Unit Testing

What is Unit Testing?

A method of testing individual units of source code to determine whether or not the behavior of the code is according to spec.

Why UnitTest?

- Find problems early
- Encourages refactoring
- Serves as documentation
- Provides confidence in code

Test Driven Development

A software development process wherein requirements are turned into very specific test cases, and then the software is written and improved upon with the goal of passing the tests.

Test Driven Development Process

1. Add a test
2. Run all current tests to see if new test fails
3. Write the code relevant to the test
4. Run tests
5. Refactor code

Repeat for new functionality

Add A Test

New feature development begins with writing the test for the new feature. The test is written in such a way that it defines the feature or improvement being made, bug being fixed, etc.

This is the differentiating feature of TDD versus simply using unit testing.

The benefits are that the developer focuses on the requirements of the feature or bug fix before writing any code.

Run Current Tests/New Test Fails

Newly added tests should fail for an expected reason because the feature has not yet been developed.

If the test passes, it could mean that the feature is already implemented. Or else, the requirement could be misunderstood or the test itself could be flawed.

New test failure also rules out false confidence in a poorly written test that always passes.

Write Code

Code is now written that will cause the test to pass. At this stage, the code written may be poor quality or inelegant. This is fine as it is part of the process.

Run Tests

At this stage, all previously written tests should continue to pass, and the newly added test specific to the feature under development should pass as well.

This is done to ensure confidence in the quality of the code because it is now proven that the newly added functionality has not interfered with previously implemented solutions.

If the old tests do not pass, it is up to the developer to figure out where the problem has been introduced.

Unit "Test"

```
function aNumber () {
  return 76;
}

function multFunc (fn) {
  return fn() * 10;
}

// console.log(multFunc(aNumber));

function testMultFunction() {
  var valueToTest;

  function testNumber() {
    return 32;
  }

  valueToTest = multFunc(testNumber);

  if (valueToTest === 320) {
    console.log('multFunc test passed');
  } else {
    console.log('multFunc test failed');
  }
}

testMultFunction();
```

Refactor Code

Make your code better, reduce repeated code by isolating sub-tasks into their own functions, ensure variable and method naming is clear.

As always, run tests after refactoring to ensure everything still works.