CS3251 Fall 2025
Programming Assignment 1

Handed Out: 24th Sep 2025 **Due:** 22nd Oct 2025

Objective

- Understand the creation of sockets
- Understand that application protocols are often simple plain text messages with special meaning
- Understand how to parse simple text commands.

Introduction

In this assignment, you will create a chat room on a single computer where you and your (imaginary) friends can chat with each other. The following steps will be required to achieve this:

1. Create a server program that runs on a specific port passed via the command line.
2. Create a client program that can join this server.
3. The client needs a display name and a passcode to enter the chat room. (Assume all clients use the same passcode but a different display name). The passcode will be restricted to a maximum of 5 alpha-numeric characters only. Anything over 5 letters can be treated as invalid. The display name is a maximum of 8 characters long.
4. The job of the server is to accept connections from clients, get their display name and passcode (in plaintext), verify that the passcode is correct, and then allow clients into the chat room.
5. When any client sends a message, the display name is shown before the message, followed by a colon (:), and the message is delivered to all other current clients.
6. Clients can type any text message, or can type one of the following shortcut codes to display specific text:
   (a) Type `:)` to display `[feeling happy]`
   (b) Type `:(` to display `[feeling sad]`
   (c) Type `:mytime` to display the current time
   (d) Type `:+1hr` to display the current time + 1 hour
   Note: For `:mytime` and `+1hr`, format time as "Weekday Month Day Time Year". For example, `Mon Aug 13 08:23:14 2012`.
   (e) Type `:Users` to display a list of all active users.
   (f) Type `:Msg <username> <message>` to send a private message to a specific user.
   (g) Type `:Exit` to close your connection and terminate the client
   (h) [Fun part - not graded] `\` overrides the next word until a space or newline. For example, `\:mytime` will print `:mytime` instead of the actual time.

What will you learn?

- Basic socket programming to create a client-server application
- How do multiple clients connect to a single server?
- How to keep multiple persistent TCP connections alive?
- Text parsing to develop a custom application layer protocol.

Which programming language to use?

You are required to use Python for this assignment. Your code will be tested on an Ubuntu 22.04 environment. You may set up an appropriate virtual machine to test your code locally. You may set up this VM however you prefer.
For Windows/Linux users (VirtualBox): https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview
For macOS users (Multipass): https://documentation.ubuntu.com/multipass/latest/tutorial
You can find python files with some instructions uploaded to Canvas under **`Files->PA1.zip`**.

Instructions/Expected Outputs

The autograder expects a very specific output from your programs. Please make sure that you follow all the conventions that we set and don't add any extra spaces/newlines in your output. As a rule of thumb, there should be no empty lines in your program's output and spaces should be added after a username. (For example, '`<username>: message`' instead of '`<username>:message`')

There should also not be any output from your own program on your client (such as debugging prints).

Connection Establishment and Password Checking - Single Client

Note: The passcode will be restricted to a maximum of 5 letters.

You will create two programs: a client and a server. Each program takes the following CLI parameters: the client takes the server's IP address and listening port, the username, and the password (all clients should use the same password). The server takes its listening port and the password.

If the password is correct the client should print "Connected to `<hostname>` on port `<port>`", otherwise, it should receive a failure message "Incorrect passcode". Whenever a new client successfully joins the chatroom, all other clients should receive a message indicating the username of the new user that has just joined the room (see below).

Command: `python3 server.py -start -port <port> -passcode <passcode>`
Output: `Server started on port <port>. Accepting connections`

Command: `python3 client.py -join -host <host> -port <port> -username <username> -passcode <passcode>`
Output (on Server): `<username> joined the chatroom`
Output (on new Client): `Connected to <host> on port <port>`

Resource: Sample code for providing command line arguments to a python application:
https://docs.python.org/3/library/argparse.html

Connection Establishment and Password Checking - Multiple Clients

The server should be able to handle multiple clients connecting to it. This means that by running the above client command again (with a different username), the server should perform similarly. The server should also inform the already-connected clients that a new client has joined.

Command: `python3 client.py -join -host <host> -port <port> -username <username2> -passcode <passcode>`
Output (on Server): `<username2> joined the chatroom`
Output (on new Client): `Connected to <host> on port <port>`
Output (on all other clients): `<username2> joined the chatroom`

You don't have to check for unique usernames, we will only test the code by supplying unique usernames in the test cases.
A connected client should maintain a persistent TCP connection with the server that's only terminated when the user types `:Exit`.
A client is removed only if it executes `:Exit` command (i.e., don't assume that a client will be forcibly terminated).

Chat Functionality

After successfully connecting to the server, clients should be able to type messages that get sent to the server when the user enters a newline. All text before the newline should be sent to the server, displayed on the server's screen, and broadcasted and displayed on the screens of all other clients.

Command (on a connected client with username: <username>): `Hello Room`
Output (on Server): `<username>: Hello Room`
Output (on all other clients): `<username>: Hello Room`

You don't have to consider messages longer than 100 characters. No need to test what happens when you exceed 100 characters, and no need to handle arbitrary long inputs. We do not test such corner cases.

Chat Shortcuts

As discussed earlier, clients should be able to send shortcuts that are translated to text. Emotion shortcuts `:)` and `:(` should be broadcast to the server and all other clients, excluding the original sender.

Command (on a connected client): `:)`
Output (on Server): `<username>: [feeling happy]`
Output (on all other clients): `<username>: [feeling happy]`

The time shortcuts (`:mytime` and `:+1hr`) should be broadcast to all connected clients, including the original sender.

Command (on a connected client): `:mytime`
Output (on Server): `<username>: Wed Sep 24 12:31:44 2025`
Output (on all clients, including sender): `<username>: Wed Sep 24 12:31:44 2025`

Additional Chat Commands

Additional commands are available for listing users and sending private messages.
`:Users` should specify the list of all actively connected clients, and display them in a comma-separated string with the prefix `Active Users:` (shown below)

Command (on a connected client, <username>): `:Users`
Output (on the same client <username>): `Active Users: <username1>, <username2>, <username3>`
Output (on Server): `<username>: searched up active users`

`:Msg <username> <message>` sends a private message to a specific user as shown below. We would not test sending messages to a non-existent user.

Command (on a connected client, <username>): `:Msg <username2> CS3251 is awesome`
Output (on a connected client, <username2>): `[Message from <username>]: CS3251 is awesome`
Output (on Server): `<username>: send message to <username2>`

Leaving Chatroom

As discussed earlier, clients must be able to disconnect as well. To do this, the client should be able to type `:Exit` to disconnect. All other clients should see a message that this client left the chatroom.

Command (on a connected client): `:Exit`
Output (on Server): `<username> left the chatroom`
Output (on all other clients): `<username> left the chatroom`

Grading Scheme

- Single server-client program sets up connection: 20 points
- Single server, multiple clients able to connect: 20 points
- Server receives from any client, sends to all: 20 points
- Login and passcode implementation: 15 points
- Text parsing for shortcut codes: 25 points (See 6. under the Introduction above)
- Compilation errors are not acceptable.

Programming Do's and Don'ts

- You should use **`sys.stdout.flush()`** after your print statements in Python to ensure that the message is printed on the terminal in a timely fashion.
- Your server should bind to **`127.0.0.1`** as the host. You will be only using IPv4 to bind to localhost. No attempts should be made to bind to an IPv6 address.
- It is recommended to implement the **`:Exit`** functionality first. The autograder relies on this command to cleanly terminate client connections after each test. Failure to implement this correctly will likely cause most, if not all, of the subsequent test cases to fail.

What to submit?

Please implement all functionality described above in **`server.py`** and **`client.py`** (available on Canvas) and submit these files to Gradescope. Note that there will be a minimal amount of code provided in these files, as the design of the project is up to you. Please ensure your code is well structured and readable to facilitate grading.

Further Questions?

Please use the EdStem thread Programming Assignment 1 Megathread for posting all doubts, and questions related to PA1.