Introduction to Unit & Integration Testing

Unit testing

Low-level tests that isolate and test smallest testable units of code, such as functions or class methods

Simple example

```
def feet_to_inches(feet):
    return feet * 12

def test_feet_to_inches():
    assert feet_to_inches(2) == 24
```

Developers typically write unit tests for their own code, continuously
Unit tests should be small, fast, easily automated as part of CI/CD pipeline
Should not rely on external dependencies, e.g. databases, local filesystem, external APIs

Integration testing

Tests integration of various software components

More complex, can have dependencies such as databases, filesystems, etc

May be done by developers of code under test, or other testers

Team may take bottom-up approach (unit tests then integration), a top-down approach (integration then unit tests), or a combination

AAA pattern in testing

Arrange – Set up the test in a consistent way

Act – Run the code under test

Assert – Compare results from Act against expected results & pass or fail the test

When writing a test, we should know in advance what results will mean pass or fail!

Other types of tests

Smoke tests – Do basic features work as expected?

Regression tests – Did the update cause any other existing code to break?

Acceptance tests – Does the software meet the business requirements?

System tests – Do all parts of the software work together as a whole system?

Performance tests – How does the software perform under different conditions?

Load tests – How does the software perform under heavy loads?

& others...