Milestone 3

Due Thursday, May 4 at 11:59pm

Learning Objectives:

I. **What characters do I type next?** *How do I draw from Python builtins and scientific computing modules for functionality?* This milestone will give you the chance to practice drawing from the full range of functionality we've addressed so far

II. **How do I plan a project and execute my plan?** *How do I create a piece of my project that can be written and coded on its own?* This milestone will give you the opportunity to practice coding and testing modularly for a project of larger scope.

III. **What broke and how do I fix it?** *How do I test a piece of code?* This milestone will give you an opportunity to apply your knowledge of exceptions and testing towards making your project code work.

IV. **How do I communicate science and Python with others?** *How do I document a piece of code?* This milestone will give you an opportunity to practice writing docstrings for your piece of code.

A. **Python is a physical system. Experiment!**

B. **Let me Google that for you.**

D. **Read the error output. Read it.**

F. **Write and test, write and test...**

While You Work: Habit Summary

You may have seen some of these habits listed on your lab and milestone handouts. What do they mean and why are they good habits for scientific programmers?

A. Python is a physical system. Experiment! (Debugging)

Scientific programmers can theorize and experiment about their code. When your code misbehaves, change something and see if it works! If not, figure out why. You'll learn to investigate your code by experimenting with it.

B. Let me Google that for you (Plotting and Documentation)

Google is renowned for its ability to produce results relevant to programmers. When you need to learn or re-learn how to use a piece of functionality, Google it! You'll learn when to turn to Google to find the answer to your question.

C. Computing time is cheap--use it! (Data and Workflow)

For everyday tasks, the slowest step is almost never the computer—it's the programmer. Python gives you the resources to write things quickly and get them done, so use them! You'll learn how to prioritize coding time over runtime.

D. Read the error output. Read it. (Debugging)

When a remote-control car stops working, it doesn't tell you why. Did the remote batteries die, or the car batteries? Python, on the other hand, is downright loquacious about errors. You'll learn to read and interpret the error output to fix your code.

E. Don't reinvent the wheel! (Data and Workflow)

In many languages, you have to write everything from scratch. In Python, that's almost never true: if it's general enough, someone else has probably done it. You'll learn to recognize the point when you can use the wheels others have already built.

F. Write and test, write and test… (Data and Workflow)

You know that coding project at the end of the quarter? You're not going to write it correctly all at once, and if you try to it's going to be a pain to debug. Split it up! You'll learn to write code in chunks and test them along the way.

You've started using these habits, possibly without knowing it! This part of the milestone will help you notice and solidify those habits.

While you're working, you will doubtless make use of one of these habits. **When you notice yourself using one of these habits, write down the habit and what you used it for.** Write it in a file named `habit_summary_1.txt`. It should look something like this:

```
<file habit_summary_1.txt>

Habit 1: Let me Google that for you.

While working on my file fruit_counter.py, I was trying
to write a function, is_apple, which returns True iff the
parameter is the string "apple". I couldn't remember how
to check if two strings were equal, so it occurred to me
that I should Google it. I Googled "python how to check
if two strings are equal" and found the answer: use
"string1 == string2".
```

Part 1: Identify, Write, Test, and Document a Piece of Code

For this milestone we're asking you to pick a discrete piece of code that your project will need—probably an important class definition or suite of function definitions that you'll need for your project—and then write it, test it, and write docstrings for everything in it. **Take about an hour, but no more than two hours.**

When you're done, your `milestone3` repository should contain the following:

- A Python file with your code, including

    - The code itself, which you have to write from scratch;
    - Docstrings for all classes and functions; and
    - A docstring for the file itself, which you can do by putting a docstring at the very beginning of a file (after any comments or shebangs you wish to add). Now when you import the file, e.g. `import fruit_counter`, you can access the documentation in IPython via `fruit_counter?`. **In the docstring, explain not only the contents of the file, but also how it fits into your larger project.**
- Another Python file, `tests.py`, documented with comments, which shows two examples of using your code. Make sure that when you run `python tests.py` it produces the desired output! **This will require that you spend some time testing and debugging your code, too; make sure to leave time for that.**