Exercise: `VectorCalculator` **- a vector calculator**

In this exercise, you will create a vector calculator (VectorCalc) which operates on 2-dimensional Cartesian vectors. Then, you will test it using NUnit.

As you go along, think of possible interesting test scenarios, especially "corner cases" and exceptions – these cases are most often forgotten and thus error-prone.

Through this exercise, you will gain a lot of experience in the basic use of NUnit – setup, teardown, tests and test cases. This routine is necessary and a prerequisite for later exercises.

Exercise 1: Get ready!
Download the solution prepared for this exercise. This solution contains a C# class library project containing the implementation of a 2-dimensional vector (class Vector).

Add a Nunit C# unit test project to the solution. Name this project `VectorCalculator.Test.Unit`. In this project, add a reference to the application project.

Now you are set up! From here on, all extensions to the application goes in the application project VectorCalculator while all test code goes in `VectorCalculator.Test.Unit`

Exercise 2:
Develop and test a class VectorCalc with a minimum of four of the below features:

1. Add two vectors and return the resultant vector
2. Subtract two vectors and return the resultant vector
3. Scale a vector and return the resultant vector
4. Calculate the dot product of two vectors and return it[1]
5. Calculate the angle between two vectors and return the angle in radians
6. Calculate the magnitude of a vector and return it.
7. Transform the Cartesian vector to a polar one (requires implementation of the class `PolarVector`

(If there is one or two of you who have forgotten how to take the dot product of two vectors, Uncle Google knows…)

Exercise 3: Implement your tests
Implement your unit tests in the file you added to the test project above – test the class `Calculator` as thoroughly as you can using unit tests. Is it difficult?

Exercise 4: Investigate the NUnit [TestCase] attribute
NUnit offers a number of facilities for the definition of repetitious test cases. For example, you may have perhaps 6 individual tests of the method `Add(Vector v1, Vector v2)`, one for each combination of positive, negative, and 0 value coordinates of the two arguments. Each of these methods are tagged with the property `[Test]`.

While this works just fine, it gets somewhat tedious to duplicate the test code six times, only varying the arguments. Instead, you can use the `[TestCase]` attribute to specify arguments for the individual tests. Investigate this (see e.g. TAOUT ch. 2.5 or the NUnit documentation) and refactor your test suite to use `[TestCase()]` instead of `[Test]`.

[1] Asserting that two floating-point numbers (i.e. doubles) are equal is not done using the equality operator ('=='). Because of how floating point numbers are stored in memory, two numbers may be different even though they look the same. Instead, you assert that the difference between them is less than the smallest number that can be represented, namely the constant Double.Epsilon.