NUnit Hands-on Theory Concepts

# What is Unit Testing?

Unit testing involves testing the smallest individual unit of code — typically a method or function — in isolation from the rest of the system.  
  
Unit tests are:  
- Fast  
- Focused on logic  
- Can use mocking to isolate dependencies

# What is Functional Testing?

Functional testing validates the system as a whole against business requirements. It checks if the system behaves correctly with real inputs.  
  
Key Difference:  
- Unit testing checks internal correctness of code units.  
- Functional testing checks external behavior and user-facing features.  
  
Unit testing often uses mocking for:  
- Simulating databases, file systems, or APIs  
- Isolating the method under test from external dependencies

# Types of Testing:

1. Unit Testing:  
 - Tests individual methods or classes in isolation.  
 - Uses NUnit, xUnit, MSTest frameworks.  
  
2. Functional Testing:  
 - Tests if the software behaves as expected.  
 - Done using tools like Selenium, Postman.  
  
3. Automated Testing:  
 - Tests are executed automatically using scripts/tools.  
 - Faster and more reliable than manual testing.  
  
4. Performance Testing:  
 - Measures system performance under load.  
 - Checks response time, throughput, resource usage.

# What is a Loosely Coupled & Testable Design?

- Loosely coupled design means components do not depend heavily on each other.  
- It makes the code easier to test, maintain, and extend.  
  
Example of Tight Coupling (Bad):  
public class ReportGenerator {  
 private Database db = new Database(); // tightly coupled  
}  
  
Example of Loose Coupling (Good):  
public class ReportGenerator {  
 private readonly IDatabase db;  
 public ReportGenerator(IDatabase db) {  
 this.db = db;  
 }  
}  
  
Here, we can pass a mock IDatabase while testing without touching the real DB.

# Understanding [SetUp], [TearDown], and [Ignore]:

[SetUp] → Runs before every test method. Used to initialize resources.  
[TearDown] → Runs after every test method. Used for cleanup (e.g., reset variables).  
[Ignore] → Temporarily skips a test. Useful when a test is incomplete or under review.

# Benefits of Parameterized Test Cases:

- Reduce code duplication.  
- Test multiple inputs and expected results using a single method.  
- Easy to expand and maintain.

# What is Mocking? (Using Moq Framework):

- Mocking is the act of simulating the behavior of real objects in testing.  
- Especially useful for dependencies like:  
 - Databases  
 - File systems  
 - Web APIs  
- It helps in isolating the code being tested.  
  
Moq Example:  
var mockRepo = new Mock<IUserRepository>();  
mockRepo.Setup(repo => repo.GetUser("admin")).Returns(new User { Name = "admin" });  
  
Benefits:  
- Test in isolation without needing real DB/API  
- Control return values for edge cases  
- Improve speed and stability of tests

# Why Testing Private Methods is Not Beneficial:

Unit testing should focus on the public interface of a class.  
Private methods are implementation details — they may change frequently even if the public behavior stays the same.  
  
Instead of testing private methods directly, test the public methods that use them. This ensures that your tests stay stable and focused on actual system behavior.