <div data-bbox="261 1447 904 1682">  </div>
21	<p>Your SimpleTests.cs file contains the following starter code:</p>

	<pre>using Xunit; namespace Exercise2_Tests { public class SimpleTests { [Fact] public void Test1() { } } }</pre>
22	<p>Rename Test1 to Add_Two_Numbers:</p> <div><pre>public class SimpleTests { [Fact] public void Test1() { } }</pre></div> <div><p>Rename: Test1 X</p><p>New name: Add_Two_Numbers</p><p><input type="checkbox"/> Include comments</p><p><input type="checkbox"/> Include strings</p><p><input type="checkbox"/> Preview changes</p><p>Rename will update 1 reference in 1 file.</p><p>Apply</p></div> <p>Click Apply.</p> <div><pre>using Xunit; namespace Exercise2_Tests { public class SimpleTests { [Fact] public void Add_Two_Numbers() { } } }</pre></div>
23	<p>You are going to write some simple tests for a Calculator. This calculator is going to be created in a new project of type Class Library.</p>

	<p>Add a new class library project to the solution called MathsLibrary.</p>  <p>Solution Explorer should now look as follows:</p> 
24	Delete the file Class1.cs .
25	<p>In SimpleTests.cs add the following code to Add_Two_Numbers:</p> <pre>[Fact] public void Add_Two_Numbers() { // Arrange var num1 = 5; var num2 = 2; var expectedValue = 7; // Act var sum = Calculator.Add(num1, num2); //Assert Assert.Equal(expectedValue, sum); }</pre>
	<p>This test code uses the standard testing pattern called the triple A pattern: <i>Arrange</i>, <i>Act</i>, <i>Assert</i>.</p> <p><i>Arrange</i> is for setting up items you need for the test.</p> <p><i>Act</i> is for carrying out the action you are testing.</p> <p><i>Assert</i> is for confirming the acted upon code behaves as expected.</p>
26	The arrange phase creates three variables: num1 , num2 , and

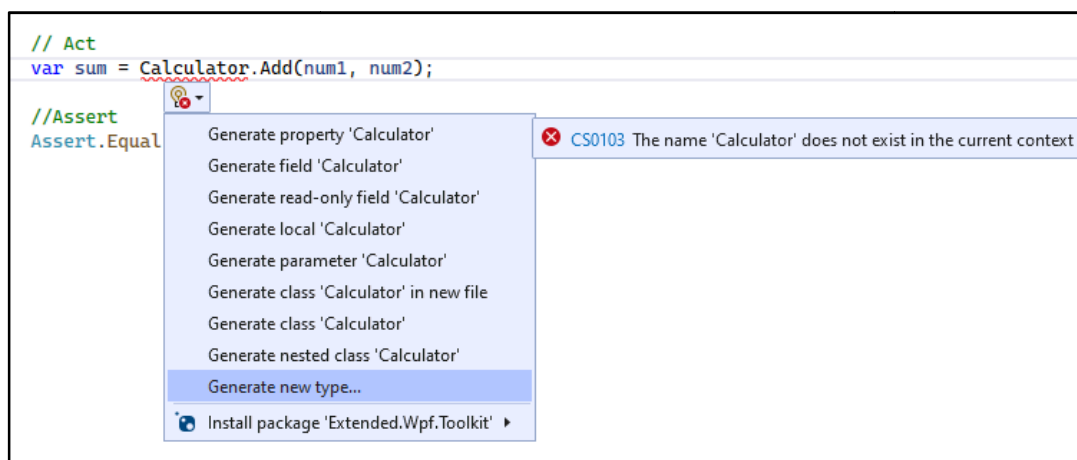
expectedValue.

The act phase calls an **Add** method on a **Calculator**, passing in **num1** and **num2** as parameters and assigning the result to a variable called **sum**.

The assert phase checks whether the **expectedValue** and the **sum** values are equal.

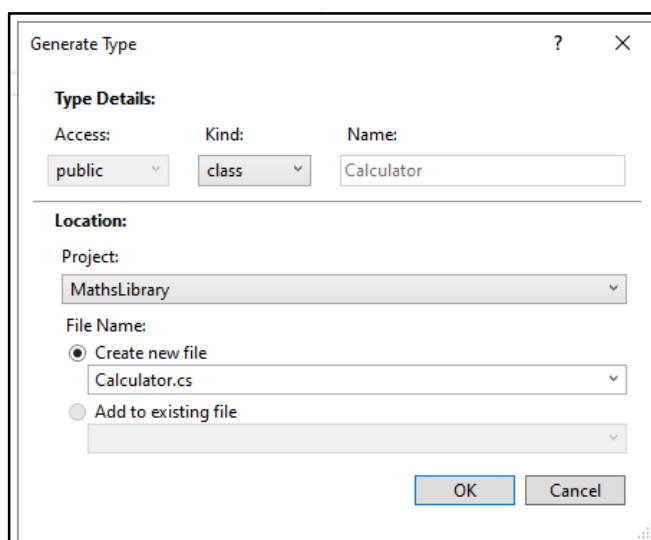
The Calculator type does not exist so you will use Visual Studio to help you create it.

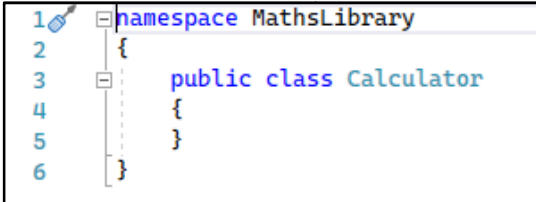
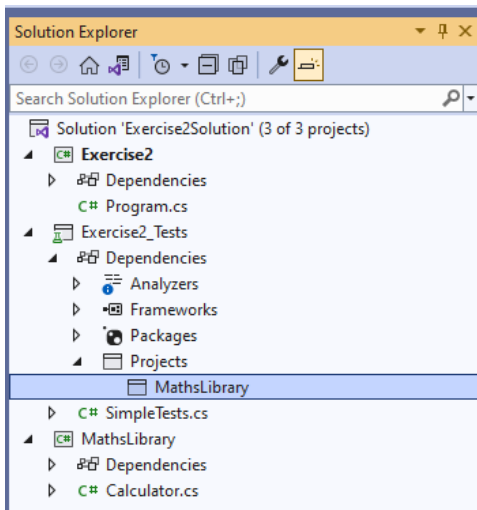
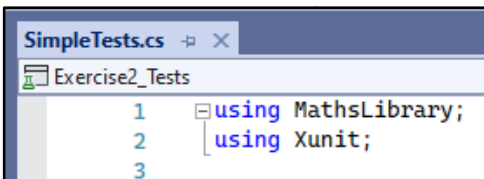
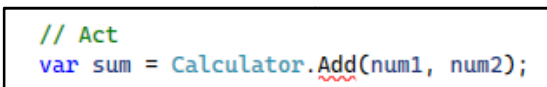
Press **Ctrl+.** (Ctrl+dot) on **Calculator** to see the available options:

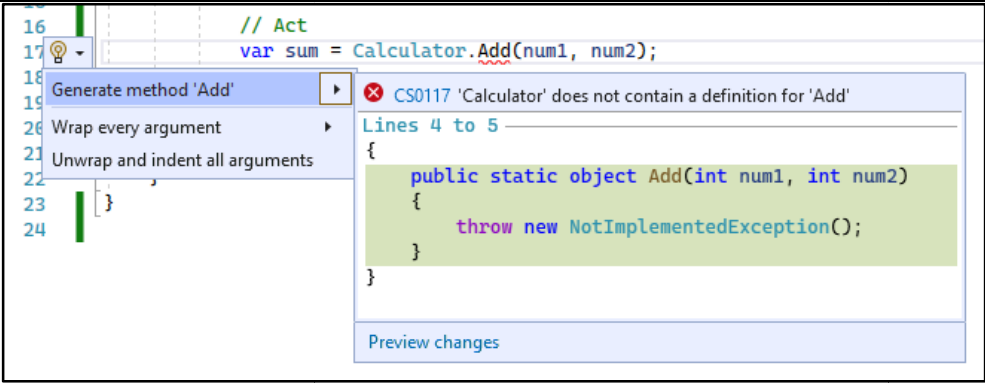
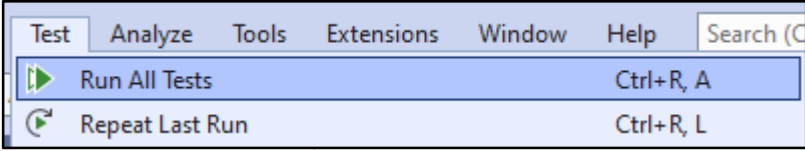


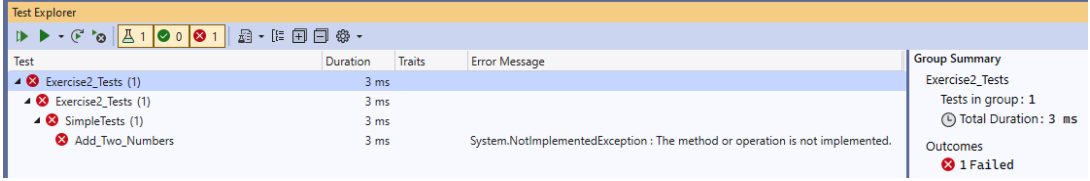
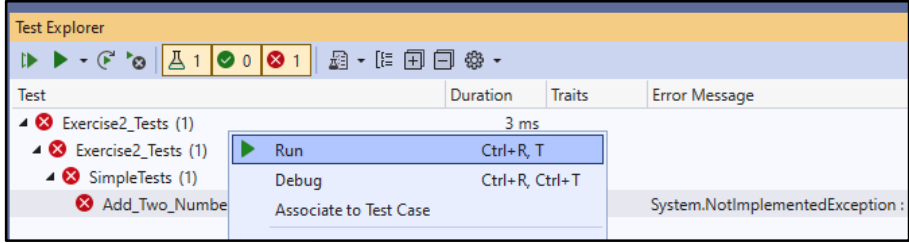
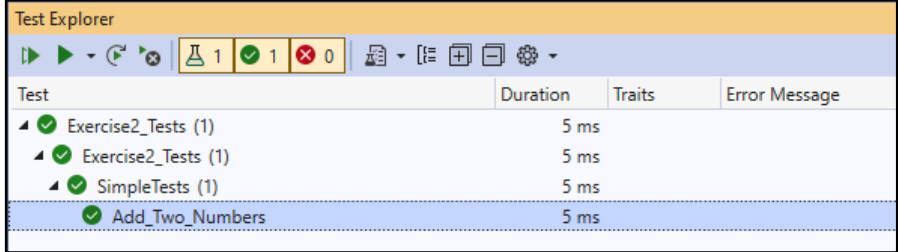
You want Calculator to be created in your **MathsLibrary** project rather than locally within the Test project so choose **'Generate new type...'**

- 27 In the dialog box, ensure a **publicclass** will be created and change the project to **MathsLibrary** and **Create new file**:



	Click OK .
28	<p>Visual Studio has created a new class (a kind of type) called Calculator in MathsLibrary:</p>  <pre> 1 namespace MathsLibrary 2 { 3 public class Calculator 4 { 5 } 6 }</pre> <p>Visual Studio has also added a <i>reference</i> to the MathsLibrary project:</p>  <p>It has also imported the MathsLibrary namespace into your test project:</p>  <pre> 1 using MathsLibrary; 2 using Xunit; 3</pre>
29	<p>The act phase of the test code now recognises the Calculator type but displays an error because Calculator does not contain a definition for Add:</p>  <pre> // Act var sum = Calculator.Add(num1, num2);</pre>
30	Use Ctrl+dot on Add to generate the method:

	
31	<p>Calculator.cs now contains an Add method:</p> <pre>public class Calculator { public static object Add(int num1, int num2) { throw new NotImplementedException(); } }</pre>
32	<p>You want your Add method to return whole numbers so change the word object to int.</p> <pre>public static int Add(int num1, int num2) { throw new NotImplementedException(); }</pre>
33	<p>You will run the test and observe the outcome.</p> <p>Test -> Run All Tests.</p> 
34	<p>Ensure the Test Explorer window is visible:</p> <p>Test -> Test Explorer.</p> <p>Expand the Test until you see the Add_Two_Numbers failed test (it appears in red) alongside the error message: The method operation is not implemented.</p>

	
35	<p>You will edit the Add method code to ensure the method is implemented and confirm that the test passes.</p> <p>Delete the line of code: throw new NotImplementedException;</p> <p>Replace the code with: return 7;</p> <pre>public static int Add(int num1, int num2) { return 7; }</pre> <p>This is hard-coding the expected value, which allows you to confirm the test is working correctly.</p>
36	<p>Right-click the failed test in Test Explorer and select Run.</p>  <p>The test should now pass:</p> 
37	<p>The final step is to refactor the code within the method to perform the calculation so that additional tests adding different integers will also pass.</p> <p>Edit the Add method as follows:</p>

	<pre>public static int Add(int num1, int num2) { return num1 + num2; }</pre>
38	<p>Re-run the test to ensure it continues to pass.</p> <p>The process that you just followed is called Test-Driven Development (TDD). It follows a three-stage approach referred to as <i>red-green-refactor</i>, whereby you write a test before implementing the code. You ensure the test fails. This is the red stage. This is to guard against any false positives. You then write enough implementation to get the test to pass. This is the green stage. You then refactor the code to improve the implementation, ensuring the tests still pass.</p>
39	<p>If you have time, write a test for a Subtract method, then use Visual Studio to help build the implementation. Use Test Explorer to run your tests.</p>
40	<p>Solutions are provided in the End folder for your reference.</p>

