1 Server

Key terms: `hills ssh $`

Reading: Course policies[1]

Reading: Course philosophy[2]

You may use any kind of computer for this course; a keyboard is highly recommended. Building your programs on the student Linux server `hills` is the established standard, faster to start with, and very powerful, but relatively spartan and arcane. On your own device, you can use newer or fancier development tools but have to set up a way to transfer programs to and from the server. There is no right answer to this question — do what works for you.

Tips for developing on `hills`:
1. Use one of the editors `nano` (easy), `emacs` or `vim` (both are cults[3]). 2. Open two SSH sessions so that you can run commands in one while the editor program stays open in the other.

Tips for developing elsewhere:
1. Use your editor's "save to server" feature to save a lot of steps moving files by hand. 2. Install a trusted SSH key on the server to avoid typing your password many times. 3. Avoid targeting too new a version of Python by checking which version is currently installed on `hills`.

Either way, you need to use `hills` every week to test and send in a program and to peer review others' programs. Log in by making an SSH connection to `hills.ccsf.edu`; your device is likely to already have the SSH client software. Your password will be masked or suppressed as you are typing it, so don't think that your keyboard has stopped working or the connection has dropped. Upon login, you're running an interactive shell called `bash`, whose command prompt usually ends in `$`. To run other programs you can type their names here. When your program, whose name ends in the extension `.py`, implements the problem stated in the assignment prompt, use `~abrick/send` to turn it in:

```
[yourname@hills ~]$ ~abrick/send
Pass the filename of the file you want to send.
[yourname@hills ~]$ ~abrick/send myhomework.py
Success. Thank you for submitting your homework for 231.
```

The week after a program is due, use `~abrick/tally` to access your peer review materials. Its output indicates which peer work you should review next, the comments your work has received, the top submissions each week, and your current grade. Read (`nano`, `cat`) and run (`python3`) the peer programs listed. Write a review for each; they should be specific, constructive, in complete sentences, and both supportive and critical. Precede each review with the code of the file described. An example PR file is linked from the course policies.[4]

The peer review deliverable is a file with extension `.pr` that contains your reviews, ranked in order from best to worst. These lines contain linebreaks or be wrapped; they should be *long* lines. You may have to remove newlines added automatically by your editor (e.g., `nano -w` turns off this behavior). If you describe an error, make sure you include the input that triggered it. If you encounter an irredeemably dishonest or irrelevant submission, to write an explanatory comment that begins with the special marker "N/A".

First, we'll see which styles of iteration are more or less robust.

[1] `https://fog.ccsf.edu/~abrick/policies.html`
[2] `https://fog.ccsf.edu/~abrick/pedagogy.html`
[4] `https://fog.ccsf.edu/~abrick/policies.html`