NUnit Unit Testing Framework VB.NET Cheat Sheet

AUTOMATE THE PLANET

Installation

```
Install-Package NUnit
Install-Package NUnit.TestAdapter
Install-Package Microsoft.NET.Test.Sdk
```

Test Execution Workflow

```
Imports NUnit.Framework
Namespace NUnitUnitTests
    ' A class that contains NUnit unit tests. (Required)
    <TestFixture>
    Public Class NonBellatrixTests
        <OneTimeSetUp>
        Public Sub ClassInit()
            ' Executes once for the test class. (Optional)
        End Sub
        <SetUp>
        Public Sub TestInit()
            ' Runs before each test. (Optional)
        End Sub
        <Test>
        Public Sub TestMethod()
        End Sub
        <TearDown>
        Public Sub TestCleanup()
            ' Runs after each test. (Optional)
        End Sub
        <OneTimeTearDown>
        Public Sub ClassCleanup()
            ' Runs once after all tests in this class are executed. (Optional)
            ' Not guaranteed that it executes instantly after all tests from the class.
        End Sub
    End Class
End Namespace
' A SetUpFixture outside of any namespace provides SetUp and TearDown for the entire assembly.
<SetUpFixture>
Public Class MySetUpClass
    <OneTimeSetUp>
    Public Sub RunBeforeAnyTests()
        ' Executes once before the test run. (Optional)
    End Sub
    <OneTimeTearDown>
    Public Sub RunAfterAnyTests()
        ' Executes once after the test run. (Optional)
    End Sub
End Class
```

Attributes

NUNIT 3.X	MSTEST V2.X	XUNIT.NET 2.X	COMMENTS
<Test>	<TestMethod>	<Fact>	Marks a test method.
<TestFixture>	<TestClass>	n/a	Marks a test class.
<SetUp>	<TestInitialize>	Constructor	Triggered before every test case.
<TearDown>	<TestCleanup>	IDisposable.Dispose	Triggered after every test case.
<OneTimeSetUp>	<ClassInitialize>	IClassFixture<T>	One-time triggered method before test cases start.
<OneTimeTearDown>	<ClassCleanup>	IClassFixture<T>	One-time triggered method after test cases end.
<Ignore("reason")>	<Ignore>	<Fact(Skip="reason")>	Ignores a test case.
<Property>	<TestProperty>	<Trait>	Sets arbitrary metadata on a test.
<Theory>	<DataRow>	<Theory>	Configures a data-driven test.
<Category("")>	<TestCategory("")>	<Trait("Category", "")>	Categorizes the test cases or classes.

Assertions – Classic Model

```
Assert.AreEqual(28, _actualFuel) ' Tests whether the specified values are equal.
Assert.AreNotEqual(28, _actualFuel) ' Tests whether the specified values are unequal. Same as AreEqual for numeric values.
Assert.AreSame(_expectedRocket, _actualRocket) ' Tests whether the specified objects both refer to the same object
Assert.AreNotSame(_expectedRocket, _actualRocket) ' Tests whether the specified objects refer to different objects
Assert.IsTrue(_isThereEnoughFuel) ' Tests whether the specified condition is true
Assert.IsFalse(_isThereEnoughFuel) ' Tests whether the specified condition is false
Assert.IsNull(_actualRocket) ' Tests whether the specified object is null
Assert.IsNotNull(_actualRocket) ' Tests whether the specified object is non-null
Assert.IsInstanceOf(_actualRocket, GetType(Falcon9Rocket)) ' Tests whether the specified object is an instance of the expected type
Assert.IsNotInstanceOf(_actualRocket, GetType(Falcon9Rocket)) ' Tests whether the specified object is not an instance of type
StringAssert.AreEqualIgnoringCase(_expectedBellatrixTitle, "Bellatrix") ' Tests whether the specified strings are equal ignoring their casing
StringAssert.Contains(_expectedBellatrixTitle, "Bellatrix") ' Tests whether the specified string contains the specified substring
StringAssert.DoesNotContain(_expectedBellatrixTitle, "Bellatrix") ' Tests whether the specified string doesn't contain the specified substring
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix") ' Tests whether the specified string begins with the specified substring
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix") ' Tests whether the specified string begins with the specified substring
StringAssert.IsMatch("(281)388-0388", "(?d{3})?-? *d{3}-? *-?d{4}") ' Tests whether the specified string matches a regular expression
StringAssert.DoesNotMatch("281)388-0388", "(?d{3})?-? *d{3}-? *-?d{4}") ' Tests whether the specified string does not match a regular expression
CollectionAssert.AreEqual(_expectedRockets, _actualRockets) ' Tests whether the specified collections have the same elements in the same order and quantity.
CollectionAssert.AreNotEqual(_expectedRockets, _actualRockets) ' Tests whether the specified collections does not have the same elements or the elements are in a different order and quantity.
CollectionAssert.AreEquivalent(_expectedRockets, _actualRockets) ' Tests whether two collections contain the same elements.
CollectionAssert.AreNotEquivalent(_expectedRockets, _actualRockets) ' Tests whether two collections contain different elements.
CollectionAssert.AllItemsAreInstancesOfType(_expectedRockets, _actualRockets) ' Tests whether all elements in the specified collection are instances of the expected type
CollectionAssert.AllItemsAreNotNull(_expectedRockets) ' Tests whether all items in the specified collection are non-null
CollectionAssert.AllItemsAreUnique(_expectedRockets) ' Tests whether all items in the specified collection are unique
CollectionAssert.Contains(_actualRockets, falcon9) ' Tests whether the specified collection contains the specified element
CollectionAssert.DoesNotContain(_actualRockets, falcon9) ' Tests whether the specified collection does not contain the specified element
CollectionAssert.IsSubsetOf(_expectedRockets, _actualRockets) ' Tests whether one collection is a subset of another collection
CollectionAssert.IsNotSubsetOf(_expectedRockets, _actualRockets) ' Tests whether one collection is not a subset of another collection
Assert.Throws(Of ArgumentNullException)(Function() New Regex(Nothing)) ' Tests whether the code specified by delegate throws exact given exception of type T
```

Assertions – Constraint Model

```
Assert.That(28, [Is].EqualTo(_actualFuel)) ' Tests whether the specified values are equal.
Assert.That(28, [Is].[Not].EqualTo(_actualFuel)) ' Tests whether the specified values are unequal. Same as AreEqual for numeric values.
Assert.That(_expectedRocket, [Is].SameAs(_actualRocket)) ' Tests whether the specified objects both refer to the same object
Assert.That(_expectedRocket, [Is].[Not].SameAs(_actualRocket)) ' Tests whether the specified objects refer to different objects
Assert.That(_isThereEnoughFuel, [Is].[True]) ' Tests whether the specified condition is true
Assert.That(_isThereEnoughFuel, [Is].[False]) ' Tests whether the specified condition is false
Assert.That(_actualRocket, [Is].Null) ' Tests whether the specified object is null
Assert.That(_actualRocket, [Is].[Not].Null) ' Tests whether the specified object is non-null
Assert.That(_actualRocket, [Is].InstanceOf(Of Falcon9Rocket)()) ' Tests whether the specified object is an instance of the expected type
Assert.That(_actualRocket, [Is].[Not].InstanceOf(Of Falcon9Rocket)()) ' Tests whether the specified object is not an instance of type
Assert.That(_actualFuel, [Is].GreaterThan(20)) ' Tests whether the specified object greater than the specified value
Assert.That(28, [Is].EqualTo(_actualFuel).Within(0.50)) ' Tests whether the specified values are nearly equal within the specified tolerance.
Assert.That(28, [Is].EqualTo(_actualFuel).Within(2).Percent) ' Tests whether the specified values are nearly equal within the specified % tolerance.
Assert.That(_actualRocketParts, Has.Exactly(10).Items) ' Tests whether the specified collection has exactly the stated number of items in it.
Assert.That(_actualRocketParts, [Is].Unique) ' Tests whether the items in the specified collections are unique.
Assert.That(_actualRocketParts, Does.Contain(_expectedRocketPart)) ' Tests whether a given items is present in the specified list of items.
Assert.That(_actualRocketParts, Has.Exactly(1).Matches(Of RocketPart)(Function(part) part.Name Is "Door" AndAlso part.Height Is "200")) ' Tests whether the specified collection has exactly the stated item in it.
```

Author Attribute

```
<TestFixture>
<Author("Joro Doev", "joro.doev@bellatrix.solutions")>
Public Class RocketFuelTests
    <Test>
    Public Sub RocketFuelMeassuredCorrectly_When_Landing()
    End Sub
    <Test>
    <Author("Ivan Penchev")>
    Public Sub RocketFuelMeassuredCorrectly_When_Flying()
    End Sub
End Class
```

Pairwise Attribute

```
<Test, Pairwise>
Public Sub ValidateLandingSiteOfRover_When_GoingToMars(
    <Values("a", "b", "c")> ByVal a As String,
    <Values("+", "-")> ByVal b As String,
    <Values("x", "y")> ByVal c As String)
    Debug.WriteLine("{0} {1} {2}", a, b, c)
End Sub
```

Range Attribute

```
<Test>
Public Sub CalculateJupiterBaseLandingPoint(
    <Values(1, 2, 3)> ByVal x As Integer,
    <Range(0.2, 0.6)> ByVal y As Double)
    '...
End Sub
```

Timeout Attribute

```
<Test, Timeout(2000)>
Public Sub FireRocketToProximaCentauri()
    '...
End Sub
```

Execute Tests in Parallel

```
<Assembly: Parallelizable(ParallelScope.Fixtures)>
<Assembly: LevelOfParallelism(3)>
```

Repeat Attribute

```
<Test>
<Repeat(10)>
Public Sub RocketFuelMeassuredCorrectly_When_Flying()
End Sub
```

Combinatorial Attribute

```
<Test, Combinatorial>
Public Sub CorrectFuelMeassured_When_X_Site(
    <Values(1, 2, 3)> ByVal x As Integer,
    <Values("A", "B")> ByVal s As String)
    '...
End Sub
```

Random Attribute

```
<Test>
Public Sub GenerateRandomLandingSiteOnMoon(
    <Values(1, 2, 3)> ByVal x As Integer,
    <Random(-1.0, 1.0, 5)> ByVal d As Double)
    '...
End Sub
```

Retry Attribute

```
<Test>
<Retry(3)>
Public Sub CalculateJupiterBaseLandingPoint(
    <Values(1, 2, 3)> ByVal x As Integer,
    <Range(0.2, 0.6)> ByVal y As Double)
    '...
End Sub
```

```
<TestFixture>
<Parallelizable(ParallelScope.Fixtures)>
Public Class TestFalcon9EngineLevels
    ' ...
End Class
```