Question -1: Differentiation between Unit Testing and Functional Testing

| **Feature** | **Unit Testing** | **Functional Testing** |
| --- | --- | --- |
| Focus | Smallest unit (method/class) | End-to-end system functionality |
| Scope | Narrow, internal logic | Broad, user-level flows |
| Dependencies | Mocked/Stubs used | Uses actual components or interfaces |
| Tool Examples | NUnit, xUnit | Selenium, Postman |
| Performed by | Developers | QA/Testers |

Question 2:Types of Testing

* **Unit Testing** – Testing smallest isolated code units (methods/classes)
* **Functional Testing** – Testing application features against requirements
* **Automated Testing** – Using scripts to perform repetitive tests (Unit/UI/API)
* **Performance Testing** – Measuring response time, load, scalability

Question 3:Benefits of Automated Testing

* Faster feedback on code changes
* Prevents regression bugs
* Enables CI/CD pipelines
* Increases confidence in refactoring

Question 4:Loosely Coupled & Testable Design

* A loosely coupled system avoids hard-coded dependencies.
* Use interfaces, dependency injection, mocks.
* Makes your code testable, flexible, and reusable.

For instance,

1. var db = new Database(); **// tightly coupled**
2. public CalculatorService(IDatabase db) {
   1. \_db = db; **// pass dependency externally (mockable)**
   2. }

**UNIT TEST CODE:**

using CalcLibrary;

using NUnit.Framework;

namespace CalcTests

{

[TestFixture] // Marks this class as a test container

public class CalculatorTests

{

private SimpleCalculator calculator;

[SetUp]

public void Init()

{

calculator = new SimpleCalculator(); // Setup runs before each test

}

[TearDown]

public void Cleanup()

{

// calculator.AllClear(); // Cleanup runs after each test

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(-1, -1, -2)]

[TestCase(0, 0, 0)]

[TestCase(1.5, 2.5, 4.0)]

public void Addition\_WithVariousInputs\_ReturnsCorrectResult(double a, double b, double expected)

{

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

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

}

[Test]

[Ignore("This is a sample ignored test.")]

public void Ignored\_Test\_Sample()

{

// This will not be executed

Assert.Fail("This test should have been ignored.");

}

} }

**TEST CASE RUN:**

