Assignment 1

Jacob Mason
@JacobAMason
jm2232

Blake Holifield
@BlakeHolifield
bah379

Eli Davis
@EliCDavis
ecd157

Ricky Arendall
@RickArendall
ra529

Lessons Learned

One of the lessons learned was about project setup. If we put a lot of time into correct project organization and setting up automated tools, it makes the TDD portions of our project much smoother. In addition, using Pycharm to automatically run unit tests when we stop typing ensured that we were never introducing defects as well as effectively kept us in the TDD mindset. As far as difficulties, parts of the project did not seem to lend themselves to the specific style of development inherent to TDD. In particular, the distance formula was just one line of functional code that was abundantly clear from the start, making the ethos of "red-green-refactor" feel like a waste of time in this particular case due to the incredibly small scale of that module. We would certainly choose to apply TDD to future projects. Some of us have used TDD before working on this project and find working in a codebase that has been developed in this manner to be preferable.

The few drawbacks we found with TDD were caused by unfamiliarity with the tools used in TDD. A few of our group members had never used Tox, coveralls, or flake8 before. Setting up the tox configs on each developer's machine was simple enough, but did require some assistance. Once all the config issues had been resolved, we found that TDD was incredibly effective. The initial overhead of setup and configuration was not particularly painful and would be even less so should we need to do another project with a duplicate environment. We feel that the massive gains from our testing environment and TDD far outweigh the small drawbacks of setup.

Organizing tests into "test classes" helped keep tests organized for a particular python module. This was especially useful when writing the command line interface since it acts as an interface to 4 other python modules written for this assignment, so a test class was written for each module the CLI interfaced with. One highlight of this kind of development was once all these tests written it was easy to see what all broke in the package whenever something changed such as a renaming of a function.

Naming and Organizational Conventions

We used Python to implement this assignment. Our assignment is in a package folder called "Assignment1" and our file naming and organization roughly follows Jeff Knupp's excellent blog post here: https://jeffknupp.com/blog/2013/08/16/open-sourcing-a-python-project-the-right-way/. Summarily, we strictly follow pep8 conventions and enforce this by using flake8, a program that will check for pep8 violations and other snafus. Our folder structure consists of the Assignment root folder "Assignment1", which contains __init__.py making it a package. This packages contains the source code and a "tests" folder, which contains all unit tests. Python files are named as the camel-case version of the primary class the file contains. Unit tests files are named the same as the source Python file the test, with the exception that they are prefixed with "test_". We use tox to run our unit tests, so knowledge of running pytest is not required. All requirements for testing are found in test_requirements.txt, and tox will read this file and ensure that the test environment is installed appropriately. Running all unit tests and flake8 can be done simply by running `tox` in the root folder (the parent of “Assignment1”).

Inside our tests, because we are using py.test, suites are not used because there exists a test file for each source file. This is efficient for finding the appropriate test without filling files with boilerplate. We use simple py.test fixtures to remove setup duplication.

GitHub

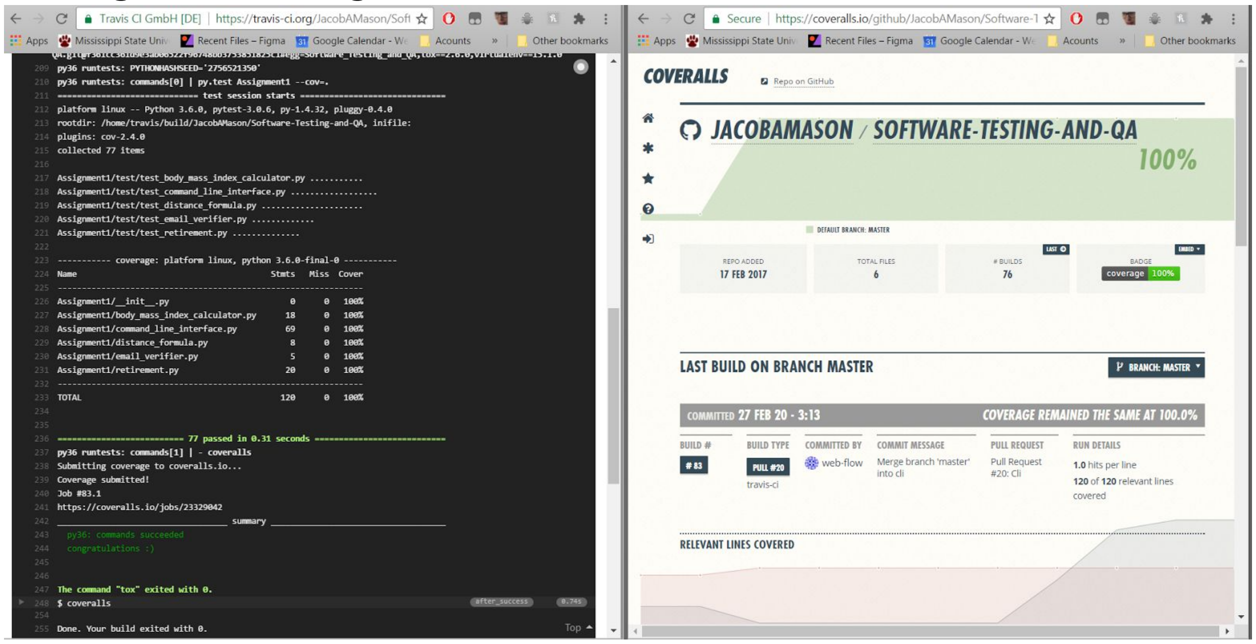
Our GitHub repository can be found at https://github.com/JacobAMason/Software-Testing-and-QA/. We used GitHub’s issue tracker to create Milestones and to track the production and acceptance of features to meet the assignment requirements.

Our GitHub usernames are as follows:
Jacob Mason - @JacobAMason
Blake Holifield - @BlakeHolifield
Eli Davis - @EliCDavis
Ricky Arendall - @RickArendall

Setup And Use

To download an Assignment, just clone the repository or download as a zip. You will need Python, specifically Python 3.6. This project should run on any system officially supported by Python 3.6. Requirements for running the assignments are found in `requirements.txt`. Requirements for running unit tests are found in `test_requirements.txt`. To install requirements, run `pip install -r test_requirements.txt` and `pip install -r requirements.txt`. This can be done from a virtual environment, if you want to keep this project's dependencies separate from your system Python.
Unit tests are managed by tox. To run them, simply issue the command `tox` from your system's terminal in the root project directory. This command will complete with coverage information printed to the terminal.

Testing and Coverage



```
py36 runtests: PYTHONHASHSEED='2756521350'
py36 runtests: commands[0] | py.test Assignment1 --cov=.
============================ test session starts =============================
platform linux -- Python 3.6.0, pytest-3.0.6, py-1.4.32, pluggy-0.4.0
rootdir: /home/travis/build/JacobAMason/Software-Testing-and-QA, inifile:
plugins: cov-2.4.0
collected 77 items

Assignment1/test/test_body_mass_index_calculator.py ...........
Assignment1/test/test_command_line_interface.py ..................
Assignment1/test/test_distance_formula.py .....................
Assignment1/test/test_email_verifier.py .............
Assignment1/test/test_retirement.py ..............

----------- coverage: platform linux, python 3.6.0-final-0 -----------
Name                                         Stmts   Miss  Cover
-----------------------------------------------------------------
Assignment1/__init__.py                          0      0   100%
Assignment1/body_mass_index_calculator.py       18      0   100%
Assignment1/command_line_interface.py           69      0   100%
Assignment1/distance_formula.py                  8      0   100%
Assignment1/email_verifier.py                    5      0   100%
Assignment1/retirement.py                       20      0   100%
-----------------------------------------------------------------
TOTAL                                          120      0   100%


========================== 77 passed in 0.31 seconds ==========================
py36 runtests: commands[1] | - coveralls
Submitting coverage to coveralls.io...
Coverage submitted!
Job #83.1
https://coveralls.io/jobs/23329042
___________________________________ summary ___________________________________
  py36: commands succeeded
  congratulations :)


The command "tox" exited with 0.
$ coveralls                                                   after_success   0.74s

Done. Your build exited with 0.
```

COVERALLS

JACOBAMASON / SOFTWARE-TESTING-AND-QA 100%

DEFAULT BRANCH: MASTER

REPO ADDED	TOTAL FILES	# BUILDS	BADGE
17 FEB 2017	6	76	coverage 100%

LAST BUILD ON BRANCH MASTER

COMMITTED 27 FEB 20 - 3:13 COVERAGE REMAINED THE SAME AT 100.0%

BUILD #	BUILD TYPE	COMMITTED BY	COMMIT MESSAGE	PULL REQUEST	RUN DETAILS
# 83	PULL #20 travis-ci	web-flow	Merge branch 'master' into cli	Pull Request #20: Cli	1.0 hits per line 120 of 120 relevant lines covered

RELEVANT LINES COVERED

We used TravisCI for unit tests and Coveralls for coverage reporting. In addition to running unit tests, we run flake8 which examines our source code for pep8 violations and other simple code smells such as unused variable names. Coveralls also enforces a code coverage threshold, 100% in our case, which will fail a commit that lowers coverage.

Our TravisCI reports can be found here:
https://travis-ci.org/JacobAMason/Software-Testing-and-QA
Our Coveralls reports can be found here:
https://coveralls.io/github/JacobAMason/Software-Testing-and-QA

TravisCI and Coveralls run for every commit pushed to the repository and also run for Pull Request merges. To help developers on our team stay aware of the state of the repo, we use Slack integrations for GitHub, TravisCI, and Coveralls, so we can get updates on their various outputs.

Screencasts

Red, Green, Refactor (Be sure to turn on 1080p to be able to actually read the code)
https://www.youtube.com/watch?v=Ldrxn17MOx4

Sample Execution
https://www.youtube.com/watch?v=jJ3Do05n2mA