Objective. To become comfortable working with Python and git.

This week we will spend a little time working on little programs. These programs will give you a taste of what it is like to work with Python in a computer scientist’s environment. Beneath the Mac O/S is a Unix-based, extensible operating system. We’ll be using git to check out and turn in our work. We’ll introduce you to emacs, an editor suitable for a wide variety of tasks. Finally, we’ll get our first exposure to python3, the primary tool of this semester’s study. Your job, this week, is to begin developing a working knowledge of these systems. It will be worth it.

Environment Setup. The following things should be done the first time you use any new environment.

1. The first time you use our machines you should introduce yourself:

   ```
   git config --global user.name "Your Name"
   git config --global user.email "your-email@williams.edu"
   git config --global push.default simple
   git config --global core.editor emacs
   ```

   This creates an the identity you’ll use for turning in work.

2. Keep yourself organized. Create a cs135 directory in your home directory:

   ```
   mkdir ~/cs135
   ```

   You can then change your current directory to cs135:

   ```
   cd cs135
   ```

   The unix cheat sheet is helpful for learning basic unix commands.

3. Clone the shared repository (replace 18xyz with your username):

   ```
   git clone ssh://18xyz@gala.cs.williams.edu/~cs135/shared.git ~/cs135/shared
   ```

   These tasks only need to be done once this semester in any environment you use. All machines in our lab are a single, shared environment. Your laptop is an environment. A particular OIT machine is an environment.

   The Workflows.txt document in the shared repository describes how we expect you to manage your workflow for this course.
Lab Setup. Now, you need to get a copy of the repository for this lab. You clone this lab's private repository with:

```
    git clone ssh://18xyz@gala.cs.williams.edu/~cs135/18xyz/lab1.git ~/cs135/lab1
```

You can now change your directory to be the lab1 subdirectory of cs135 with:

```
    cd ~/cs135/lab1
```

Making progress. Whenever you begin a session of work, you should make sure you get the latest copy of your work. This is called *pulling* the repository from the server. You should

```
    git pull
```

During the lab session, we'll create a couple of files that we want to turn in. Suppose we create a file called `hello.py`. First, we need to commit to the changes to our repository:

```
    git add hello.py
    git commit -m 'My first program!'
```

You should commit every time you think you've made progress. Committing is an important part of managing the progress you make. It is also helpful in backing up your work.

When you're finished with a work session, you should commit one last time and *push* the changes to the server:

```
    git commit -am 'Done with work today!'
    git push
```

The `-a` switch causes all tracked files to be committed. This guarantees you get the latest version of your work every time you pull.

Required Tasks. You must push to the server the following before 11pm next Sunday:

1. To demonstrate your facility with emacs, create a file, `README.txt`, that tells me a bit about you. What's your focus at Williams? Where are you from? Where, in your hometown, is the best place to eat?

2. Write your first python program, `hello.py`. It should simply print out *Hello, world!* Welcome, programmer!

3. We'll write another program that takes an argument and manipulates it in some manner. Your program should contain a documentation string (a *docstring*) at the top in triple-quotations. It should have at least one function definition that takes at least one parameter.

   Here's one suggestion we'll likely work on in lab: write a program that prints anagrams of a query word (presented as an argument). This script includes a function, `canonical(s)`, that takes a string, converts it to lowercase, sorts the letters of the string, strips any spaces out and returns the result. The canonical function can then be used to identify any words in the unix spelling dictionary (`/usr/share/dict/words`) that have the same canonical form as your query word argument.

*