Overview

Bookmark this page

Advanced Testing

Introduction

This lesson's topics step beyond *programming* into a central question of *software engineering*: how do you manage code that is too big to fit inside your own head?

Up until now, most of the programs that you've written in Python have probably been 200 lines or fewer, and you wrote them on your own. A single programmer can completely understand a program this small. As programs get bigger and you begin working in teams, it becomes difficult to understand or remember how the changes you made on line 523 of your program will effect the operation of code written on lines 10, 200, or 2000.

There are tools that can help us manage the complexity of a large codebase. The tools that we'll explore for this lesson are testing, linting, and flaking. While exploring them, we'll introduce a software design pattern that makes it easier to write maintainable code: dependency injection.

Learning Objectives

Upon successful completion of this lesson, you will be able to:

- Use dependency injection and mocking to create easily testable code.
- Differentiate between unit testing and acceptance testing.
- Create a file for your project that defines a code style standard, and run an automatic analysis of your code to test whether it conforms to that standard.
- Automatically identify areas of your code that might be difficult to maintain or test.

New Words, Concepts, and Tools

- unittest
- unittest.TestCase
- dependency injection
- TestCase.setUp
- mock
- mock.MagicMock
- flake8
- pylint
- coverage

Prerequisites

To be successful in this lesson you will need to be very familiar with the following concepts, covered earlier in the class:

1. Modules and absolute / relative imports
2. Concepts of unit and functional tests

Before you Start

Read the following articles and watch a short video in preparation for this lesson.

1. Managing Software Complexity
2. Technical Debt: this is a wiki page for discussions on the types of problems we'll be addressing.
3. Test-driven development: Write better code in less time by Evan Dorn: Watch ONLY the first 3 minutes and 44 seconds of the video, STOP at 3:44! After 3:44 it introduces Test Driven Development which we will circle back to at the end of the lesson but will confuse our discussion right now.

Suggested Workflow

- Explore the "Before you Start" readings and video

- Work through the lesson content pages
- Watch the required videos
- Do the practice activity
- Submit your assignment

At the End of the Lesson