**Q1. EXPLAIN THE MEANING OF UNIT TESTING AND ITS DIFFERENCE ON COMPARISON WITH FUNCTIONAL TESTING.**

**Solution:**

**Unit Testing is a type of software testing where individual components or functions (called *units*) of a program are tested in isolation to ensure that they work as expected.**

* **It focuses on a single function, method, or class.**
* **Usually written and run by developers.**
* **Helps catch bugs early in development.**
* **Often automated using frameworks (e.g., JUnit for Java, NUnit for C#, PyTest for Python).**

**Functional Testing is a type of testing that validates the entire application or feature against the business requirements.**

* **It focuses on what the system does rather than how it does it.**
* **Ensures that the software behaves correctly from the user's perspective.**
* **Usually conducted by QA testers.**
* **It can be done manually or via automation (e.g., Selenium).**

**Example:  
Testing a login feature by:**

* **Entering a username and password.**
* **Clicking "Login".**
* **Verifying that the user is redirected to the dashboard.**
* **Difference Between Unit Testing and Functional Testing**

| **Feature** | **Unit Testing** | **Functional Testing** |
| --- | --- | --- |
| **Scope** | Individual components (methods/functions) | Entire application features or modules |
| **Focus** | Internal logic | User-facing behavior |
| **Performed by** | Developers | Testers (or QA engineers) |
| **Level** | Low-level (code level) | High-level (system or acceptance level) |
| **Tools** | NUnit, JUnit, PyTest, xUnit | Selenium, QTP, Postman, etc. |
| **Speed** | Very fast | Slower (involves full flow) |
| **Dependency** | No or few external dependencies | Works with full system and all dependencies |
| **Automation** | Common and easy | Possible but more complex |

**Q2. LIST VARIOUS TYPES OF TESTING**

· **Unit Testing** – Tests individual methods or components.

· **Functional Testing** – Ensures the software meets functional requirements.

· **Automated Testing** – Uses scripts/tools to automatically run tests (can be unit or functional).

· **Performance Testing** – Checks how the system performs under load, stress, or scalability scenarios.

**Q3. UNDERSTAND THE BENEFIT OF AUTOMATED TESTING**

### **Benefits of Automated Testing**

* **Repeatability**: Run tests frequently and consistently.
* **Speed**: Faster than manual testing.
* **Reliability**: Reduces human error.
* **Coverage**: Helps test more scenarios in less time.
* **Cost-effective**: Saves effort and cost in long-term maintenance.
* **CI/CD Support**: Essential for DevOps and continuous delivery pipelines.

**Q4. EXPLAIN WHAT IS LOOSELY COUPLED & TESTABLE DESIGN**

**Loosely Coupled Design**:

* Classes/components are **independent** and interact via **interfaces or abstractions**, not directly relying on specific implementations.
* Makes code **easier to test, maintain, and extend**.

**Testable Design**:

* Code written with **testability in mind** allows for easy injection of test doubles (mocks/stubs).

**CODE:**

using CalcLibrary;

namespace StudentGrades.nUnitTests

{

[TestFixture]

public class GradeCalculatorTests

{

private SimpleCalculator calculator;

[SetUp]

public void Setup()

{

calculator = new SimpleCalculator();

}

public void TearDown()

{

calculator = null;

}

[TestCase(50, 30,80)]

[TestCase(10, 20, 30)]

[TestCase(100, 14, 114)]

[TestCase(20, 30, 50)]

public void Addition\_EqualTest(double a, double b, double expected)

{

var actual = calculator.Addition(a, b);

Assert.AreEqual(expected, actual);

}

[TestCase(22,22,11)]

[TestCase(70, 10, 100)]

[TestCase(120, 1, 122)]

[TestCase(11, 10, 90)]

public void Addition\_NotEqualTest(double a, double b, double Notexpected)

{

var actual = calculator.Addition(a, b);

Assert.AreNotEqual(Notexpected, actual);

}

}

}

