WHAT IS UNIT TESTING?

In this lecture we're going to start the course on Test Driven Development by defining what unit testing actually is. We'll go over some of the common types of software testing, look more closely at the specifics of unit testing, and review a simple example.

WHY DO WE UNIT TEST?

- Software bugs hurt the business!
- Software testing catches the bugs before they get to the field.
- Need several levels of safety nets

So why do we go through the effort of writing and running unit tests?
<click> Because software bugs can hurt the business! We don't want any bugs making it out to the field for our customers to see as that can hurt our reputation and cause customers to look at using other products.
<click> Software testing attempts to address this problem by catching any bugs in the software before they get to the field. <click> This is done systematically with a multi-layered approach where each layer of testing provides a safety net for catching bugs before they get to the field.

LEVELS OF TESTING

- **Unit Testing** - Testing at the function level.
- **Component Testing** - Testing is at the library and compiled binary level
- **System Testing** - Tests the external interfaces of a system which is a collection of sub-systems.
- **Performance Testing** - Testing done at sub-system and system levels to verify timing and resource usages are acceptable.

There are several levels of testing which provide the layers of safety nets for catching any bugs that might be in the code.

- The lowest level are Unit Tests. Unit tests validate individual functions in the production code. Unit tests are generally the most comprehensive tests which should test all positive and negative test cases for a function.
- Next comes component level testing which tests the external interfaces for individual components. Components are essentially a collection of the functions.
- Then comes system level testing which tests the eternal interfaces at a system level. Systems can be collections of component or of sub-systems.
- Lastly comes performance testing, which tests systems and sub-systems at expected production loads to verify that response times and resource utilization (i.e. memory, CPU, disk usage) are acceptable.

UNIT TESTING SPECIFICS

- Tests individual functions
- A test should be written for each test case for a function (all positive and negative test cases).
- Groups of tests can be combined into test suites for better organization.
- Executes in the development environment rather than the production environment.
- Execution of the tests should be automated

Now lets look at some specifics on unit testing.

- Unit testing tests individual functions in the code.
- Each test case for the function should have a corresponding unit test.
- Groups of unit tests can be combined into test suites which can help with organizing the tests.
- The unit tests should execute in your development environment rather than the production environment. This is important to ensure you can run them easily and often.
- And lastly the unit tests should be implemented in an automated fashion. You should be able to click a button and the unit tests will build and execute.

A Simple Example

```
import pytest

# Production Code
def str_len( theStr ):
    return len(theStr)

# A Unit Test
def test_string_length():
    testStr = “1”                    // Setup
    result = str_len(testStr)        // Action
    assert result == 1               // Assert
```

Here we have a very simple example showing a unit test for a bit of production code. The production code is a function that returns the length of a passed in string. The unit test is a single positive test case that verifies a length of 1 is returned for a string with one character in it. The “test_str_len” call is the unit test for the str_len production code. The unit test performs three steps:

- A setup step where it creates the test string.
- An action step where it calls the production code to perform the action that is under test.
- An assertion step where the test validates the results of the action.

This is a common structure that all of your unit tests should follow.

SUMMARY

- Unit tests are the first safety net for catching bugs before they get to the field.
- Unit tests validate test cases for individual functions.
- They should build and run in the developer's development environment.
- Unit tests should run fast!

To summarize, here are the key points on unit testing:
<click> Unit tests are our first safety net for catching bugs in the production code
<click> Unit tests validate test cases for individual functions
<click> They should build and run in the development environment
<click> Unit tests should run fast! We ideally want a developer re-running the unit tests every 3-5 minutes and this can be difficult with a slow build processes or if any of the tests run slow (i.e. more than a few seconds).