1. What is Unit Testing? How is it Different from Functional Testing?

Unit testing is like checking a small piece of your machine—say just the calculator's "add" button—to make sure it works exactly right. You test one method or function at a time.

Functional testing, on the other hand, checks whether the entire machine works—like testing if you can actually do a full calculation (add, subtract, get results) using the UI.

Unit = small piece

Functional = full user experience.  
  
  
2. Types of Testing

Unit Testing – Test small, individual methods.

Functional Testing – Test whether full features or business flows work.

Automated Testing – Let the computer run your tests without manual effort.

Performance Testing – See how fast your code runs under pressure or heavy load.  
  
  
 Why Automated Testing Helps

Imagine you have to test your app every time you change something. Manually? That’s tiring. Automated testing does it instantly, reliably, and every time you change the code, which saves time and avoids human error.  
  
What is Loosely Coupled & Testable Design?

A loosely coupled design means your code doesn't overly depend on other classes or data.

For example:

Bad: Calculator depends on a Database to get numbers.

Good: Calculator takes inputs from arguments, not from other classes.

This makes it easier to test, because you don’t need everything to run—just the piece you want to test.

1. Write Your First Unit Test – Calculator Add Operation

public class CalculatorTests

{

[Test] // Marks this method as a test

public void Add\_ReturnsCorrectSum()

{

var calc = new Calculator();

Assert.That(calc.Add(2, 3), Is.EqualTo(5));

}

}

Ques.Why Use [SetUp], [TearDown], [Ignore]?

[SetUp]: Runs before each test. Good for setting up your calculator.

[TearDown]: Runs after each test. Good for cleanup/reset.

[Ignore]: Temporarily skip a test without deleting it.

Code:

[SetUp]

public void Init() => calc = new Calculator();

[TearDown]

public void Cleanup() => calc = null;

[Test, Ignore("Not ready yet")]

public void Subtract\_WillBeImplementedLater() {}

Ques. Why Use Parameterized Test Cases?

[TestCase(2, 3, 5)]

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

[TestCase(0, 0, 0)]

public void Add\_WorksForMultipleInputs(int a, int b, int expected)

{

Assert.That(calc.Add(a, b), Is.EqualTo(expected));

}