

OSTPL Mini Project: Chat Room

REPORT

**Rahil Parikh 1811032
Sanmit Sahu 1811038
A2 Batch**

Index

1) Abstract.....	03
2) Theory -Python.....	04
3) Theory tkinter.....	05
4) Theory - Sockets in Python.....	10
5) Theory - Server Socket Methods.....	13
6) Theory - The Client.....	13
7) Theory - Client Socket Methods.....	14
8) Advantage.....	14
9) Disadvantage.....	14
10) Output.....	15
11) Conclusion.....	18

Abstract

Chat is the process of communicating, interacting and/or exchanging messages over the Internet. It involves two or more individuals that communicate through a chat-enabled service or software. Chat may be delivered through text, audio or video communication via the Internet.

A chat application has basic two components, viz server and client. A server is a computer program or a device that provides functionality for other programs or devices. Clients who want to chat with each other connect to the server. The chat application we made is more like a chat room, rather than a peer to peer chat. So this means that multiple users can connect to the chat server and send their messages. Every message is broadcasted to every connected chat user.

Theory

This project is designed with python, tkinter library and TCP sockets.

Python

Python is a widely used general-purpose, high level programming language. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python features a dynamic type system and automatic memory management. Python's simple syntax, modules and packages makes it possible for us to develop applications rapidly.

Professionally, Python is great for backend web development, data analysis, artificial intelligence, and scientific computing. Python contains a large *library* of standard functions which can be used for common programming tasks. Many developers have also used Python to build productivity tools, games, and desktop apps. The syntax of the language is clean and the length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

Python supports modules and packages, which encourages program modularity and code reusability. Python also supports file management and database connection. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

1. Import the Tkinter module.
2. Create the GUI application main window.
3. Add one or more of the above-mentioned widgets to the GUI application.
4. Enter the main event loop to take action against each event triggered by the user.

Tkinter Widgets:

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

1. **Tk(screenName=None, baseName=None, className='Tk', useTk=1):** To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

```
m=tkinter.Tk()
```

where m is the name of the main window object

2. **mainloop():** There is a method known by the name mainloop() is used when your application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed. `m.mainloop()`

```
import tkinter
```

```
m = tkinter.Tk()
"widgets are added here"
m.mainloop()
```

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes.

1. **pack() method:** It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:** It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:** It organizes the widgets by placing them on specific positions directed by the programmer.

Some of the major widgets are explained below:

1) **Button:** To add a button in your application, this widget is used.

The general syntax is:

```
w=Button(master, option=value)
```

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the Buttons. Number of options can be passed as parameters separated by commas. Some of them are listed below.

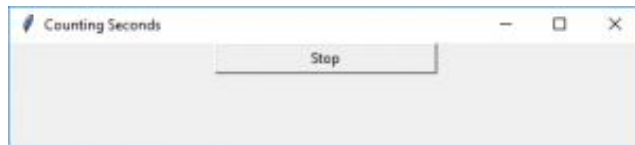
- **activebackground:** to set the background color when the button is under the cursor.
- **activeforeground:** to set the foreground color when the button is under the cursor.
- **bg:** to set the normal background color.

- command: to call a function.
- font: to set the font on the button label.
- image: to set the image on the button.
- width: to set the width of the button.
- height: to set the height of the button.

Example

```
import tkinter as tk
r = tk.Tk()
r.title('Counting Seconds')
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
button.pack()
r.mainloop()
```

Output:



2)**Entry**: It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used. The general syntax is:

`w=Entry(master, option=value)`

master is the parameter used to represent the parent window.

There are a number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

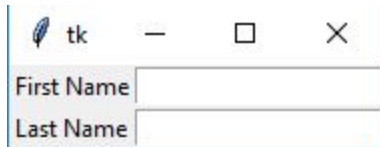
- bd: to set the border width in pixels.
- bg: to set the normal background color.
- cursor: to set the cursor used.
- command: to call a function.
- highlightcolor: to set the color shown in the focus highlight.
- width: to set the width of the button.

- height: to set the height of the button.

Example:

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Output:



3)**Scrollbar:** It refers to the slide controller which will be used to implement listed widgets. The general syntax is:

```
w = Scrollbar(master, option=value)
```

master is the parameter used to represent the parent window.

There are a number of options which are used to change the format of the widget.

Number of options can be passed as parameters separated by commas. Some of them are listed below.

- width: to set the width of the widget.
- activebackground: To set the background when the mouse is over the widget.
- bg: to set the normal background color.
- bd: to set the size of the border around the indicator.
- cursor: To appear the cursor when the mouse over the menubutton.

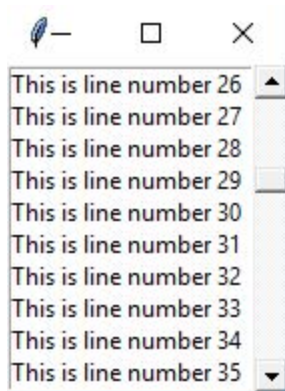
Example:

```
from tkinter import *
root = Tk()
scrollbar = Scrollbar(root)
```



```
scrollbar.pack( side = RIGHT, fill = Y )
mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
    mylist.insert(END, 'This is line number' + str(line))
mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )
mainloop()
```

Output:



Sockets in Python

Definition:

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as `send()` and `recv()` work with sockets in the same way they do with files and pipes. A socket is one endpoint of a two-way communication link between two programs running on the network.

A socket is bound to a port number so that the tcp layer can identify the application that the data is destined to be sent to. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.

The server just waits, listening to the socket for a client to make a connection request.

On the client side : The client knows the host name of the machine of which the server is running and the port number in which the server is listening. To make a connection request .

The client also needs to identify itself to the server so it binds to the local port number that it will use during connection. This is usually assigned by system. If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote and endpoint set to the address and port of the client.

It needs a new socket for connection requests while trending to the needs of a connected client.

On the client side, if the connection is accepted a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

To create a socket, you must use the `socket.socket()` function available in `socket` module, which has the general syntax –

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- `socket_family` – This is either `AF_UNIX` or `AF_INET`, as explained earlier.
- `socket_type` – This is either `SOCK_STREAM` or `SOCK_DGRAM`.
- `protocol` – This is usually left out, defaulting to 0.

Once you have a socket object, then you can use required functions to create your client or server program.

The chat server does the following things

1. Accept multiple incoming connections for clients.
2. Read incoming messages from each client and broadcast them to all other connected clients.

We will be using TCP sockets for this purpose, and therefore we use `AF_INET` and `SOCK_STREAM` flags. We use them over UDP sockets because they're more telephonic, where the recipient has to approve the incoming connection before communication begins, and UDP sockets are more post-mail sort of thing (anyone can send a mail to any recipient whose address s/he knows), so they don't really require an establishment of connection before communication can happen. Clearly, TCP suits more to our purpose than UDP sockets, therefore we use them.

def accept_incoming_connecons():

This is just a loop that waits forever for incoming connections and as soon as it gets one, it logs the connection (prints some of the connection details) and sends the connected client a welcome message. Then it stores the client's address in the addresses dictionary and later starts the handling thread for that client. Of course, we haven't yet defined the target function `handle_client()`.

def handle_client(client):

Naturally, after we send the new client the welcoming message, it will reply with the name s/he wants to use for further communication. In the `handle_client()` function, the first task we do is we save this name, and then send another message to the client, regarding further instructions. After this comes the main loop for communication: here we receive further messages from the client and if a message doesn't contain instructions to quit, we simply broadcast the message to other connected clients (we'll be defining the broadcast method in a moment). If we do encounter a message with exit instructions (i.e., the client sends a {quit}), we echo back the same message to the client (it triggers close action on the client side) and then we close the connection socket for it. We then do some cleanup by deleting the entry for the client, and finally give a shoutout to other connected people that this particular person has left the conversation.

def broadcast(msg, prex="")

This is pretty much self-explanatory; it simply sends the message to all the connected clients, and prepends an optional prefix if necessary. We do pass a prefix to `broadcast()` in our `handle_client()` function, and we do it so that people can see exactly who is the sender of a particular message. That was all the required functionalities for our server.

Server Socket Methods

Sr.No.	Method & Description
1	s.bind() This method binds address (hostname, port number pair) to socket.
2	s.listen() This method sets up and start TCP listener.
3	s.accept() This passively accept TCP client connection, waiting until connection arrives (blocking).

The Client:

This module manages any client willing to connect on a specific host and port. It authorizes the client to send and receive messages.To do:

- Write or improve docstrings
- Create a specific log file to log crashes and exceptions.

def receive():

We need an infinite loop again, Because we'll be receiving messages quite non-deterministically, and independent of how and when we send the messages. We don't want this to be a walkie-talkie chat app which can only either send or receive at a time; we want to receive messages when we can, and send them when we want. The functionality within the loop is pretty straightforward; the `recv()` is the blocking part. It stops execution until it receives a message, and when it does, we move ahead and append the message to `msg_list`. We will soon define `msg_list`, which is basically a Tkinter feature for displaying the list of messages on the screen.

def send(event=None):

We're using event as an argument because it is implicitly passed by Tkinter when the send button on the GUI is pressed. my_msg is the input field on the GUI, and therefore we extract the message to be sent using msg = my_msg.get(). After that, we clear the input field and then send the message to the server, which, as we've seen before, broadcasts this message to all the clients(if it's not an exit message). If it is an exit message, we close the socket and then the GUI app (via top.close())

Client Socket Methods

Sr.No.	Method & Description
1	s.connect() This method actively initiates TCP server connection.

ADVANTAGES

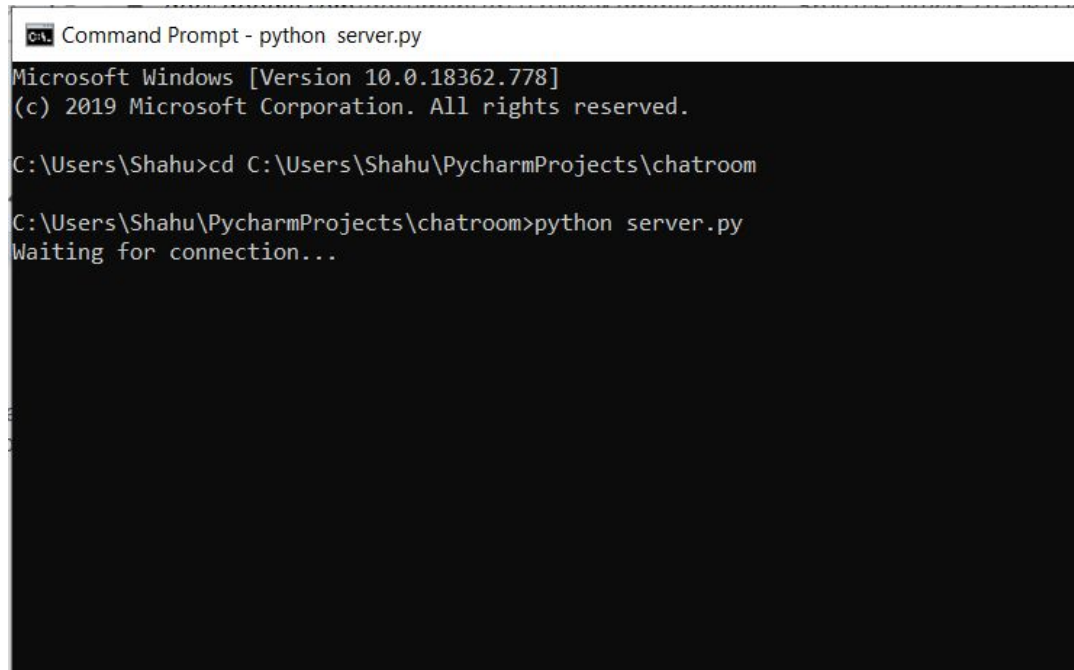
· If developed further by adding a database, frontend and buying a server place on the internet, this project can be converted into a real time chat application.

LIMITATION

· The clients can only chat by connecting to a local server.· Two clients on different networks cannot chat together.

Working model

1. Server is running and working and waiting for client

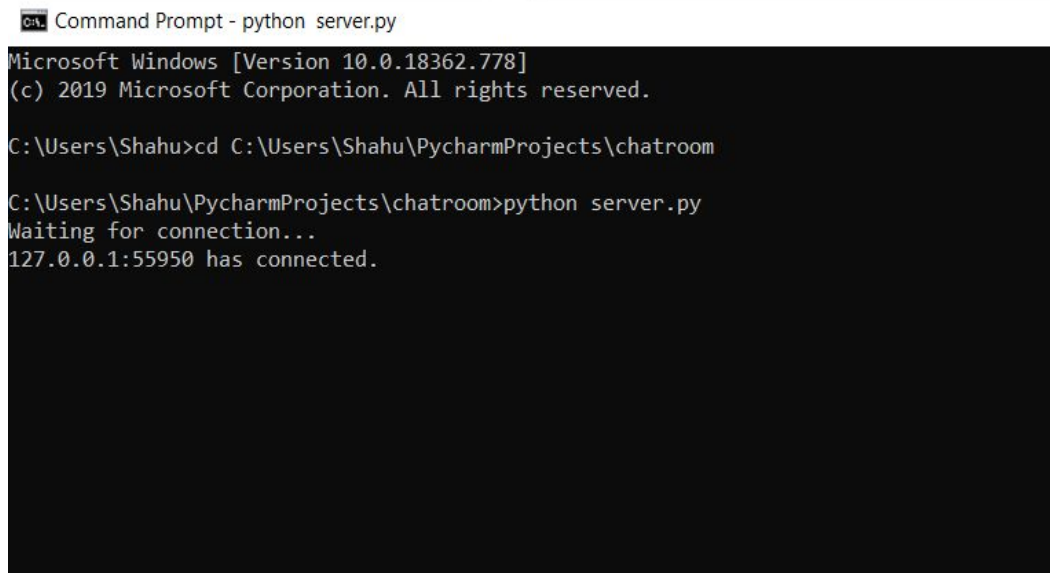


```
Command Prompt - python server.py
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Shahu>cd C:\Users\Shahu\PycharmProjects\chatroom

C:\Users\Shahu\PycharmProjects\chatroom>python server.py
Waiting for connection...
```

2. Clients Connecting



```
Command Prompt - python server.py
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Shahu>cd C:\Users\Shahu\PycharmProjects\chatroom

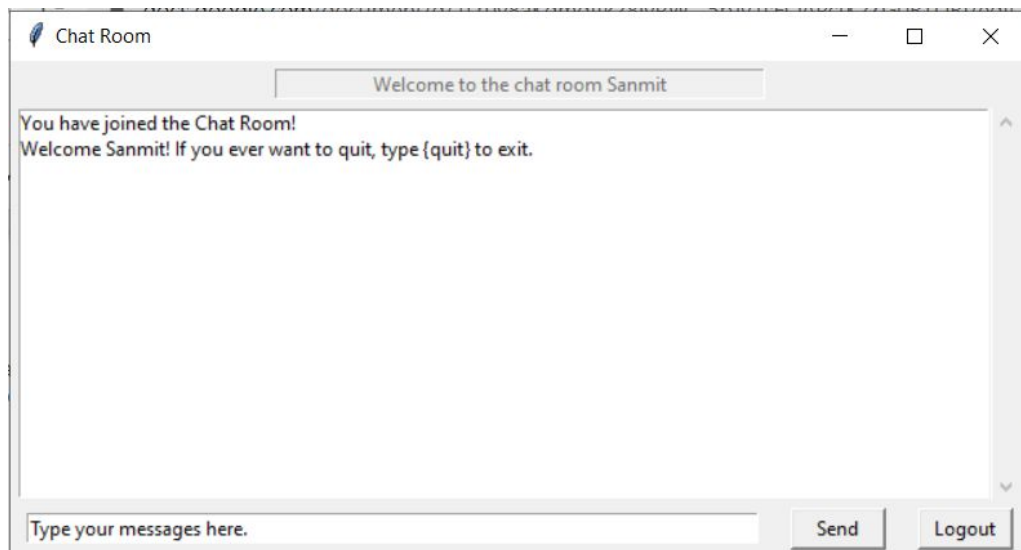
C:\Users\Shahu\PycharmProjects\chatroom>python server.py
Waiting for connection...
127.0.0.1:55950 has connected.
```

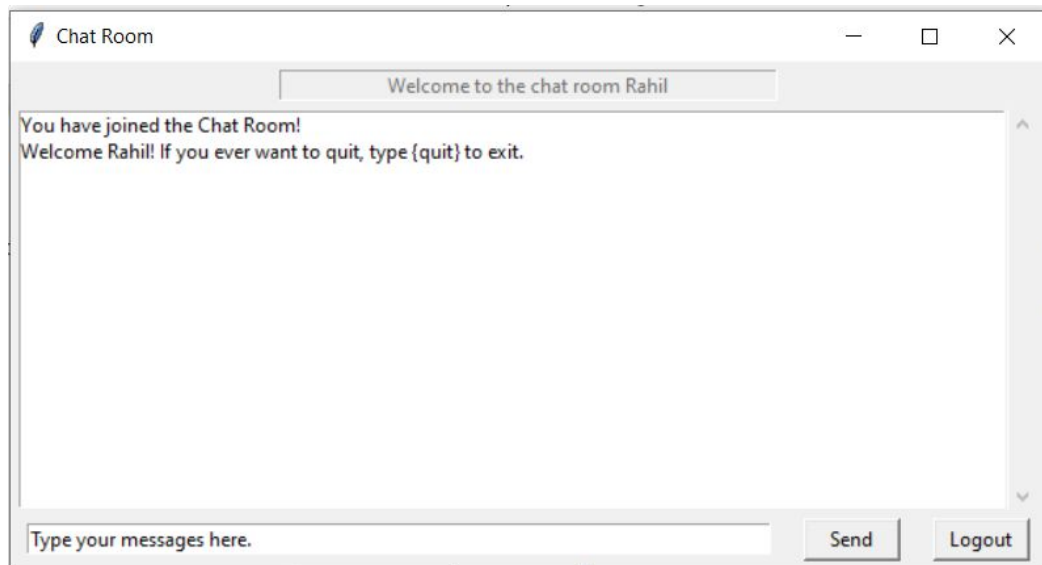
```
Command Prompt - python server.py
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Shahu>cd C:\Users\Shahu\PycharmProjects\chatroom

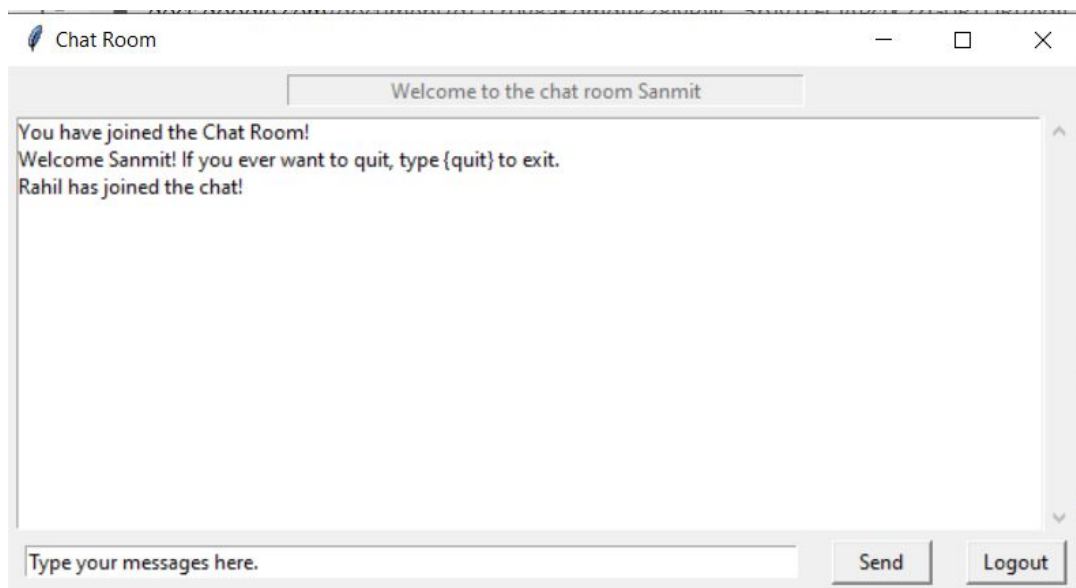
C:\Users\Shahu\PycharmProjects\chatroom>python server.py
Waiting for connection...
127.0.0.1:55950 has connected.
127.0.0.1:55952 has connected.
```

3. Clients entering their names

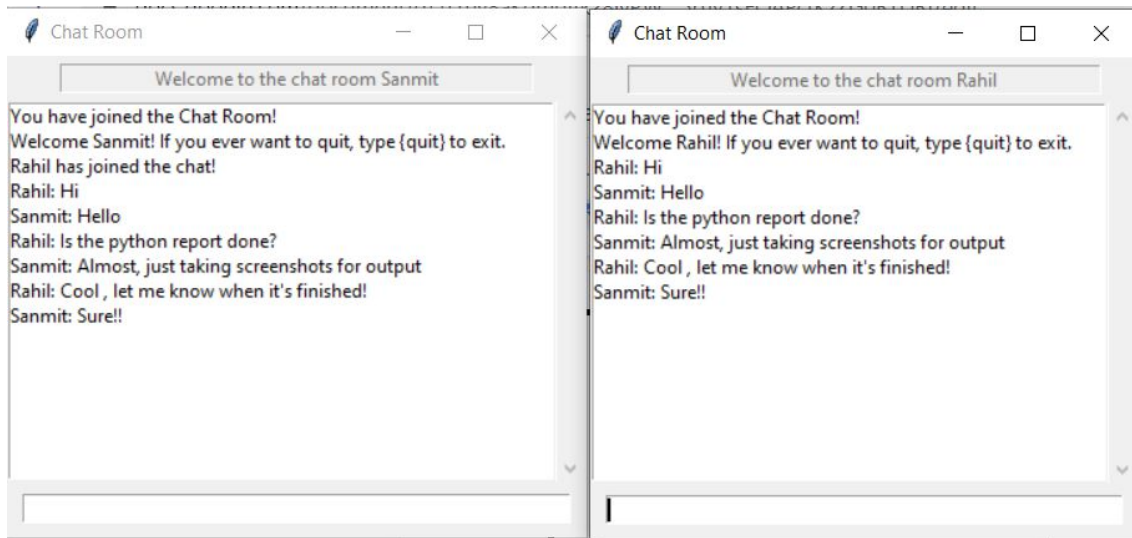




4.Clients getting notified who has joined the chat



5.Clients chatting by connecting to server



CONCLUSION:

In this project we have used the basics of networking in python. We also learned how to make a GUI for our application. This project teaches us how to create a basic chat application by creating a local network and using TCP sockets.