NUnit Testing

Explain the meaning of Unit testing and its difference on comparison with Functional testing

| **Unit Testing** | **Functional Testing** |
| --- | --- |
| Tests individual units (methods/classes) in isolation | Tests end-to-end user workflows |
| Uses mocks to isolate dependencies (e.g., fake databases) | Tests integrated components with real dependencies |
| Fast execution (milliseconds per test) | Slower execution (seconds/minutes per test) |
| Developer-focused (white-box) | QA/Business-focused (black-box) |
| Example: Testing a Calculator.Add() method | Example: Testing "user login → add item to cart → checkout" |

List various types of testing

| **Type** | **Purpose** |
| --- | --- |
| Unit Testing | Validate individual code units |
| Functional Testing | Verify feature requirements |
| Automated Testing | Scripted tests for repeatability |
| Performance Testing | Measure speed/scalability under load |

Understand the benefit of automated testing

* Faster feedback during development
* Reusable tests for regression testing
* Higher accuracy vs manual testing
* Enables CI/CD pipelines (auto-test on every code change)
* Cost-effective long-term

Explain what is loosly coupled & testable design

public class OrderProcessor {

private IDatabase \_db;

public OrderProcessor(IDatabase db) => \_db = db;

public void Process() => \_db.Save();

}

Write your first testing program to validate a calculator addition operation

[TestCase(2, 2, 4)]

[TestCase(5, 3, 8)]

public void AddTest(int a, int b, int expected) { ... }

Understand the need of [SetUp], [TearDown] & [Ignore] attributes.

| **Attribute** | **Purpose** |
| --- | --- |
| [SetUp] | Run code BEFORE each test (e.g., initialize objects) |
| [TearDown] | Run code AFTER each test (e.g., clean up resources) |
| [Ignore] | Temporarily disable a test (e.g., [Ignore("WIP")]) |

TESTING OUTPUT

