Lab 4

Learning Objectives:

I. **What characters do I type next?** *How do I tell Python to interact with the Unix command line and with files in a Unix architecture?* This lab will provide the opportunity to practice interacting with Unix using Python builtins.

II. **How do I plan a project and execute my plan?** *How do I integrate Git into my workflow for version control?* This lab will provide the opportunity to practice using Git within a Unix environment to track changes and communicate about them with an external server.

D. **Read the error output. Read it.** *Git and Python will usually tell you what went wrong.* This lab will raise lots of errors. Instead of trying to guess where you could have possibly gone wrong, use this as an opportunity to practice directing your attention towards what Git or Python said went wrong.

While you work is about forming a habit of reading error output from Git and Python.

Part 1 helps you set up Git; you'll only have to do this once on any given machine.

Part 2 helps you practice using Git for everyday tasks, as well as for the situations where you'll need it: somebody broke your code.

Part 3 gives you practice with accepting input to a Python program from the command line and writing its output to files.

In **part 4**, you'll submit your lab on your own.

While You Work: Read the Error Output

Programming languages will throw errors. Read them in order to figure out what the error was. If you make an error and figure out what it is, let us know! We'll put it on the board as a resource for the rest of the class. We'd like to keep track of what errors you encounter so that the class as a whole can benefit.

Part 0: Get Set Up in Git

Git likes to know who you are, so take a few seconds to tell it:

```
git config --global user.name "Firstname Lastname"
git config --global user.email your_email@stanford.edu
```

From now on, your name and email will appear next to commits you make.

Part 1: Get the Lab from Github

Today's lab has starter code, located at `https://github.com/physics91si/username-lab4`. Use what you know from lecture to get the lab into your local Physics 91SI directory. You can name the repo `lab4`. **Be sure to replace the word `username` with your Github username.**

Once that's done, look around the new repository using the Unix and Git commands you know. See if you

can find the following information:

- What visible and hidden files are in the repository?
- Does your fresh repository have any untracked files?
- When was the repository created?
- Who created it?
- What changes have been made to the repository?

Part 2: Somebody Messed With the Code

You should have noticed that the Lab 4 repository has a file named `fib.py`. Open it using Vim. It should look kinda familiar: it's the instructors' solution to the Fibonacci sequence part of Lab 1. But somebody has changed it; now it does something a little different.

Run it on the command line using `./fib.py` to see what it does. You'll get an error; read it to figure out how you're supposed to use the program.

When you try to use it correctly, you should run into another error. This one *wasn't* your fault! Somebody messed up the program. Look at the `git log` to see who.

Your task is to figure out what went wrong. You should try using all of these resources:

- The error output itself
- The body of the code
- Git, which has commands like `git diff`

Once you've figured out what went wrong, use Git's version control to revert to the last working version!

Part 3: Storing Output in a File

Now the code should work fine: check that `./fib.py approx #` (for some number ≥ 2) and `./fib.py converge` do what they're supposed to.

But the output from `./fib.py converge` is pretty ugly. That output belongs in a file; **your task is to rewrite `fib.py` so that it puts the output in a file specified by the user.** The final version should have the following behavior:

- It accepts an optional command line argument as follows: `./fib.py converge *filename*`.
- If the filename argument is passed, the output from this function is written to the specified file, rather than printed to the terminal.

Run it and see if it works! You can inspect the newly created file with Vim.

Part 4: Submit Your Lab

Use what you learned today about Git to submit your lab!