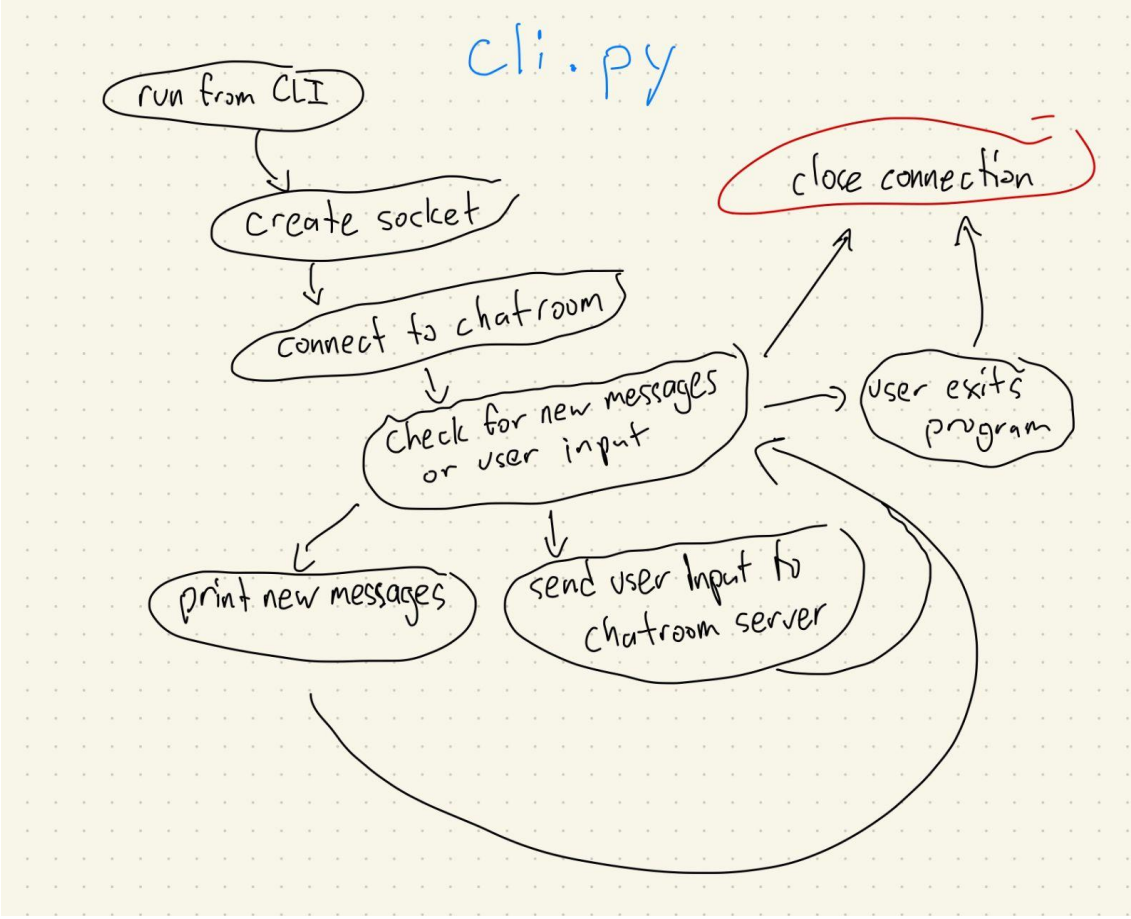
Introduction/Reference:

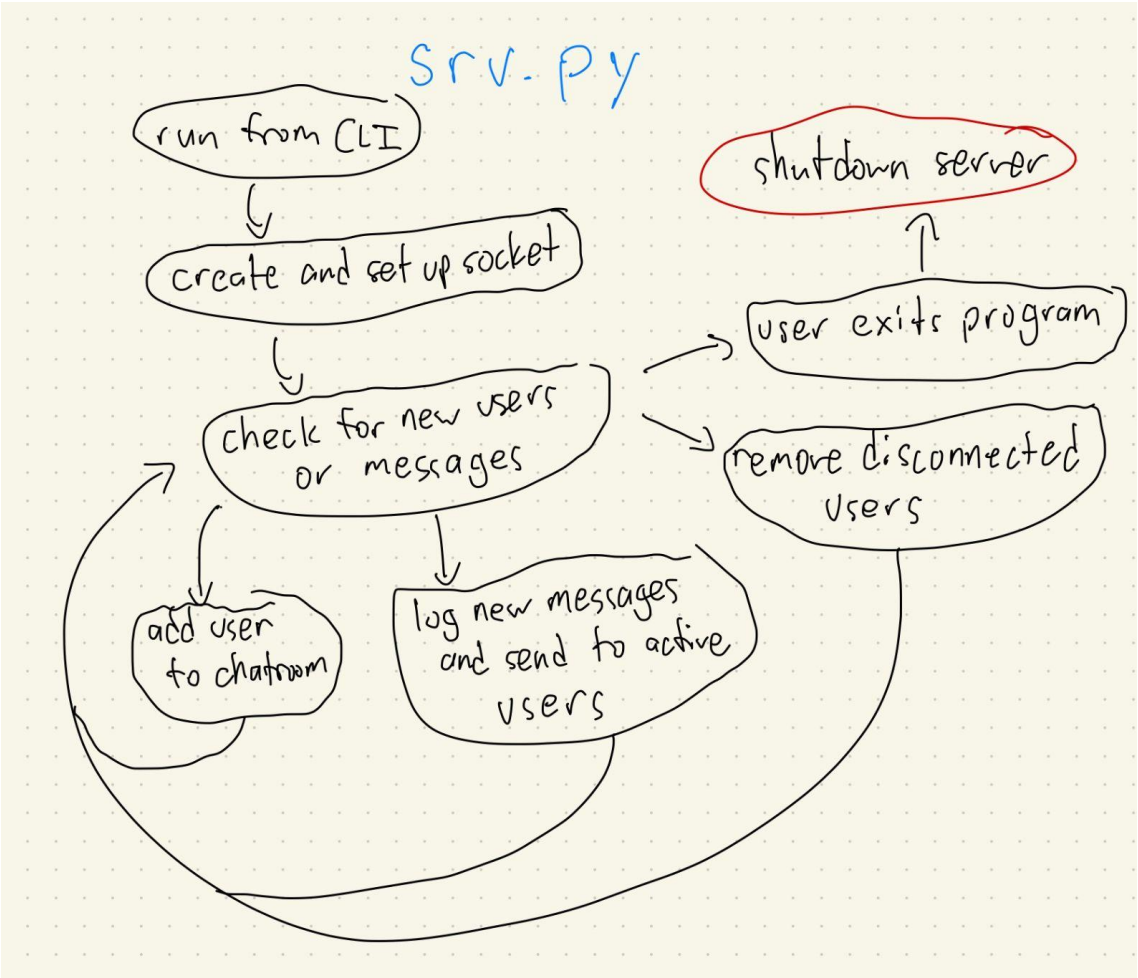
I used Ubuntu 18.04 on a Virtual Machine on VMware Fusion on my Macbook Pro. I coded the project in Python 3.6.

For reference on sockets and select I used the following websites:

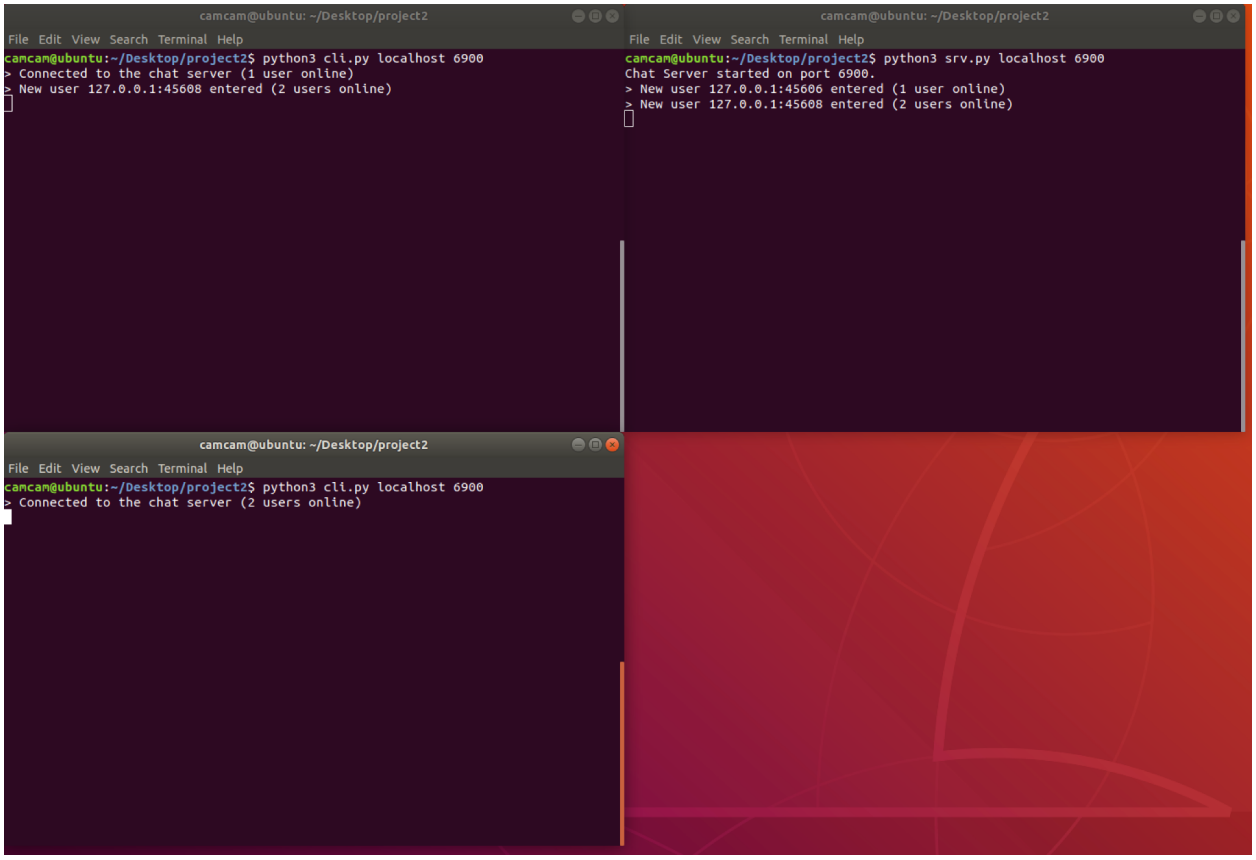
- https://docs.python.org/3/library/socket.html
- https://pymotw.com/2/select/

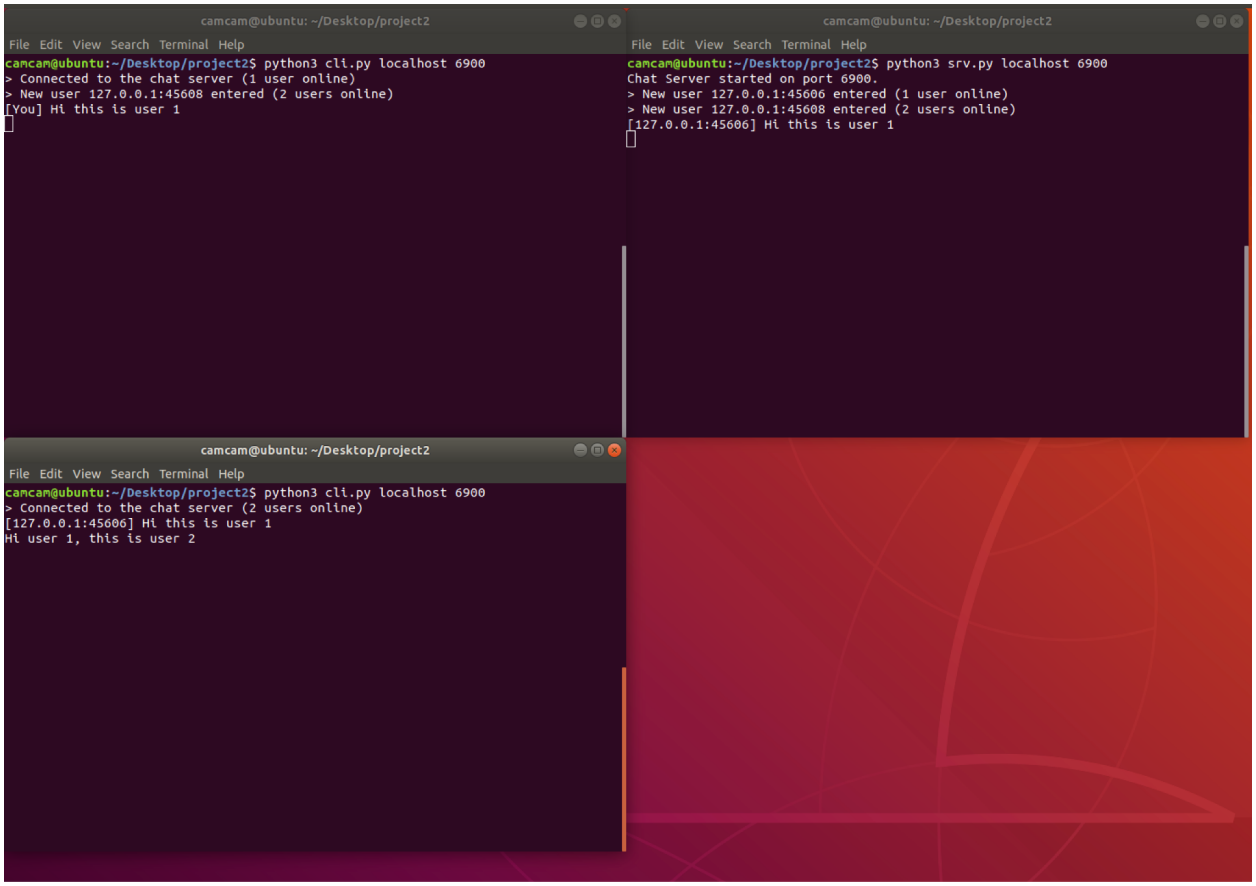
Flow Chart:

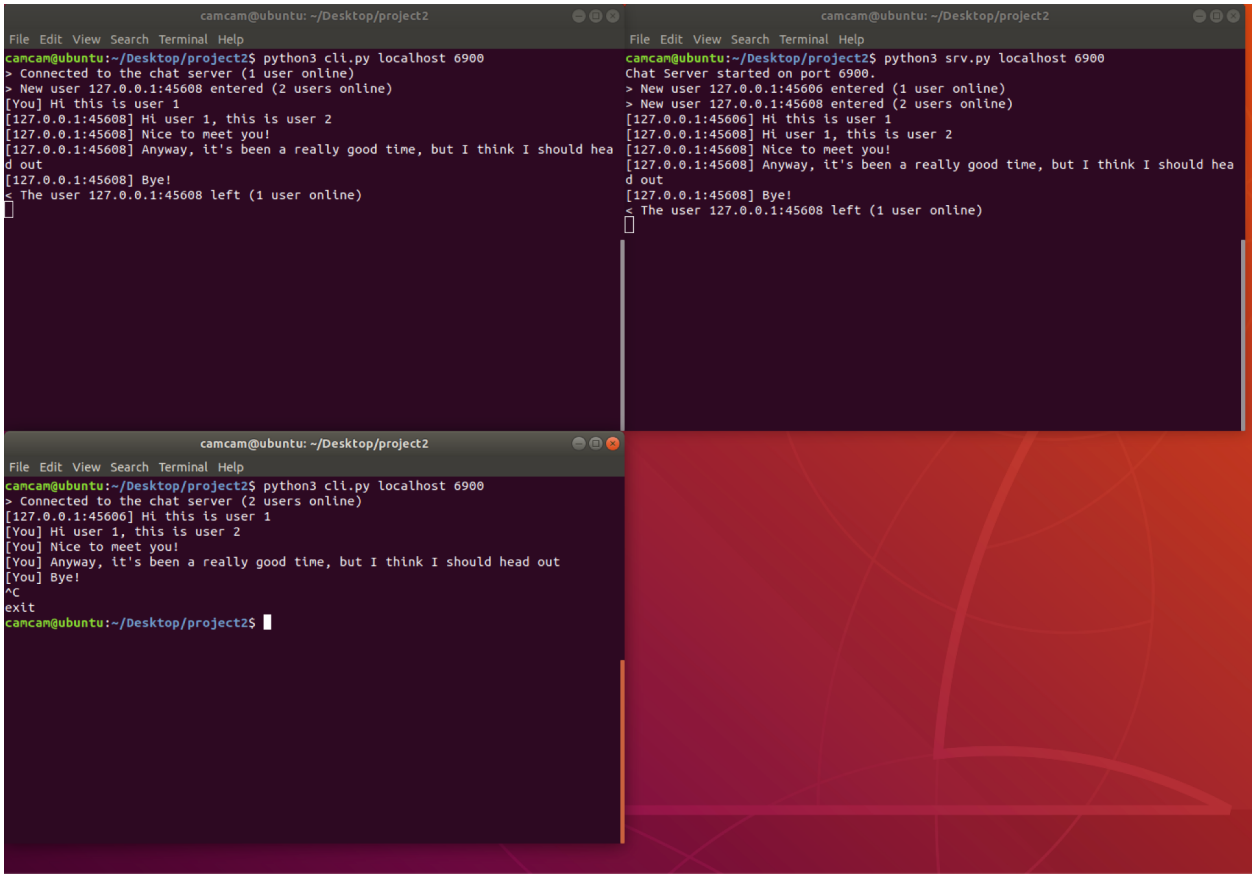


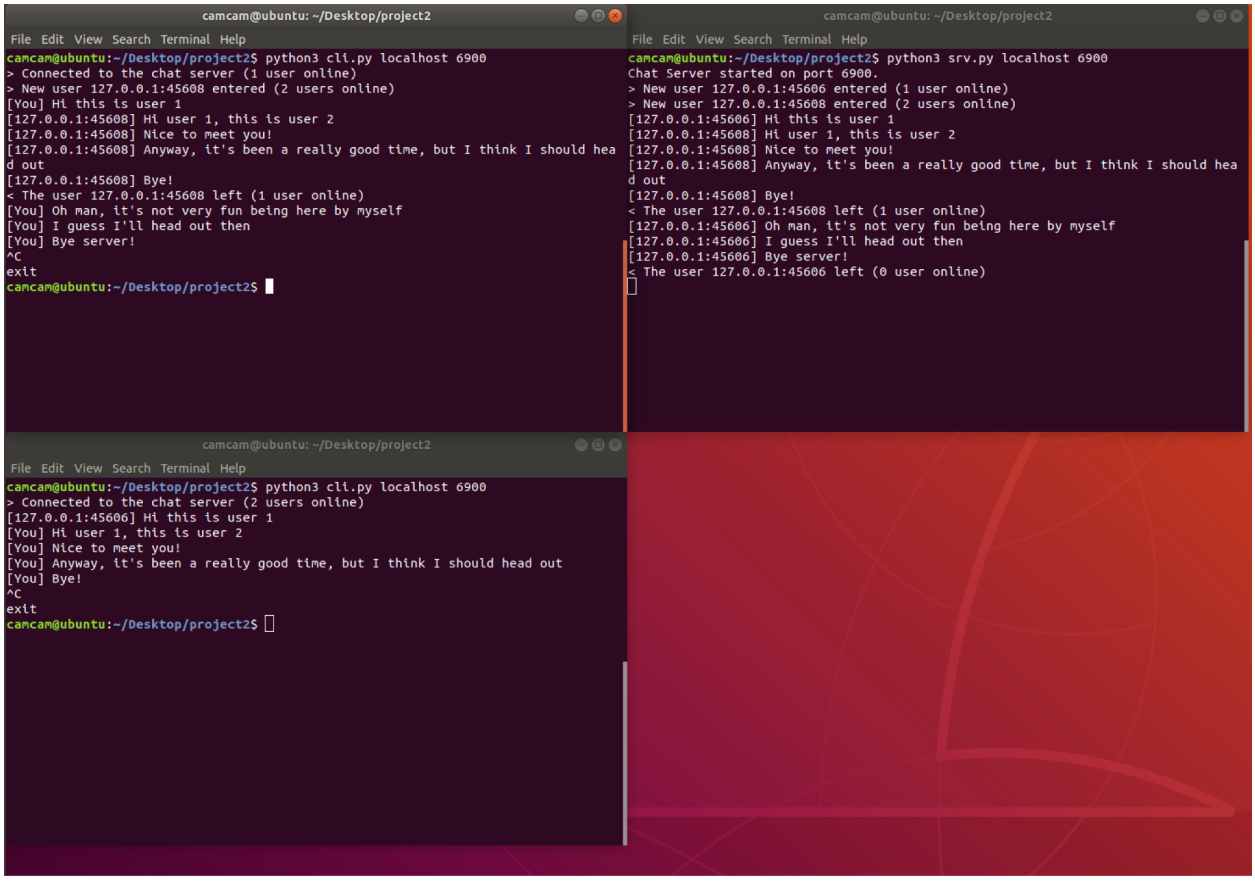

srv.py
run from CLI
create and set up socket
check for new users or messages
add user to chatroom
log new messages and send to active users
remove disconnected users
user exits program
shutdown server

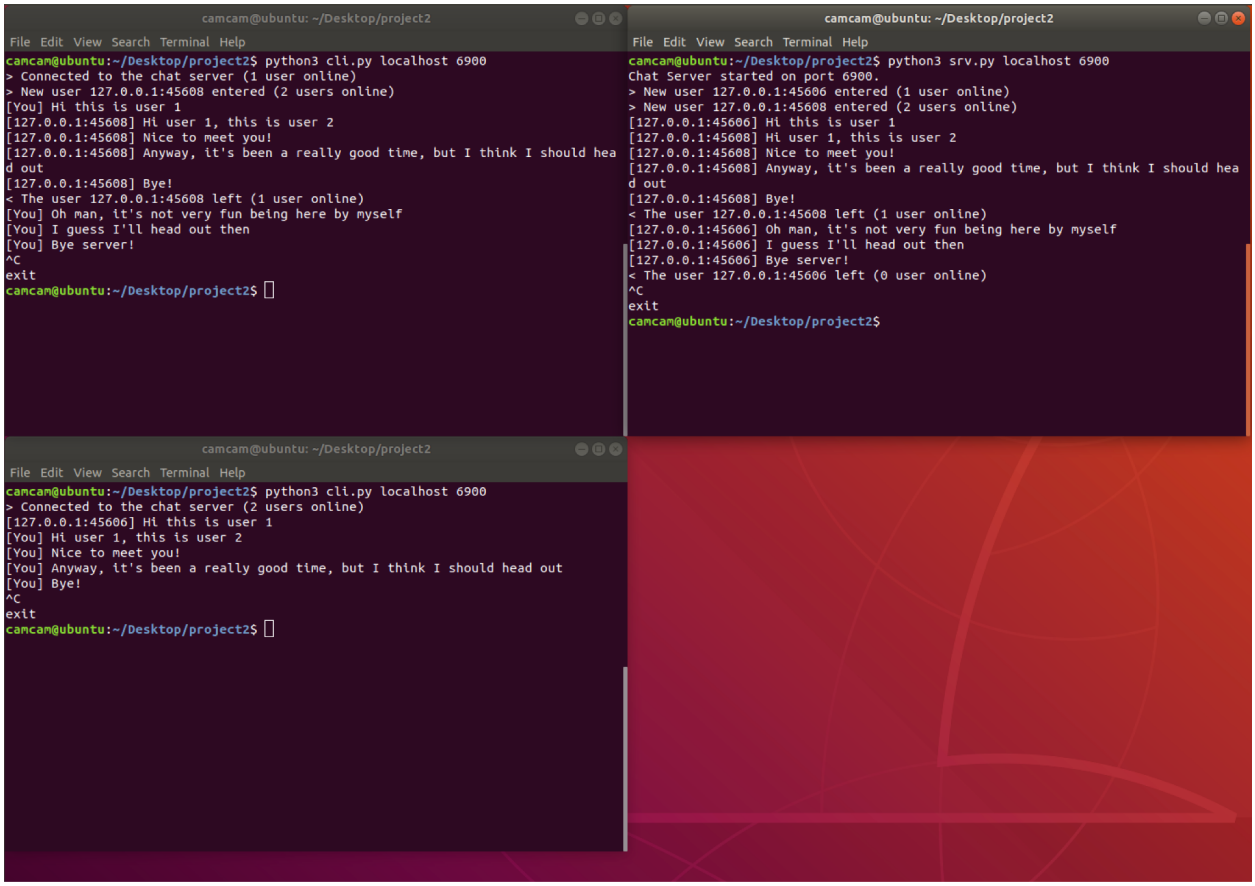
Snapshots:


camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 srv.py localhost 6900
Chat Server started on port 6900.
> New user 127.0.0.1:45606 entered (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (2 users online)


camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[You] Hi this is user 1
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 srv.py localhost 6900
Chat Server started on port 6900.
> New user 127.0.0.1:45606 entered (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[127.0.0.1:45606] Hi this is user 1
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (2 users online)
[127.0.0.1:45606] Hi this is user 1
Hi user 1, this is user 2


camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[You] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should hea
d out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 srv.py localhost 6900
Chat Server started on port 6900.
> New user 127.0.0.1:45606 entered (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[127.0.0.1:45606] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should hea
d out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (2 users online)
[127.0.0.1:45606] Hi this is user 1
[You] Hi user 1, this is user 2
[You] Nice to meet you!
[You] Anyway, it's been a really good time, but I think I should head out
[You] Bye!
^C
exit
camcam@ubuntu:~/Desktop/project2$


camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[You] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should head out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
[You] Oh man, it's not very fun being here by myself
[You] I guess I'll head out then
[You] Bye server!
^C
exit
camcam@ubuntu:~/Desktop/project2$
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 srv.py localhost 6900
Chat Server started on port 6900.
> New user 127.0.0.1:45606 entered (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[127.0.0.1:45606] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should head out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
[127.0.0.1:45606] Oh man, it's not very fun being here by myself
[127.0.0.1:45606] I guess I'll head out then
[127.0.0.1:45606] Bye server!
< The user 127.0.0.1:45606 left (0 user online)
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (2 users online)
[127.0.0.1:45606] Hi this is user 1
[You] Hi user 1, this is user 2
[You] Nice to meet you!
[You] Anyway, it's been a really good time, but I think I should head out
[You] Bye!
^C
exit
camcam@ubuntu:~/Desktop/project2$


camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[You] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should head out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
[You] Oh man, it's not very fun being here by myself
[You] I guess I'll head out then
[You] Bye server!
^C
exit
camcam@ubuntu:~/Desktop/project2$
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 srv.py localhost 6900
Chat Server started on port 6900.
> New user 127.0.0.1:45606 entered (1 user online)
> New user 127.0.0.1:45608 entered (2 users online)
[127.0.0.1:45606] Hi this is user 1
[127.0.0.1:45608] Hi user 1, this is user 2
[127.0.0.1:45608] Nice to meet you!
[127.0.0.1:45608] Anyway, it's been a really good time, but I think I should head out
[127.0.0.1:45608] Bye!
< The user 127.0.0.1:45608 left (1 user online)
[127.0.0.1:45606] Oh man, it's not very fun being here by myself
[127.0.0.1:45606] I guess I'll head out then
[127.0.0.1:45606] Bye server!
< The user 127.0.0.1:45606 left (0 user online)
^C
exit
camcam@ubuntu:~/Desktop/project2$
camcam@ubuntu: ~/Desktop/project2
File Edit View Search Terminal Help
camcam@ubuntu:~/Desktop/project2$ python3 cli.py localhost 6900
> Connected to the chat server (2 users online)
[127.0.0.1:45606] Hi this is user 1
[You] Hi user 1, this is user 2
[You] Nice to meet you!
[You] Anyway, it's been a really good time, but I think I should head out
[You] Bye!
^C
exit
camcam@ubuntu:~/Desktop/project2$

Block by Block Explanations:

srv.py

```
def welcomeUserPrint(conn):
    if len(conn) - 1 > 1:
        return "> Connected to the chat server (" + str(len(conn) - 1) + " users online)"
    else:
        return "> Connected to the chat server (" + str(len(conn) - 1) + " user online)"

def newUserPrint(cli, c):
    if len(c) - 1 > 1:
        return "> New user " + cli + " entered (" + str(len(c) - 1) + " users online)"
    else:
        return "> New user " + cli + " entered (" + str(len(c) - 1) + " user online)"


def leftUserPrint(cli, c):
    if len(c) - 1 > 1:
        return "< The user " + cli + " left (" + str(len(c) - 1) + " users online)"
    else:
        return "< The user " + cli + " left (" + str(len(c) - 1) + " user online)"
```

This block of code consists of simple helper functions that take either the list of connections, or both the list and a string containing a specific client's ip and port information. It then converts it into a readable format depending on how many users are active.

```
# Take Host IP and Port # from cli arguments
if len(sys.argv) != 3:
    raise Exception("ERROR! Usage: script, IP addr, Port #")
cli_args = sys.argv
host, port = str(cli_args[1]), int(cli_args[2])
srv_addr = (host, port)
```

This code block takes the host ip address and port number from the cli arguments when the python script file is first run in the terminal. It then type casts them and saves them in a form that can be used to bind the ip address and port to the server socket. It also checks to make sure the correct number of arguments are getting passed through.

```
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.setblocking(0)
srv.bind(srv_addr)
srv.listen(5)
srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

print("Chat Server started on port " + str(port) + ".")
```

This code block creates the server socket. It first instantiates a socket object from the builtin socket library and assigns it to the srv variable. We then make sure the socket is non blocking then bind the ip address and port number from the previous block to the server socket. We listen for up to 5 concurrent connections before we start rejecting more. We then use setsockopt to make sure we can reuse the port even if the process crashes or has been killed. Finally, after all this is set up, we print a log message of the port we are running the chat room on.

```
# Client vars
connections = [srv]

while True:
    try:
        sr, sw, se = select.select(connections, [], [])
        for s in sr:
            if s is srv:
                connection, address = s.accept()
                connections.append(connection)
```

In this code block, we create the list that stores the active connections then enter a non-terminating while loop where we keep checking for changes in the active client sockets using select(), unless the user manually exits the program with ctrl+c. If the socket/connection currently being checked is the server socket, that means a new client want to connect and we accept the connection and add it to the active connections list.

```
# Server log user joined message with user count
ip, port = connection.getpeername()[0], connection.getpeername()[1]
print(newUserPrint(str(ip) + ":" + str(port), connections))

# Send welcome message with user count
connection.send(welcomeUserPrint(connections).encode())

# Send user joined message with user count
for c in connections:
    if c is not srv and c is not connection:
        c.send(newUserPrint(str(ip) + ":" + str(port), connections).encode())
```

In this code block, we print a log message using the helper functions from the first code block, providing the ip and port information from the connection using getpeername(). We send a welcome message to the newly connected client socket, encoding it in binary. We also check to see if there are any other active client sockets and if there are we send them a message about the new user in the chat room.

```
else:
    data = s.recv(1024)
    if data:
        ip, port = s.getpeername()[0], s.getpeername()[1]
        print("[" + str(ip) + ":" + str(port) + "] " + data.decode('ascii'))
        for c in connections:
            if c is not srv and c is not s:
                ip, port = s.getpeername()[0], s.getpeername()[1]
                message = "[" + str(ip) + ":" + str(port) + "] " + data.decode('ascii')
                c.send(message.encode())
```

This code block is used when the socket currently being checked is not the server, and thus not a new socket client attempting to connect. It receives the message being sent by the active client socket. If the message is not empty, a log message of the client ip, port, and message is printed and if there are other active client sockets the message is also sent to them as well.

```
else:

    # Remove the disconnected client
    connections.remove(s)

    # Server log user left message with user count
    ip, port = s.getpeername()[0], s.getpeername()[1]
    print(leftUserPrint(str(ip) + ":" + str(port), connections))

    # Send user left message with user count
    for c in connections:
        if c is not srv and c is not s:
            ip, port = s.getpeername()[0], s.getpeername()[1]
            c.send(leftUserPrint(str(ip) + ":" + str(port), connections).encode())

    # Close the socket
    s.close()
```

This code block is used when the message sent by the client socket is empty, meaning the connection was closed. So we remove the socket from the connections list and log the user leaving the chatroom using the helper functions from the first code block. We check if there are other active client sockets and if there are we send them a message about the user leaving the chat room. Finally we close the socket.

```
except KeyboardInterrupt:
    print("\nexit")
    srv.close()
    sys.exit()
```

This code block checks to see if the user manually exited the server python script and gracefully closes the server socket and shuts down the program.

cli.py

```
# Take Host IP and Port # from cli arguments
if len(sys.argv) != 3:
    raise Exception("ERROR! Usage: script, IP addr, Port #")
cli_args = sys.argv
host, port = str(cli_args[1]), int(cli_args[2])
cli_addr = (host, port)
```

This code block takes the host ip address and port number from the cli arguments when the python script file is first run in the terminal. It then type casts them and saves them in a form that can be used to bind the ip address and port to the client socket. It also checks to make sure the correct number of arguments are getting passed through.

```
cli = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cli.connect(cli_addr)
```

This code block instantiates a socket object for the client and then connects to it using the variable we created in the previous code block.

```
while True:
    try:
        sr, sw, se = select.select([sys.stdin, cli], [], [])
        for s in sr:
            if s is cli:
```

In this code block, we enter a non-terminating while loop where we keep checking for changes in the active client socket and standard input, where the user types chat messages, using select(), unless the user manually exits the program with ctrl+c. If the socket/connection currently being checked is the client socket, that means there is an incoming message that the client needs to receive.

```
    data = s.recv(1024)
    if data:
        print(data.decode('ascii'))
    else:
        s.close()
```

In this code block, the data from the chat room server is received and if it is not empty it is printed so that the user can see the message. If it is empty that means the connection is closed on the server side so it is closed on the client side.

```
else:
    message = input()
    print('\033[F' + '[You] ' + message)
    cli.send(message.encode())
```

This code block is reached if the socket being checked is the standard input, which is when a user types a message into the terminal. It takes the input from the terminal then clears the typed line and replaces it with a formatted chat line that is much prettier. It then sends the message to the chatroom server.

```
    except KeyboardInterrupt:
        print("\nexit")
        cli.close()
        sys.exit()
```

This code block checks to see if the user manually exited the client python script and gracefully closes the client socket and shuts down the program.

Function Explanations:

1. socket(socket.AF_INET, socket.SOCK_STREAM):
    a. This function creates a new socket, in this case specifically AF_INET, and with a specific socket type, in this case sock_stream.
2. setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1):
    a. This function allows us to set socket options. In this case, we are trying to make sure we can continue to use the same port even if the program has crashed or been killed.
3. bind(("0.0.0.0", 7777)):
    a. This function allows us to bind a socket to an address, with the assumption that the socket is not already bound to an address.
4. listen(5):
    a. This allows a server to accept connections with a variable specified to limit the number of connections. If the number is reached, then the server will start refusing connections.
5. connect((host, port)):
    a. This function initiates a connection to a remote socket at a given address
6. accept():
    a. This function allows for accepting an incoming connection, and the socket must be bound to an address and listening for connections. When a connection is successfully accepted it returns a new socket object and the address bound to the socket on the other side of the connection.
7. close():
    a. This function marks a socket as closed. The socket object is officially unable to have any more operations on it. It is good to explicitly use close() rather than waiting for garbage collection.