Advanced Exercise – Week 5

Due: Before October 22, 10:00PM

Submission Instructions

To receive credit for this assignment you will need to stop by someone's office hours, demo your running code, and answer some questions.

> For first two parts, you're welcome to keep going with the RPN example from lecture and the homework, **or** you may use code from one of your other EECS classes.
>
> Remember, you **MUST** use **PRIVATE** repositories for code from other EECS classes. Private services are generally free for students, check out the GitHub Student Developer Pack.

1 Coverage

Test suites are only as good as the code they test. If you don't test a calculator's multiply function, then you will never notice if it breaks. The term *coverage* refers to how much of your code your tests cover—how much is tested.

In addition to running your test suite, *integration hooks* (the things that run when you push changes to GitHub) can also do a test suite coverage analysis. Some tools integrate with Travis, while others are standalone. I've had success with Coveralls and Python, but you're welcome to try any code coverage tool you like.

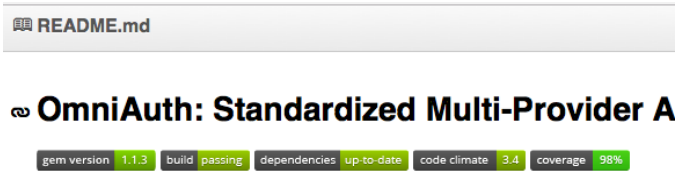
Your job: Add automated code coverage analysis to any project.

Submission checkoff:

- ☐ Which coverage tool did you use, why?
- ☐ What did you have to do to get the automatic integration working?
- ☐ Add some more code that you don't have a test for, show that coverage goes down.
  - ☐ (You don't have to do this live, save a copy of an old report before adding new code)

2 A little vanity never hurt :)

Maybe you've seen the little icons that show up on GitHub pages before that report the status of tools:



These are usually called *badges*. They're a new-ish thing, but give a nice at-a-glance status view for your project.

Submission checkoff:

- ☐ Add a badge for your build status from TravisCI
- ☐ If your coverage service supports badges (coveralls does), add that too

3 Exploring Python's many libraries

One of the things that makes Python very powerful is the diverse array of *modules* (the Python word for libraries) that it includes. For example, it's annoying that in our calculator, if we have a typo, you can't just hit the up arrow and edit the line, you have to type the whole thing again.

The bash terminal (and many, many other programs) use the `readline` library for user input, and it provides many of the nicer things you're used to (up arrow for last commands, Ctrl-r to search history, tab completion, etc).

By the magic of Python, we can get all those features really easily, simply add the line:

```
import readline
```

near the top of your `rpn.py` file. Now when you run your calculator, up arrow works!

Your Job: Find a Python library that does something cool and integrate it into your calculator. (Or think of a cool feature you want and find a Python library that does it!)

Some ideas:

- Log smarter. Currently our calculator prints out its internal stack all the time, but really we only want to do that if we're debugging things.
- Colorize things. Print back the user input but highlight operators in a different color text. Maybe print negative numbers as a bold red.
- Handle errors better. Lots of unexpected inputs can kill the whole program, that's not great. Find something that will automatically recover and start again.
- If you're feeling adventurous, it's not too hard to create a basic graphical interface, number buttons and a text box, in Python (TkInter is probably easiest, though not the prettiest – there are many options).
- Or anything else you think is cool!

Submission checkoff:

- ☐ What new feature did you add?
- ☐ What module(s) did you use to make it happen?