FIT4004 Assignment 3

Automated Unit Testing and Continuous Integration

15% of your final mark
Incremental submission - read the instructions *carefully*!

Parts of this assignments will be completed in pairs

In this assignment, you will create a small module in Python, and test it using `unittest` and `mock`. You will also set up continuous integration, so that testing occurs as you update your source code repository.

Typically, developers are expected to write unit tests for their own code, hence the need to write your own test suite. However, you are *not* going to be marked on the quality of your implementation - you just have to have one to test against. Comprehensive unit tests and fixing bugs that you find as you commit changes should result in a module that is as close as possible to functionally correct.

Github Accounts

In the unlikely event you don't already have one, please create a free account on GitHub (github.com).

Groups and repositories

Parts of this assignment will be done in groups of two (or, if there are uneven numbers and there is no alternative, groups of three).

Register a team name, list the members and github usernames on the form on Moodle. I will create repositories for you; you'll receive an emailed invitation from GitHub which you'll need to accept to access the repository.

The Module (written in pairs)

You will implement a Python program that analyses "tweets" (posts on Twitter) made by an individual, produces a graph showing frequency of tweeting at particular times of the day, and posts the graph as a tweet. Create a Twitter account for the purpose if you don't have one (or you want to keep your graphs separate from your other tweets).

Your program should run from the command line and take the following command-line arguments:

`-t TIMEZONE`: the timezone of the tweeter for analysis purposes. TIMEZONE should be represented as +/- HH:MM (that is, if a tweet occured at 13:45 UTC, and their timezone was +10:00, the tweet is regarded as having occured at 23:45). The default is UTC.
`-a DATE`: the date (in local time as specified by the -t argument, if present) to begin analysis from. Specified as YYYY-MM-DD.
`-b DATE`: the last date (in local time as specified by the -t argument, if present) to analyse. Specified as YYYY-MM-DD.
`-i ID`: The id of the tweeter to analyse. ID should start with the @ sign.

The graph should be a simple line graph showing the time of day on the X-Axis and the average number of tweets per hour on the Y-Axis.

To access the Twitter API, you can use the Tweepy library.

http://tweepy.readthedocs.io/en/v3.5.0/

If you need to look, the Twitter API itself is documented at
https://dev.twitter.com/docs

You can use matplotlib to produce charts in Python:

https://matplotlib.org/

This unit is on testing, not Python development; the reason you are writing this program is to have something to test. You will not be marked on the Python code other than for functional completeness and basic readability; however, designing for testability will make it easier to get a high mark on the testing aspects of the assignment!

YOU MUST INCLUDE INSTRUCTIONS ON HOW TO BUILD AND USE YOUR PROGRAM, including how to put in a Twitter OAuth API access key. Put it in the Readme.md file in the root directory of your repository.

You can develop your Python program on any operating system you choose; this assignment shouldn't be particularly OS-sensitive. However, so that I can conduct some manual "exploratory testing" on your completed program, you should ensure your program can run under Ubuntu 16.04 LTS. If you want to confirm that it runs in this environment and don't already have access to an instance, you can run it in a Virtualbox (or VMWare, if you have it) VM, or create an instance on the free tier of Amazon AWS.

Your test strategy

Your test plan should be simple. You only need to test the functional correctness of the module and run the tests using the unittest module, performing continuous integration using drone.io.

You must "mock out" anything that accesses a network using the `mock` module.

If the contents of the calls to network access methods are things that you should check to ensure the module is working correctly, you should use the facilities within `mock` to check their correctness. However, you should do this *after* getting your tests working - this may be time consuming and is less important than getting a good set of tests up and running.

The strategy you use to select your test cases is up to you. You should explicitly document your test case selection strategy in a plain text (or Markdown) file called "`test-strategy.txt`" or "`test-strategy.md`" and add it to the repository.

If you want to measure statement or branch coverage as part of your test strategy, there is a standard Python tool called `coverage.py` that can help you achieve this:

http://coverage.readthedocs.io/en/coverage-4.0.3/

Unit Tests (pairs)

Based on your test strategy, devise a test suite for the module.

You should use your unit tests to fix bugs in your module.

Repository and Continuous Integration (pairs)

You must set up and use Travis CI as your continuous integration server. Every time you commit to the master branch, the (appropriately mocked) tests should be run on Travis CI.

As Monash students, if your GitHub account is registered as part of their education program, you get a free private account on Travis CI (https://education.travis-ci.com/).

Test analysis (individual)

As in previous assignments, after completing the testing process, write a short report (no longer than two pages), about the test process, including:

- What bugs, if any, did you find with your unit tests? Please be brief.
- Did you think your testing strategy resulted in adequate tests? Please explain why or why not.
- Do you think it works better for the programmer responsible for code to write the unit tests for it, or for somebody else to do it? Explain why.
- How long did writing mock code take? Was it a major component of the overall effort?
- What would you do differently next time if faced with a similar task?

Marking criteria

- Completion of module according to spec (minor consideration - remember, you are not being marked on the quality of the implementation).
- Reasonableness of test case selection strategy.
- Quality of test cases (including quality of test case code).
- Effectiveness of mock usage.
- Appropriate usage of CI system.
- Quality of test analysis.
- Readability of test analysis.

Submission

Submission will be performed by uploading all items to your repositories.

Your completed module and unit tests should be submitted by Thursday 25th May at 5PM.

Your test analysis report should be uploaded by Friday 26th May at 11:55 PM (that's as late as I can make it).

Plagiarism

You're in fourth year. You know the drill by now.

Special consideration

If a student faces exceptional circumstances (serious illness or injury, family emergency etc) that prevent them completing the assignment, they may apply for special consideration according to the policy and procedure on the Faculty website:

http://www.infotech.monash.edu.au/resources/student/equity/special-consideration.html

Other late submissions

A penalty of 10% per working day will apply to submissions after the deadline.

Assignment forum

You can ask questions on the discussion forum, which the chief examiner will monitor and respond to daily (and at least once on weekends). All students should monitor this forum - I may clarify aspects of the assignment on the forum and any such clarifications are considered "official". You may of course also email the chief examiner, or arrange to see them in person if you prefer.