Lab Exercise: Unit-testing class `Calculator`

In this exercise, you will use your existing implementation of class `Calculator` to create your first NUnit test suite of an application class. You will organize your application code and test code in separate projects and "link" them together, just as you saw in the introductory video `NUnitIntro.mp4` on Blackboard for this class. If you have *not* watched the video, do so now!

It is a prerequisite for this exercise that you have downloaded and installed NUnit and ReSharper. Again, check Blackboard for instructions

---

In Lab Exercise 1, you created class `Calculator` and tested it as well as you could using hand-testing. In this exercise you will use NUnit to test the class again.

Exercise 1: Get ready!

1. Create a solution for this exercise.
2. Copy the *application* project (not the test project) from Exercise 1 to the new solution folder and include the application project in the solution.
3. Add another C# class library project to the solution. Name this project `Calculator.Test.Unit`. In this project, add a reference to the application project and a reference to `NUnit.Framework.dll`.

At this time, you should have 1 C# *solution* containing 2 C# *class library projects*, one named `Calculator` (this is the *application* project) and one named `Calculator.Test.Unit` (this is the *test* project).

Exercise 2: Define your test fixture

Add a new C# source file to the test project (that's `Calculator.Test.Unit`). In this file, define the class `CalculatorUnitTests`. This class is your test fixture and will hold all your unit tests for class `Calculator`.

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 NUnit tests. Is it difficult?

Exercise 4: Compare and contrast

Compare your tests in the two lab exercises and reflect on hand-testing vs. unit testing with a framework:

udvidelsesmuligheder
Extensibility Which form of test is easiest to extend, e.g. if new functionality is required for class `Calculator`?

Vedligeholdelse
Maintainability Which form of test is easiest to maintain?

Lærbarhed
Readability Imagine you are new to the project. Which form of test is easiest to read?

Automatisering
Automation Which type of test is easiest to automate? If you wanted to collect and compare test results every 15 minutes, which kind of test is it easiest to see if passed or failed?

Exercise 5 (optional): Investigate the `[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 `Power()`, one for each combination of positive, negative, and 0 value of the two arguments. Each of these methods are tagged with the property `[Test]`.

While this works just fine, it gets kind of 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 and refactor your test suite to use `[TestCase]` instead of `[Test]`.