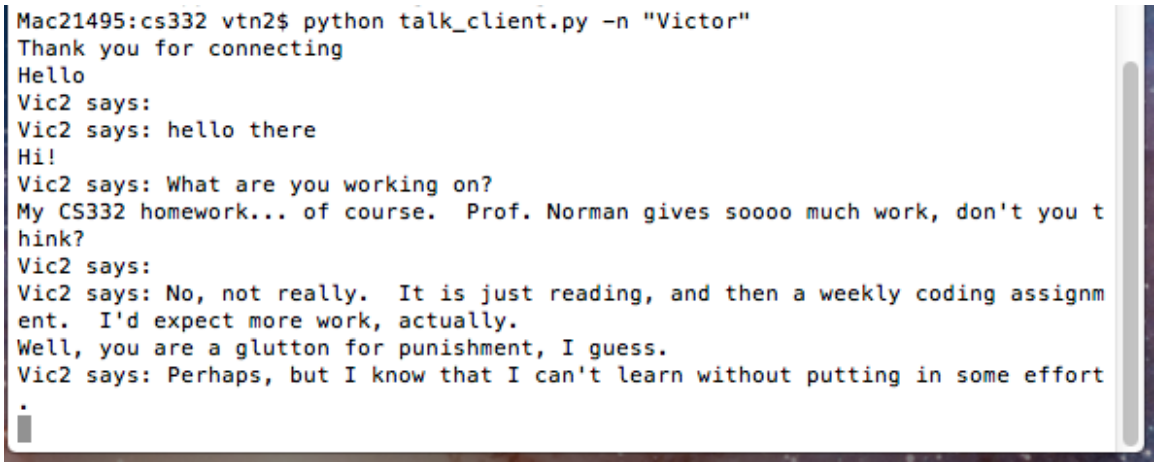
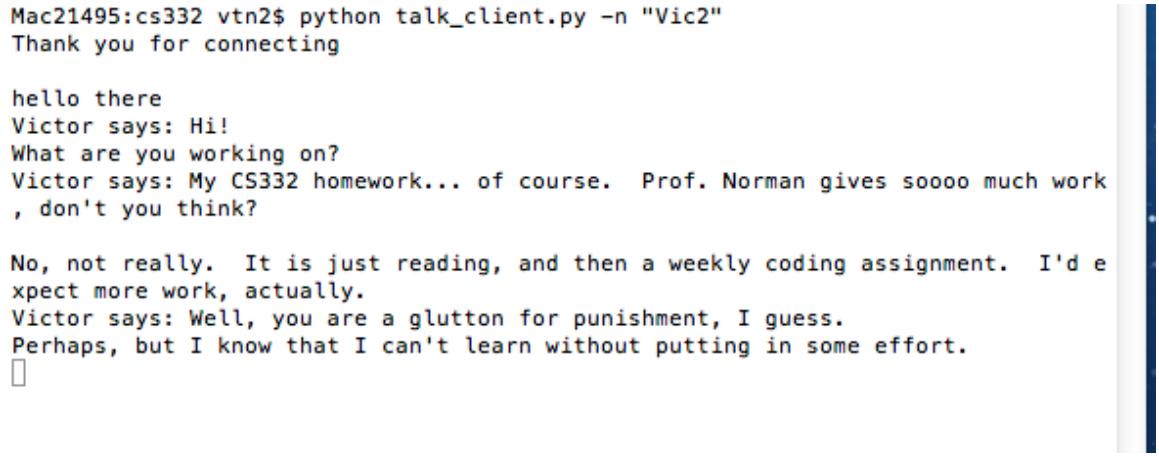
CS332, Fall 2019: TCP Chat Client and Server

For this assignment, you need to write a chat system that will allow multiple users to chat interactively in a Terminal window. One server will be running on a designated port, and any number of clients can connect and talk to each other interactively.

Here are some screen shots when 2 clients have connected to the server.





The basic functionality is this:

- Your client reads options from the command line (see below).
- Your client connects to the server.
- Your client continuously reads from stdin and from the server socket.
    - If it gets data from stdin, it prepends **<Yourname> says:** to the text and sends it to the server.
    - If it gets data from the socket, it prints it to stdout and flushes stdout.

You may implement this in python3, C, or C++. I've done it in python because it is clean and neat and I love it.

You must implement these command-line arguments:

Name(s)	Option	Meaning	Default value (if option not given)
-v, --verbose		Turn on verbose printing to help debugging	False
-s, --server	Server name or IP address	The machine the server is running on	127.0.0.1 (i.e., localhost – this machine)
-p, --port	Port number	The TCP port the server is listening on.	12345
-n, --name	A string	The name by which you are identified to others.	Your machine name.

When verbose is turned on, your client should print out enough information to help you figure out what your program is doing. E.g., my client prints out a message whenever it gets something from stdin or the socket and whenever it sends something out stdin or the socket.

Implementation Notes

Your main loop is a **while True:** loop and the first thing you do inside there (typically) is a call to select(). The select() call has stdin and the server socket in the list you are waiting to read from.

Your code should handle errors gracefully. I.e., if a system call returns an error, you should print out a useful message and exit gracefully. The user should not see a core dump or a stack trace printed out. You must handle the case where the server goes down when your client is still connected.

You will find the first lines of the talk_client.py file in `/home/cs/332/fa2019/chat/` folder.

My `talk_server` file is in `/home/cs/332/fa2019/chat.`
Run it on a linux box in either lab: ./talk_server –h. The –h option shows what the command-line options are. Running it with –v is especially useful. You will need to run the server before you can legitimately test your client.

You exit your talk_client (and the talk_server) by doing Ctrl-C in the terminal window in which it is running.

There are a **ton** of implementations of this online. For your own learning experience, do not copy and paste code from them. If you look up how to use select(), e.g., you'll probably see an implementation of the client or server… Try to implement as much as you can without copying code from online. **You must cite every place where you find code that you copy into your code.**

Those Who Want to do something really cool

Those who want to make a really nice system could consider this:

- Use `curses` to display each chatter's lines in a different color.
- Use `curses` to split the window into two parts: you type in one half and all the others' output goes into the other half.

Grading Rubric: 20 pts total

- 15 points for correctness.
  - Operates correctly: 8
  - Handles command-line options correctly: 4
  - Handles error conditions correctly: 3
- 5 points for clean, well-indented, well-documented code.

Don't forget to put your name, date, etc., in a comment at the top of your file(s). Submit by copying to `/home/cs/332/current/<yourid>/chat/`.