**Unit Testing Objectives :**

**1. Meaning of Unit Testing -**  
- Tests the smallest parts of an application in isolation (e.g., methods).  
- Written by developers using frameworks like NUnit.  
  
**Difference between Unit Testing & Functional Testing -**  
- Unit tests are fast, isolated, and code-focused.  
- Functional tests validate behavior through the full application flow.

**2. Types of Testing -**

- Unit Testing: Test individual units or components.  
- Functional Testing: Verify the system behaves as expected.  
- Automated Testing: Uses scripts to execute tests automatically.  
- Performance Testing: Evaluates responsiveness under load.

**3. Benefits of Automated Testing -**

- Provides fast feedback in development.  
- Detects bugs early.  
- Enables continuous integration.  
- Saves time compared to manual testing.

**4. Loosely Coupled & Testable Design -**

A loosely coupled system is one where components are independent and minimally rely on each other. This makes your code flexible, easier to change, and easier to test.

A testable design is a code structure that allows developers to write automated tests easily, especially unit tests.

Example:  
public interface IDataService  
{  
 string GetData();  
}  
  
public class BusinessLogic  
{  
 private readonly IDataService \_service;  
  
 public BusinessLogic(IDataService service)  
 {  
 \_service = service;  
 }  
  
 public string Process() => \_service.GetData();  
}

**5. Unit Test: Calculator Addition (NUnit) -**

[TestFixture]  
public class CalculatorTests  
{  
 [Test]  
 public void Add\_TwoNumbers\_ReturnsSum()  
 {  
 var calc = new Calculator();  
 var result = calc.Add(2, 3);  
 Assert.AreEqual(5, result);  
 }  
}

**6. [SetUp], [TearDown], and [Ignore] -**

- [SetUp]: Runs before every test method.  
- [TearDown]: Runs after every test method.  
- [Ignore]: Skips a test.  
  
Example:  
[TestFixture]  
public class SampleTests  
{  
 private Calculator calc;  
  
 [SetUp]  
 public void Init() => calc = new Calculator();  
  
 [TearDown]  
 public void Cleanup() => calc = null;  
  
 [Test]  
 [Ignore("This test is temporarily disabled")]  
 public void TestToIgnore() { }  
}

**7. Parameterized Tests with [TestCase] -**

Example:  
[TestFixture]  
public class CalculatorTests  
{  
 [TestCase(2, 3, 5)]  
 [TestCase(0, 0, 0)]  
 [TestCase(-1, 1, 0)]  
 public void Add\_TestCases(int a, int b, int expected)  
 {  
 var calc = new Calculator();  
 var result = calc.Add(a, b);  
 Assert.AreEqual(expected, result);  
 }  
}