Week 6 - Lecture Notes

Welcome to Week 6!

This week we will go over an important concept that transcends specific programming languages, and that is unit testing. You should be familiar with testing from the previous IS210 class, where tests were being run against your code every time you made a commit to your git repository for an assignment.

First off, lets define what we mean by unit testing. A unit refers to the small testable parts of an application. Unit testing is simply the process by which we individually and independently verify the unit for proper operation. How you define the exact boundaries of what a 'unit' entails depends on many factors, like how your application is structured and what the purpose of the application is. A unit could be an entire module, but it is more commonly an individual function or procedure. When dealing with OOP, a unit can be an entire class or the class' individual methods. However, at the end of the day, a unit is however you define it.

Writing good tests for your units of code, however defined, is a bit of an inexact science. Good unit tests at the end of the day are verifying that your code does what it should be doing; obviously, this insinuates that you have a deep understanding of the *requirements* of your code to begin with. If you do not have a clear grasp of what your code should be doing, you should not be writing tests yet, and focus on defining the purpose of the unit in question. There is a lot of information on how to develop good, informative software requirements, which this course will not cover. If you are interested in this, however, you can check out these resources:

- Software Requirements, 3rd Edition
- Code Complete, 2nd Edition

There are other kinds of testing as well, besides unit testing, including:

- **Regression testing** - tests that ensure that bugs that have been found and fixed do not come back in future releases of the code
- **Software performance testing** - tests that are used to determine how some code performs in terms of speed, responsiveness or usage of resources.
- **Security testing** - tests that help determine whether or not the code under review is vulnerable to security issues, like critical data being exposed. What is interesting about security testing is that passing a security test does not mean that there are *no* security flaws at all. It only means that the code is not vulnerable to the flaws you are testing.

This week we will focus on how we can write theses tests ourselves by reviewing a practical application from "Dive In Python". The reading will cover code that deals with handling Roman

Numerals and how to approach testing and verifying that this code does what we need it to. The reading will also cover an idea called *refactoring*, which is the process by which you take already working code and make it better in some form. Of course, its tough to refactor code and ensure that you have not introduced bugs without testing, which is one of the main benefits of testing your code to begin with.