**WEEK-2\_ NUnit-Handson**

**Objectives**

* Explain the meaning of Unit testing and its difference on comparison with Functional testing
  + Smallest unit to test mocking dependencies
* List various types of testing
  + Unit testing, Functional testing, Automated testing, Performance testing
* Understand the benefit of automated testing
* Explain what is loosly coupled & testable design
  + Write code that is NOT dependent on the class for data.
* Write your first testing program to validate a calculator addition operation
  + TestFixture, Test
* Understand the need of [SetUp], [TearDown] & [Ignore] attributes.
* Explain the benefit of writing parameterised test cases.
  + TestCase

**TestFixture & Test**

Please download the application available [here](https://cognizantonline.sharepoint.com/:u:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/DotNet/02%20-%20NUnit,%20C%23%204.5,%20ASP.Net%20Core/Handson/CalcLibrary.zip?csf=1&e=aLxB66). This will be used to write Unit test cases  
  
Follow the steps listed below to write the NUnit test cases for the application.

* Create a Unit test project(.Net Framework) in the solution provided.
* Add the CalcLibrary project as reference
* Create a class “CalculatorTests” to write all the test cases for the methods in the solution
* Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively
* Create a Test method to check the addition functionality
* Use the ‘TestCase’ attribute to send the inputs and the expected result
* Use Assert.That to check the actual and expected result match

**CODE**:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

[TestClass]

public class CalculatorTests

{

private SimpleCalculator calc;

[TestInitialize]

public void Setup()

{

calc = new SimpleCalculator();

}

[TestMethod]

[DataRow(5, 3, 8)]

[DataRow(-2, 2, 0)]

public void TestAddition(double a, double b, double expected)

{

double result = calc.Addition(a, b);

Assert.AreEqual(expected, result);

}

[TestMethod]

public void TestSubtraction()

{

double result = calc.Subtraction(10, 4);

Assert.AreEqual(6, result);

}

[TestMethod]

public void TestMultiplication()

{

double result = calc.Multiplication(4, 5);

Assert.AreEqual(20, result);

}

[TestMethod]

public void TestDivision()

{

double result = calc.Division(20, 4);

Assert.AreEqual(5, result);

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void TestDivisionByZero()

{

calc.Division(5, 0); // This should throw an exception

}

}

}

OUTPUT:

