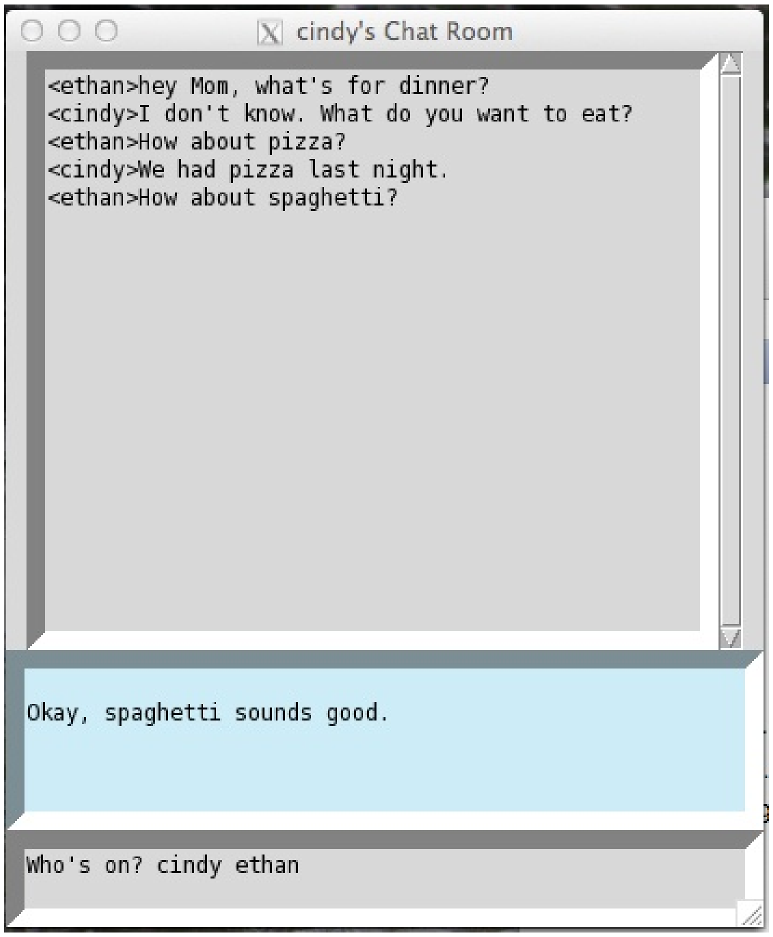
Programming Assignment 3: Chat Room

For the final assignment of the semester, you will implement a chat room application in teams of three and present the assignment during the last class. These are the teams:

Chat team 1	Julia Dana	Kevin Chabreck	Shuchee Storey
Chat team 2	Tim Jassmann	David Kale	Sina Tashakkori

A chat room has two pieces: a client and a server. The client establishes a connection with the server and sends messages to the server. The server shares messages received from a client with all clients. In the typical implementation of a chat room, the server pushes a received message out to all clients as soon as it receives a message from one client. You can find a couple of these types of implementations written in Python on-line, for example, http://www.binarytides.com/code-chat-application-server-client-sockets-python/. This implementation design is not a clean client-server architecture because the transmission of a message to all clients is initiated by the server, not the client. In contrast, your implementation is going to be strictly client-server. All data transactions will be initiated by the client. For this reason, the server will need to maintain a buffer for each of the clients. Each time the user types a message to go in the chat room, the client application will send the message to the server. The server will then save the message in each of the buffers (one for each client). At regular intervals (say once a second), a client will request the contents of the buffer. The server will respond with the contents of that client's buffer and then clear it.

The figure below is a screenshot of my chat room implementation written in Python using Tkinter to create the gui. The title bar is "cindy's Chat Room." This names the user of this execution of the client software. The user "ethan" is also executing a client. The title that appears in his gui is "ethan's chat room." The top frame contains the chat text. The middle frame is the text area in which the user can type. The user has typed "Okay, spaghetti sounds good." After a return is entered, this text will be sent to the server who stores the text in the buffer maintained for the cindy client and the buffer maintained for the ethan client. At one second intervals, each client sends a request for that buffer and then the content appears in the client's chat frame. The bottom frame displays the current users in the chat room.



```
<ethan>hey Mom, what's for dinner?
<cindy>I don't know. What do you want to eat?
<ethan>How about pizza?
<cindy>We had pizza last night.
<ethan>How about spaghetti?
```

```
Okay, spaghetti sounds good.
```

```
Who's on? cindy ethan
```

Figure 1

Part 1. Implementation a Chat Room Application in Python and Tkinter

The Server: The server is listening for two different types of requests: a request for a connection and a request from a client over an existing connection. Use the Python select statement to handle these multiple types of requests. You can see an example of the use of select at http://ilab.cs.byu.edu/python/select/echoserver.html. Specifically, the server will be handling the following types of requests:

- **connection request on the server socket -** the server will accept the request and then execute a recv to read the user's name. (Each user name needs to be unique.)
- **get request on a connected socket** - the server will send the contents of the buffer maintained for that client and clear the buffer
- **users request on a connected socket** - the server will send the names of the current users currently to the client making the request

- **put request on a connected socket** - a put request contains a client message. The server will copy the message into all client buffers

For the later three in the list above, the select statement is used to determine that an input is ready to be read on a connected socket. The server reads the input and determines whether the request is a "get", "users", or a "put" request. Thus, the request coming from the client needs to have a header that identifies the type of request. For example, a get request could be something like:

```
GET
```

The server could then respond with something like:

```
MSGS: <cindy>We had pizza last night.
<ethan>How about spaghetti?
```

where MSGS: is the response header. The server would send this to the client and clear the buffer.

A put request could be something like:

```
PUT: Okay, spaghetti sounds good.
```

The server would read this, determine the client making the request, and place "<username>Okay, spaghetti sounds good." in each client buffer. In this case, the server does not need to respond, but it could if you want.

When a connection request is accepted, you'll need to create an object that represents the client that contains the client's username, the connection socket, and the buffer. You'll keep these client objects in a list or a dictionary. When a client disconnects (which you'll be able to tell because connected socket will be ready to read, but not provide a valid "get", "users", or "put" request), then you'll need to remove the client object from the list or dictionary. Your server should not crash or display an error message if a client disconnects.

The Client: The chat client first needs to establish a connection with the server and provide the server with a user name. Then, the client needs to regularly send requests to the server for messages and users and update the messages and users frames. The client also sends "put" requests to the server when the user enters a message to be sent. Here are more details:

- Provide the user name as a command line argument, for example, by running: **./client.py cindy** If a user name is not provided, the client application should print an error message and exit. The server response should indicate whether the user name is okay. User names must be unique. If the user name is already taken, the client application should print an error message and exit.

- The Tkinter library (and interface to Tcl/Tk) is provided with the python installation so I've used this for my gui. If you want to use something else, I'm okay with that, but the directions in this assignment will provide you with some hints about how to use Tkinter. There is a bunch of

Tkinter information online. However, there are probably better libraries for building gui's in python. To use Tkinter with python 3, use this import: **import tkinter**

- You should divide your code into at least a couple of classes. In particular, you want to separate your "view" code from your "socket" code (the model-view-controller design). Your view class should contain methods to create each of the frames described above and methods to update each of these frames.

- Since a Tkinter application is event driven, the view class will actually be the driver. You'll need to bind a callback method to the frame that is accepting user input. This method will be executed when the user types a return. The callback method will use your socket code class (the view class will need to contain a reference to the socket class object) to send the message to the server.

- The two methods that update the list of users and the list of chat messages will need to schedule themselves to be called by using the Tkinter **after** function. When the update users list method is executed, it will call a method in the socket class to obtain the list of users from the server. It will then display that list and execute the after function to cause itself to be executed again a second later. The update chat messages method is similar. Note that Tkinter serializes the execution of these functions. There is just one thread of control. So although three different kinds of events can occur, these are handled one at a time.

Finally, here are some notes about the gui:

- The user should only be able to edit the text in one of the gui frames. The other two frames can only be updated by the application.

- The user should only be able to send messages that are less than 100 characters in length. You can keep a counter that counts the number of characters typed by the user. Use the bind function to bind a method to a key press. This method increments the counter by 1. When the counter reaches 100, the method deletes the last character entered. Typing a return will cause the message to be sent and the counter to be set to 0.

- When the red button in the gui is clicked, the client application should stop execution nicely. It should not display an error message.

Although Tkinter is supposed to be portable, I found that the gui on my mac is different from the gui on the student machine. Keep this in mind when you are writing the application. Your target is the student machine.

I used python3 located at /usr/local/bin/python3 on student. It is a little different from python2. For example, parentheses are required for print statements in python3. Also, the send in python3 must be given a bytes data type rather than a string. The data received at the socket must then be decoded into a string. Also, the import for Tkinter is "import tkinter" for python3 and "import Tkinter" for python2.

You don't need to use python3. So that I can easily run your code, use the Unix shebang notation at the top of your .py files to provide the path to the version of python that you chose to use.

Part 2. Doing More

Each team must complete part 1 and submit the part 1 version for part of your grade. The second part of the assignment requires that you extend your implementation in some way that you choose. Here are some ideas. You may have others.

- You can add functionality to your chat room implementation. For example, this functionality could include:
    - Multiple chat rooms so that a user could choose a room that focuses on a topic of his/her interest. A user should also be able to start up a new chat room that will focus on a particular topic.
    - A method to send a message to just one user instead of all users in the chat room
    - The ability to send audio.
    - The ability to send a picture.
- A web implementation of the chat room. The web page would consist of the gui elements of the chat room and JavaScript and Ajax (or some framework) could be used to contact the chat client (in this case, it would be more like a chat proxy) that will then contact a chat server. This implementation is somewhat different than the previous implementation because unlike the previous implementation, the chat client is not always running when a user is using the chat room. The server would have to rely on periodic updates from the client to know whether the user is still on because a connection would not be maintained between the two. Cookies can be used here to store a username.
- A mobile implementation of the chat room. The client side of the application would run on the mobile device. The server side would run on the student machine. This design could be modeled after the implementation in Part 1. It should be able to use the same server.

Final Notes

1. Create a prog3 subdirectory of your 5450 directory and V1 and V2 subdirectories of that. Do your work in the appropriate directory. You should probably set up a version control system, such as Git, that will allow you to easily share the work with your teammates.
2. Keep track of your personal responsibilities and accomplishments and the number of hours spent working on the assignment. Your grade on the assignment will be based upon both the

final product and your individual contributions. At the end of the semester, I'll ask you to complete a form that provides me with that information.

3. Create a README file in each of your version directories that explains the functionality of your code, any deficiencies in your code, and how to run it.

4. Make sure each python file contains the path to the version of python that should be used to run your code at the top of the file using the Unix shebang notation.

5. Of course, your code should be well organized and documented.

6. When you are ready to submit the first version of your programming assignment, get into your prog3/V1 directory, move any files that you don't want to submit out of that directory and type the following: submit can 4450-prog3-V1 *

7. When you are ready to submit the second version of your programming assignment, get into your prog3/V2 directory, move any files that you don't want to submit out of that directory and type the following: submit can 4450-prog3-V2 *

8. During the last day of class, your team will give a 20-25 minute presentation on the second version of the chat room. You'll demo the code, explain the functionality, and explain the design.