I made three major enhancements to my AsyncClient and AsyncServer, which are the following:

1. Shareable google searching.
    a. Let's say for instance that a group of students are communicating over this chat, and are doing a research assignment. A handy feature of the messaging function in AsyncClient allows them to google search, then publically tell the other online users what they have searched (If they choose to do this; they can also search anonymously). To google search, simply type an ! followed by the search term in the messaging area. Your main browser will be pulled up, and the google search for your search term will be displayed on screen. If you would like to publically search for something, type !y followed by the search term. The other users who are currently online in the group chat will be notified of what you searched. When all of the users of the group know what you are searching for exactly, they themselves can search for different, but similar topics. e.g. Bob searches for how often rocket launches occur, which prompts Josh to search for rocket launch numbers specifically in the state of NY. This can greatly increase productivity during group projects. Interestingly enough, the server is the service which allocates the search term to the search engine, so this could be changed to include different search engines in the future.
    b. To do this, I added a modifier to the messaging function in AsyncClient, similar to how specific user mentioning is implemented. The messaging function sends the "BROWSER" JSON data to the server (a 3 tuple which includes the search term, the current user, and if the user would like to share the search), which then locks the search term to the google search engine, then optionally notifies the other users who are currently online. This data will not persist through server restarts. (although it could!)
2. Formatting options
    a. As optional parameters, the user may choose whether or not they would like to see (in the client):
        i. The time of the message (-t y or n, default y)
        ii. The user who sent the message (-u y or n, default y)
        iii. An extra blank space at the end of each message for readability (-s y or n, default n)
    b. These three commands may be used together or separately.
3. Persistent storage of chat history
    a. Now, instead of all sent messages being dumped when the server is closed, messages will be remembered through server restarts. When the server detects that there is backup data (previous messages), it will the correct messages to the screen (designated by source 'ALL'), and send the historical messages to those who enter the server.
    b. To do this, I created the message_list global variable, so that both the server and the client can receive the old data. The backup data is stored whenever a new message is sent to the server; if the backup.txt file does not exist at the time of

the message, it will be created then. When the server detects that there is previous data, it will load that previous data into message_list, and print it on all devices once a user connects. Since the server dumps the data from the backup.txt file (through a json.loads()) into a global variable called message_list, every new client instance will be able to receive the messages that came before it. Since every client can also add to the global message_list, new clients (even after the backup restore) will be completely up to date on all messages sent. A new function was created to allow the server to print the correct messages, called restore_backup(). When a new client connects to the server, and backup has not already been restored (denoted by the global backup_loaded variable), the server calls restore_backup() to print all correct previous messages to the server's screen. It is worth noting that if a message was sent to a specific user in the past, only that user will be able to see the message in the future (if they log back on). To clear the backup data at any time, simply delete the backup.txt file.