**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

**Answers:**

**Unit Testing**:

* **Definition**: Testing the smallest testable components (e.g., functions, methods) of a program in isolation to ensure they work as expected.

**Difference from Functional Testing**:

* **Scope**: Functional testing validates the entire system or feature against requirements (end-to-end behavior), while unit testing focuses on isolated code units.
* **Dependencies**: Unit testing mocks dependencies; functional testing includes real dependencies (e.g., UI, database).
* **Purpose**: Unit testing ensures code correctness at a granular level; functional testing ensures the system works as intended for users.
* **Example**: Functional testing might verify that a calculator app displays the correct sum when a user inputs 2 + 3.

### 2. Types of Testing

* **Unit Testing**: Tests individual code components in isolation (e.g., a single method).
* **Functional Testing**: Validates the system against functional requirements (e.g., end-to-end workflows).
* **Automated Testing**: Uses scripts/tools to execute tests automatically (e.g., NUnit tests for CalcLibrary).
* **Performance Testing**: Evaluates system speed, scalability, and resource usage under load.

**CODE:**

**Calculator.cs**

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

}

**CalculatorTests.cs**

using NUnit.Framework;

using CalcLibrary;

namespace CalculatorTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void Setup()

{

calc = new Calculator();

}

[TestCase(2, 3, 5)]

[TestCase(10, 0, 10)]

public void Add\_TwoNumbers\_ReturnsSum(int a, int b, int expected)

{

int result = calc.Add(a, b);

Assert.That(result, Is.EqualTo(expected));

}

}

}

**OUTPUT:**

