Using NUnit in Visual Studio

Christofer Freeberg

WASHINGTON STATE UNIVERSITY CPT_S 321

Contents

Adding NUnit to a Project/Solution 1

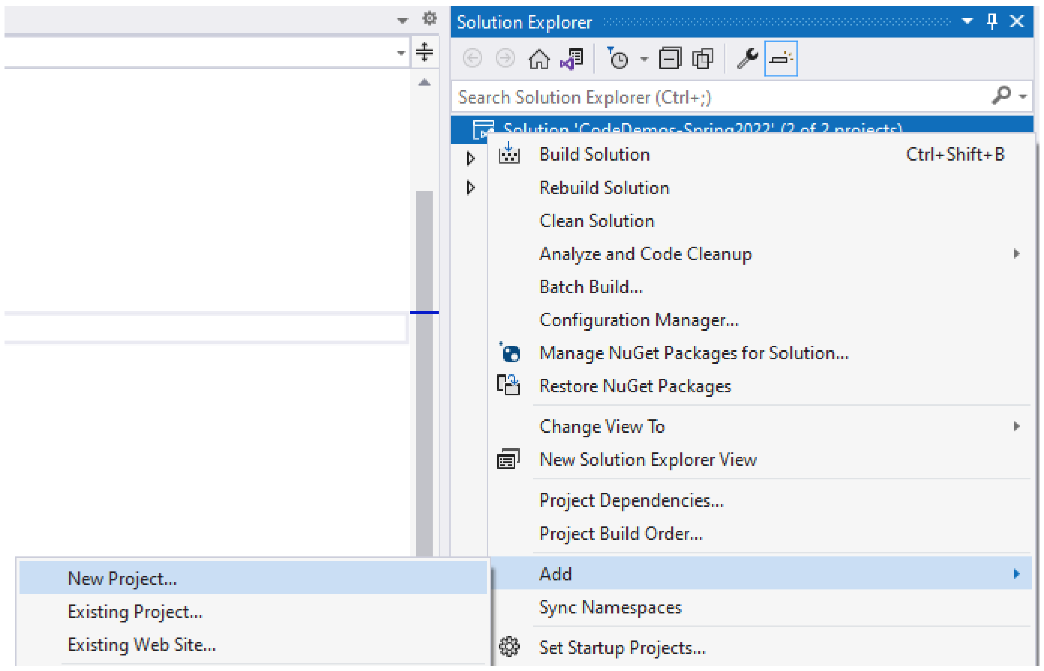
How to Use NUnit 6

- How to Run Your Tests 7
- Reading the Results 8
- Additional Resources 9

Adding NUnit to a Project/Solution

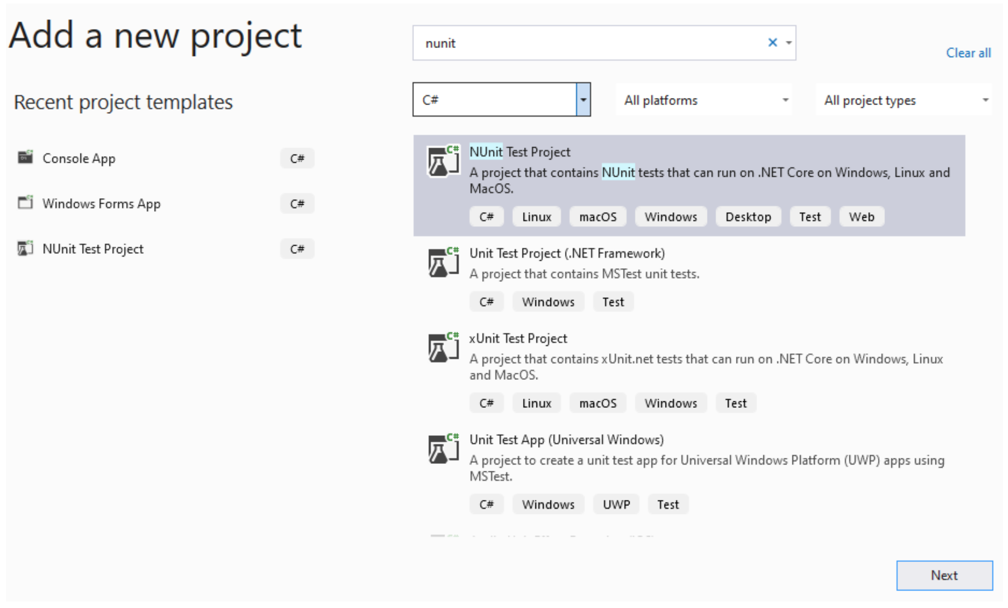
Now that NUnit is installed, we can add it to our solution

1. In Visual Studio, open the Solution Explorer and right-click on the solution and select **Add > New Project**



Note: If you can't find the Solution Explorer, it can be found under **View > Solution Explorer**

2. If NUnit does not appear on the left as shown here, search for "nunit" in the search bar. Make sure you select C# as a language. Select it and click **Next**.



NUnit Test Project
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

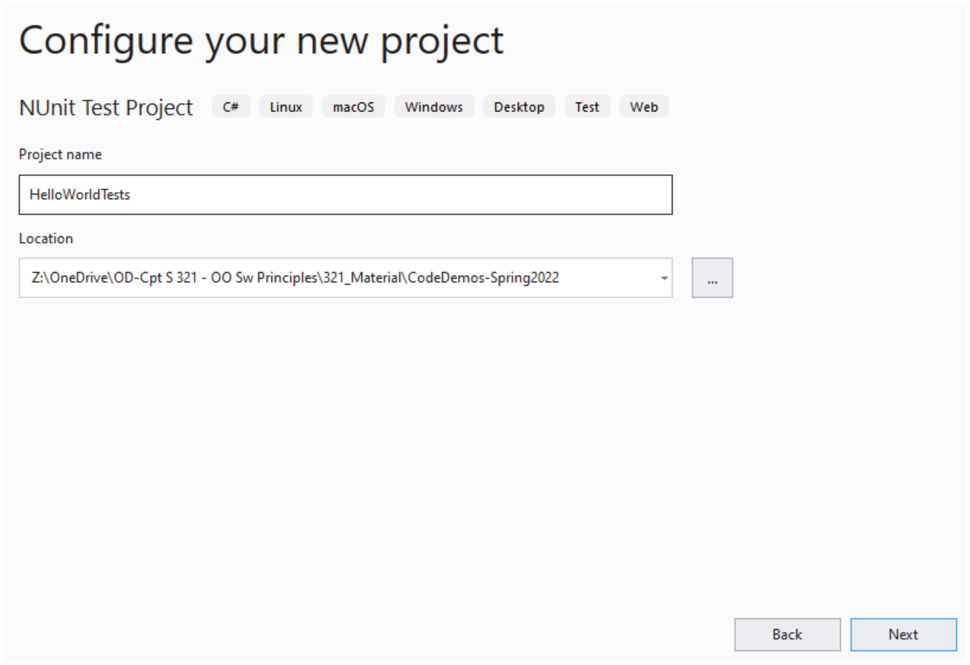
Unit Test Project (.NET Framework)
A project that contains MSTest unit tests.
C# Windows Test

xUnit Test Project
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Test

Unit Test App (Universal Windows)
A project to create a unit test app for Universal Windows Platform (UWP) apps using MSTest.
C# Windows UWP Test

3. Give your test project a name and select a location and click **Next**.

Note: It is recommended to select the same location as the rest of the solution.



Configure your new project

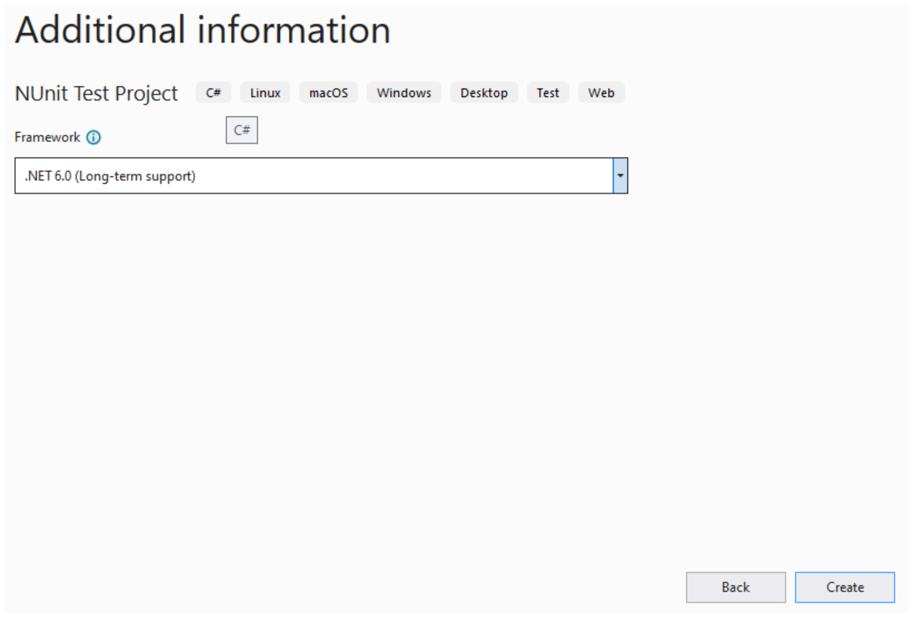
NUnit Test Project

Project name: HelloWorldTests

Location: Z:\OneDrive\OD-Cpt S 321 - OO Sw Principles\321_Material\CodeDemos-Spring2022

4. Select the targeted framework and click on create.

Note: The suggested one is typically what you want.

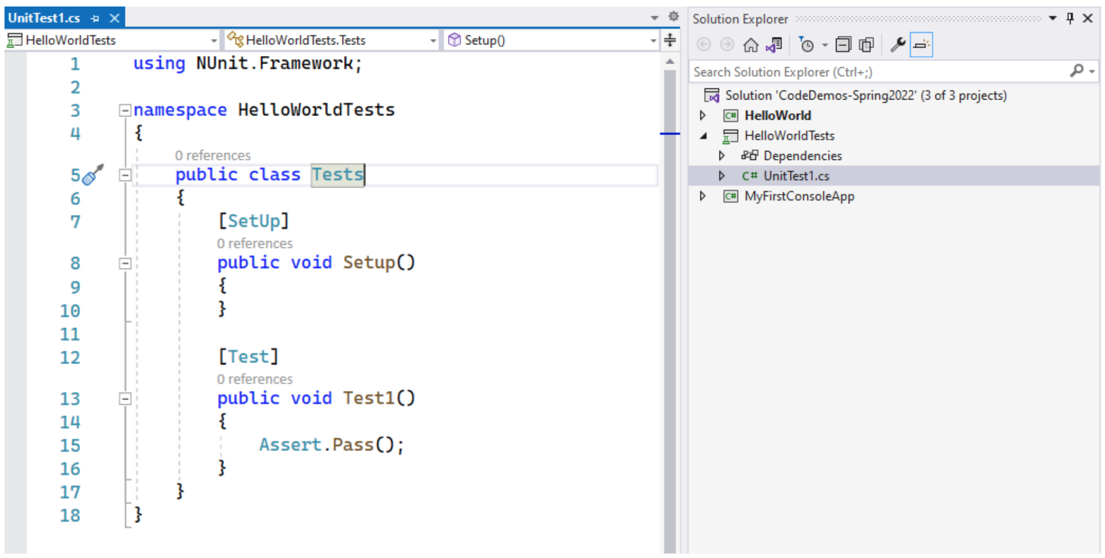


Additional information

NUnit Test Project

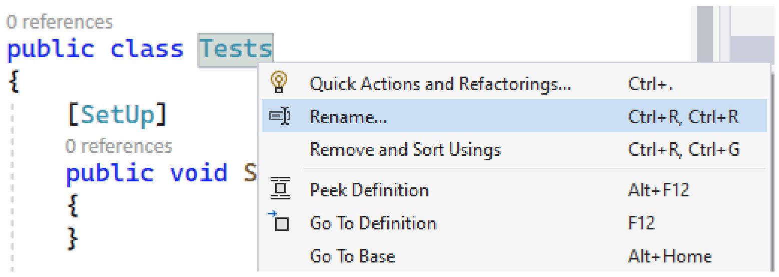
Framework: .NET 6.0 (Long-term support)

5. Visual Studio should have created a new project with the given name in the solution with a file called UnitTests1.cs or something similar.



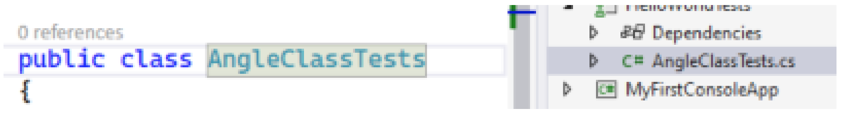
```
using NUnit.Framework;

namespace HelloWorldTests
{
    public class Tests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void Test1()
        {
            Assert.Pass();
        }
    }
}
```

6. We want to rename this file to start with the name of the class we are testing and end with



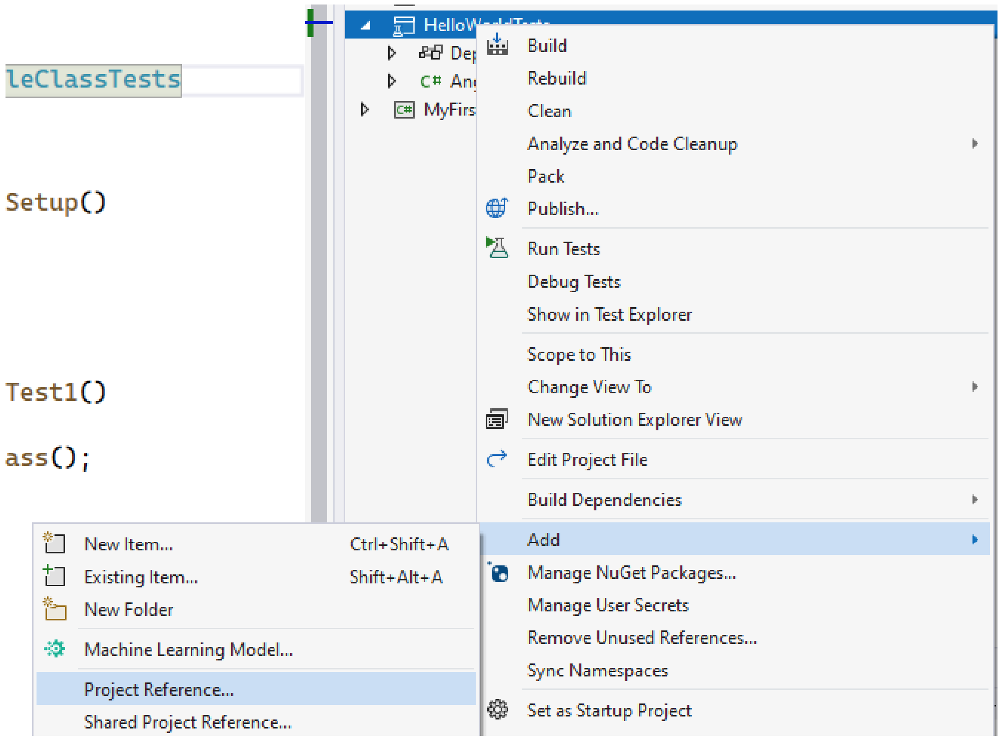
```
0 references
public class Tests
{
    [SetUp]
    0 references
    public void S
    {
    }
```

"Tests.cs". Note that we want the class name to match the file name without the extension.



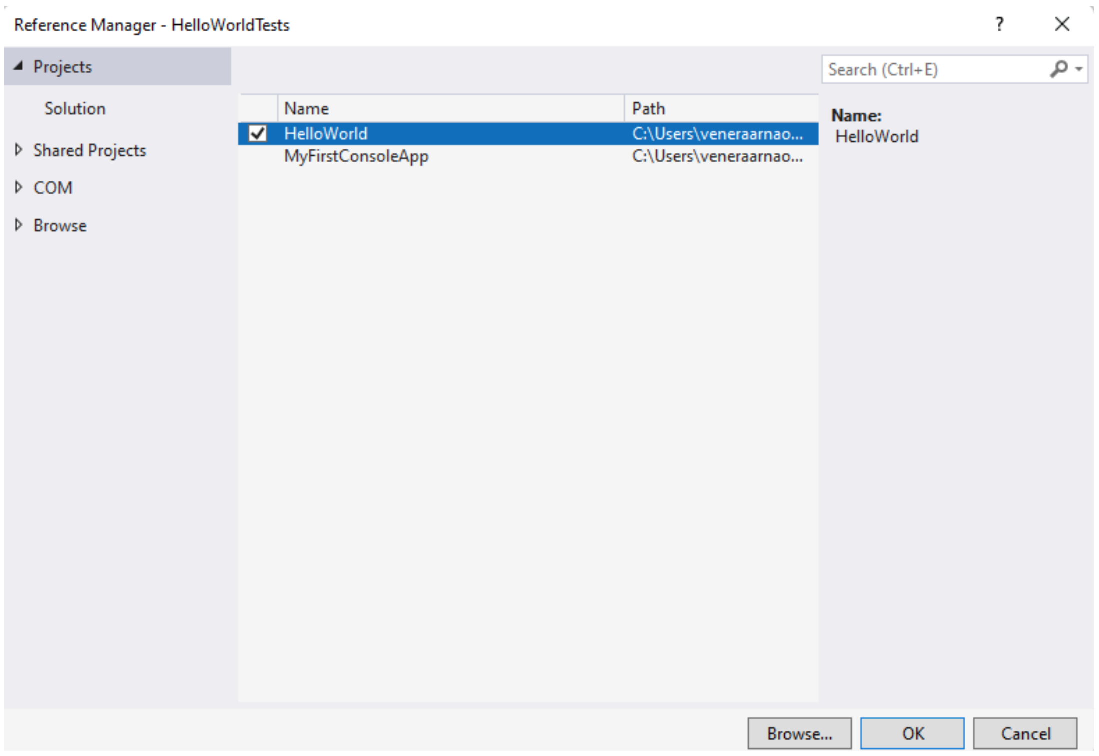
```
0 references
public class AngleClassTests
{
```

7. In the Solution Explorer, right-click on the new test project, then hover over **Add**, then click on **Project Reference..**



```
leClassTests
Setup()
Test1()
ass();
```

8. Under **Projects**, select the project you are testing in the this test project. Then click **OK**.



	Name	Path
✓	HelloWorld	C:\Users\veneraarnao...
	MyFirstConsoleApp	C:\Users\veneraarnao...

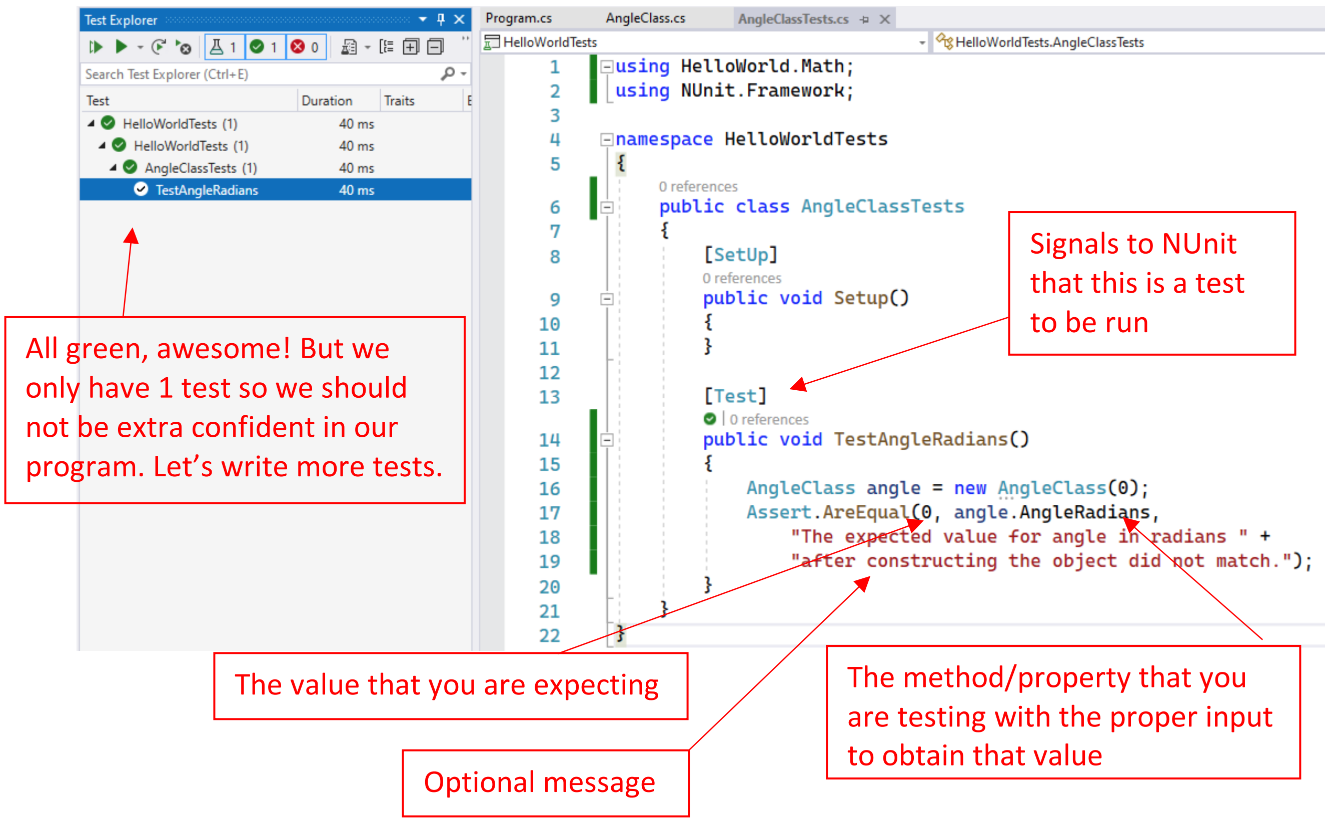
Name:
HelloWorld

How to Use NUnit

Let’s write some tests!

If you open up AngleClassTest.cs from the Solution Explorer, you will see that there is already a template written for us (a test method called **Test1**). We can use this template to write our tests.

Let’s look at an example test.



Test Explorer
Search Test Explorer (Ctrl+E)
Test
Duration
Traits
HelloWorldTests (1) 40 ms
HelloWorldTests (1) 40 ms
AngleClassTests (1) 40 ms
TestAngleRadians 40 ms
Program.cs
AngleClass.cs
AngleClassTests.cs
HelloWorldTests
HelloWorldTests.AngleClassTests
0 references
All green, awesome! But we only have 1 test so we should not be extra confident in our program. Let’s write more tests.
Signals to NUnit that this is a test to be run
The value that you are expecting
Optional message
The method/property that you are testing with the proper input to obtain that value

```
using HelloWorld.Math;
using NUnit.Framework;

namespace HelloWorldTests
{
    public class AngleClassTests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void TestAngleRadians()
        {
            AngleClass angle = new AngleClass(0);
            Assert.AreEqual(0, angle.AngleRadians,
                "The expected value for angle in radians " +
                "after constructing the object did not match.");
        }
    }
}
```

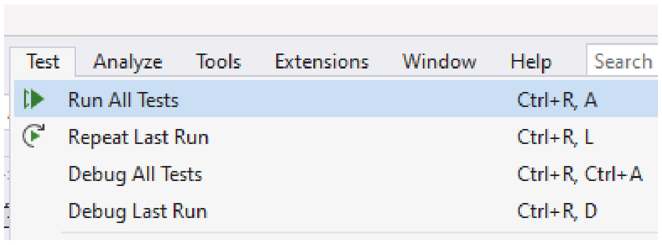
Note: It is a good practice to have a limited number of assert statements per test method. This allows for easier debugging.

How to Run Your Tests

There are two ways to run your tests, both will yield the same results.

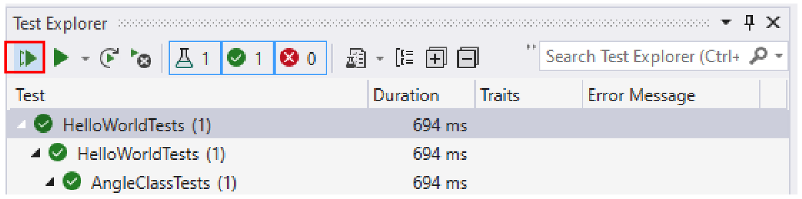
First Method:

Click on **Tests > Run > All Tests**.



Second Method:

Open the Test Explorer (**View > Test Explorer**) and click on the first button with two green arrows (if you hover over it it will say "**Run All Tests in View")**

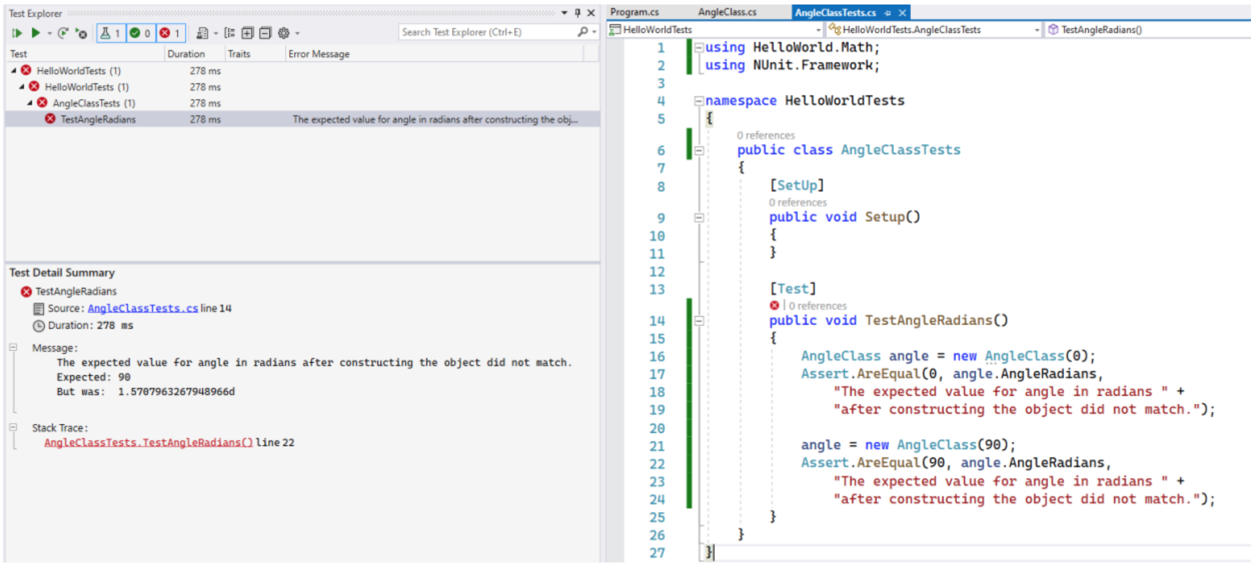


Test Explorer

Test	Duration	Traits	Error Message
HelloWorldTests (1)	694 ms		
HelloWorldTests (1)	694 ms		
AngleClassTests (1)	694 ms		

Reading the Results

After writing another assert statement in the test, it does not pass anymore – you see a red X in the Test Explorer:



Test	Duration	Traits	Error Message
HelloWorldTests (1)	278 ms		
HelloWorldTests (1)	278 ms		
AngleClassTests (1)	278 ms		
TestAngleRadians	278 ms		The expected value for angle in radians after constructing the obj...

```
Test Detail Summary
TestAngleRadians
  Source: AngleClassTests.cs line 14
  Duration: 278 ms

Message:
    The expected value for angle in radians after constructing the object did not match.
    Expected: 90
    But was:  1.5707963267948966d

Stack Trace:
  AngleClassTests.TestAngleRadians() line 22
```

```
1   using HelloWorld.Math;
2   using NUnit.Framework;
3
4   namespace HelloWorldTests
5   {
        0 references
6       public class AngleClassTests
7       {
8           [SetUp]
            0 references
9           public void Setup()
10          {
11          }
12
13          [Test]
            | 0 references
14          public void TestAngleRadians()
15          {
16              AngleClass angle = new AngleClass(0);
17              Assert.AreEqual(0, angle.AngleRadians,
18                  "The expected value for angle in radians " +
19                  "after constructing the object did not match.");
20
21              angle = new AngleClass(90);
22              Assert.AreEqual(90, angle.AngleRadians,
23                  "The expected value for angle in radians " +
24                  "after constructing the object did not match.");
25          }
26      }
27  }
```

If you click on a failed test you can see what the expected value of the test as well as the actual value at the bottom of the Test Explorer. This can be useful for debugging. If a test contains more than 1 assert statements, you will also have the line of the assert that failed (in this case line 22).

Additional Resources

Visit the following resources for more examples:

https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-nunit

http://dotnetpattern.com/nunit-assert-examples