TABLE OF CONTENTS

1. ABSTRACT

- ·Chat rooms are Web sites or programs that allow people to send text messages to one another in real time. The chat room works as a virtual room, where groups of people send messages that others can read instantaneously.

- We are going to build a chat room which is an interface that allows two or more people to chat and send messages to everyone in the room. The aim is to do a chat room server and allow multiple clients to connect to it using sockets in Python.

- · The idea is to use the socket module which comes built-in with Python and provides us with socket operations that are widely used on the Internet, as they are behind any connection to any network.

- · This involves a client server architecture where there is one server which keeps listening to new client connections formed and messages sent by them, and broadcasts it to the other clients.

- Motivation to this project is to make communication inside an organization or office or company more reliable and easy to use.

2. Introduction

The key concept behind this project is Socket programming.A socket is one endpoint of a two-way communication link between two programs running on the network.

Socket programming is a way of connecting two points on a network to communicate with each other.

1. Generally, one socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.
   Characteristics of Client and Server:

.1. Client:

- · Always initiates requests to servers.
- · Waits for replies.
- · Receives replies.
- · Usually connects to a small number of servers at one time.
- · Usually interacts directly with end-users using any user interface such as graphical user interface.

.2. Server:

- · Always wait for a request from one of the clients.
- · Serves clients' requests then replies to the clients with the requested data.
- · A server may communicate with other servers in order to serve a client request.
- · If additional information is required to process a request (or security is implemented), a server may request additional data (passwords) from a client before processing a request.
- · End users typically do not interact directly with a server, but use a client.

3. Requirement Analysis

Hardware Requirements

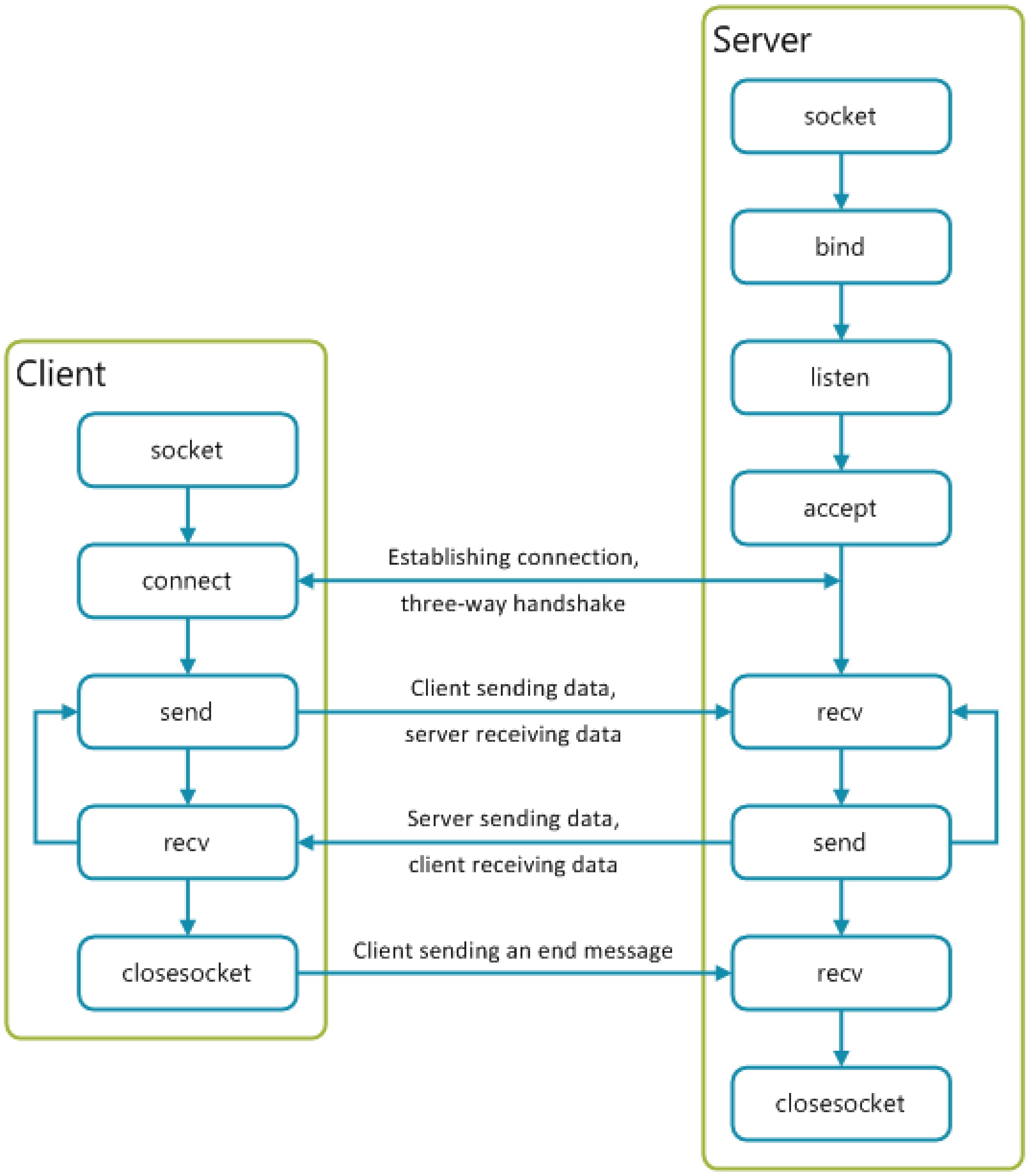
Processor	**2.4 GHz Clock Speed**
RAM	**1 GB**
Hard Disk	**500 MB (Minimum Space)**

Software Requirements

Operating System Version	**Windows 7 and Above**
Platform	**Jupyter Notebook**
Backend	**Python**
Special Tools	**Socket, Threading, colorama,Playsound**
Server	**Server**

4. Architecture and Design

Diagram:



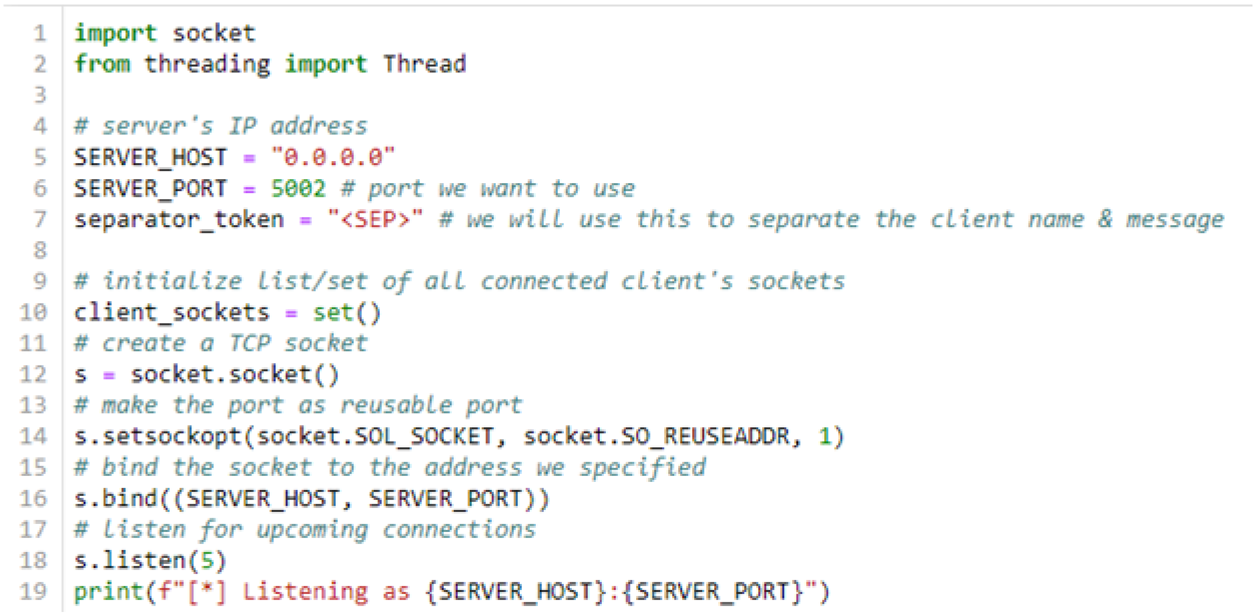
5. Implementation

SERVER CODE:

In our architecture, the whole job of the server is to do two essential operations:

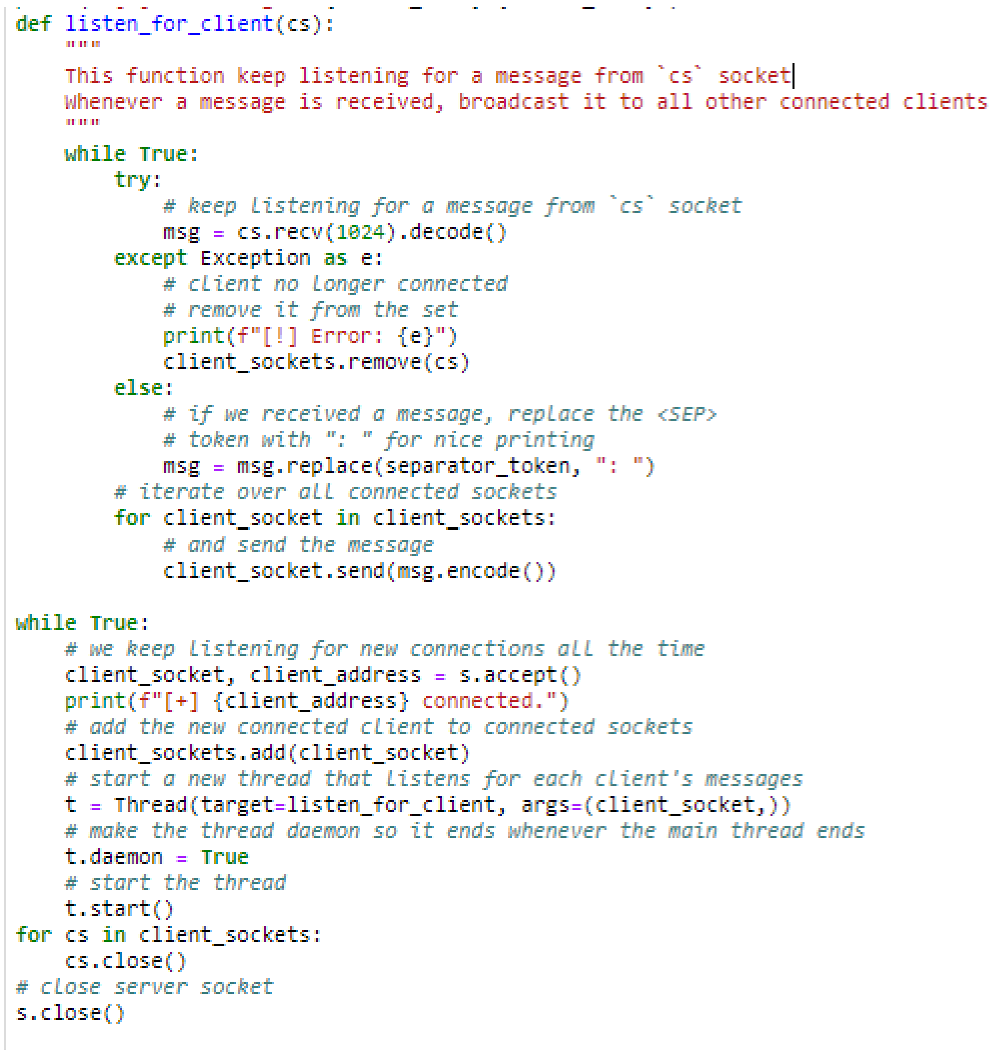
- Listening for upcoming client connections, if a new client is connected, we add it to our collection of client sockets.
- Start a new thread for each client connected that keeps listening for upcoming messages sent from the client and broadcasts it to all other clients.

The below code creates a TCP socket and binds it to the server address, and then listens for upcoming connections:



```python
import socket
from threading import Thread

# server's IP address
SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5002 # port we want to use
separator_token = "<SEP>" # we will use this to separate the client name & message

# initialize list/set of all connected client's sockets
client_sockets = set()
# create a TCP socket
s = socket.socket()
# make the port as reusable port
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# bind the socket to the address we specified
s.bind((SERVER_HOST, SERVER_PORT))
# listen for upcoming connections
s.listen(5)
print(f"[*] Listening as {SERVER_HOST}:{SERVER_PORT}")
```

- We've used `"0.0.0.0"` as the server IP address. this means all IPv4 addresses on the local machine. You may wonder, why we don't just use localhost or `"127.0.0.1"` ? Well, if the server has two IP addresses, let's say `"192.168.1.2"` on a network and `"10.0.0.1"` on another, then the server listens on both networks.
- We're not yet accepting connections, as we didn't call `accept()` method, the below code finishes the server code recipe:



```
def listen_for_client(cs):
    """
    This function keep listening for a message from `cs` socket
    Whenever a message is received, broadcast it to all other connected clients
    """
    while True:
        try:
            # keep listening for a message from `cs` socket
            msg = cs.recv(1024).decode()
        except Exception as e:
            # client no longer connected
            # remove it from the set
            print(f"[!] Error: {e}")
            client_sockets.remove(cs)
        else:
            # if we received a message, replace the <SEP>
            # token with ": " for nice printing
            msg = msg.replace(separator_token, ": ")
        # iterate over all connected sockets
        for client_socket in client_sockets:
            # and send the message
            client_socket.send(msg.encode())

while True:
    # we keep listening for new connections all the time
    client_socket, client_address = s.accept()
    print(f"[+] {client_address} connected.")
    # add the new connected client to connected sockets
    client_sockets.add(client_socket)
    # start a new thread that listens for each client's messages
    t = Thread(target=listen_for_client, args=(client_socket,))
    # make the thread daemon so it ends whenever the main thread ends
    t.daemon = True
    # start the thread
    t.start()
for cs in client_sockets:
    cs.close()
# close server socket
s.close()
```

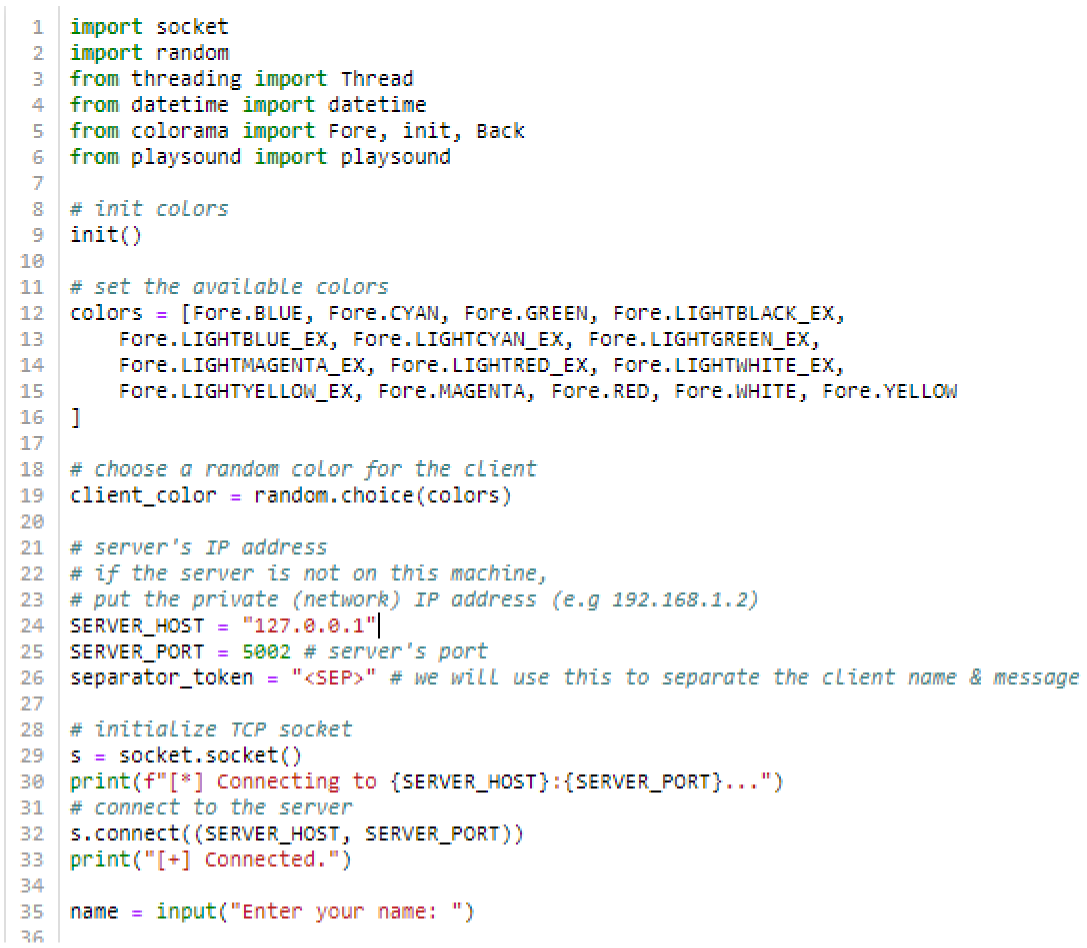
- we add the connected client socket to the collection of our sockets, and then we start a new thread and we set it as a daemon thread that executes our defined `listen_for_client()` function, which given a client socket, it waits for a message to be sent using `recv()` method, if so, then it sends it to all other connected clients.Finally, we are closing all the sockets.

CLIENT CODE:

The client does three basic operations:

- Connects to the server.
- Keep listening for messages coming from the server (must be a client sent a message to the server and the server broadcasted it) and print it to the console.
- Waiting for user to input messages to send to the server

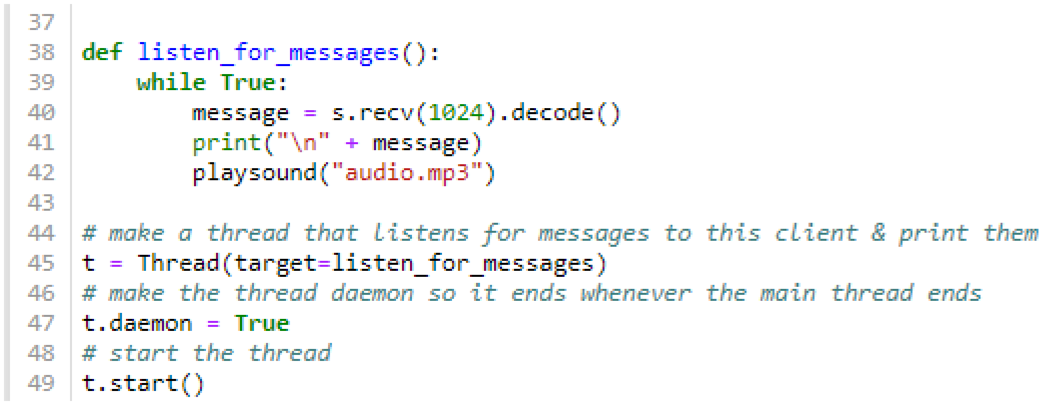
Here's the code for the first operation:



```python
import socket
import random
from threading import Thread
from datetime import datetime
from colorama import Fore, init, Back
from playsound import playsound

# init colors
init()

# set the available colors
colors = [Fore.BLUE, Fore.CYAN, Fore.GREEN, Fore.LIGHTBLACK_EX,
    Fore.LIGHTBLUE_EX, Fore.LIGHTCYAN_EX, Fore.LIGHTGREEN_EX,
    Fore.LIGHTMAGENTA_EX, Fore.LIGHTRED_EX, Fore.LIGHTWHITE_EX,
    Fore.LIGHTYELLOW_EX, Fore.MAGENTA, Fore.RED, Fore.WHITE, Fore.YELLOW
]

# choose a random color for the client
client_color = random.choice(colors)

# server's IP address
# if the server is not on this machine,
# put the private (network) IP address (e.g 192.168.1.2)
SERVER_HOST = "127.0.0.1"
SERVER_PORT = 5002 # server's port
separator_token = "<SEP>" # we will use this to separate the client name & message

# initialize TCP socket
s = socket.socket()
print(f"[*] Connecting to {SERVER_HOST}:{SERVER_PORT}...")
# connect to the server
s.connect((SERVER_HOST, SERVER_PORT))
print("[+] Connected.")

name = input("Enter your name: ")
```

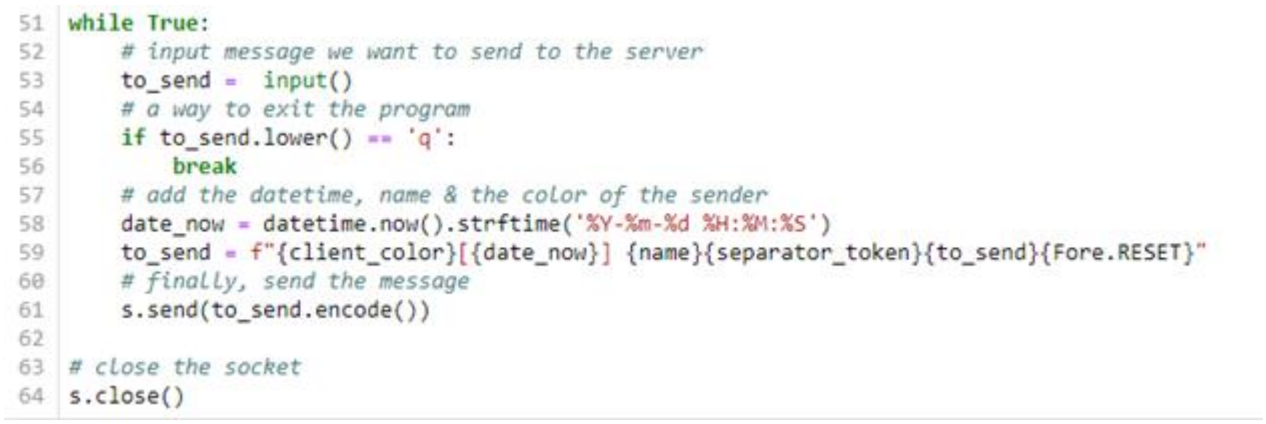
- As a side operation, we also set a color for each client, you'll see it in the output. Also, set a name for each client, so we can distinguish between clients.
- The below code is responsible for the second operation; keep listening for messages from the server and print them to the console and also for each

client and message, a notification sound will be played. So that the other client will be aware of the incoming or sending messages



```python

def listen_for_messages():
    while True:
        message = s.recv(1024).decode()
        print("\n" + message)
        playsound("audio.mp3")

# make a thread that listens for messages to this client & print them
t = Thread(target=listen_for_messages)
# make the thread daemon so it ends whenever the main thread ends
t.daemon = True
# start the thread
t.start()
```

- We also want it to be in a separate thread as a daemon thread, so we can do other things while listening for messages.
- Now let's do the final task; waiting for user input for messages, and then send them to the server:



```python
while True:
    # input message we want to send to the server
    to_send =  input()
    # a way to exit the program
    if to_send.lower() == 'q':
        break
    # add the datetime, name & the color of the sender
    date_now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    to_send = f"{client_color}[{date_now}] {name}{separator_token}{to_send}{Fore.RESET}"
    # finally, send the message
    s.send(to_send.encode())

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

- We add the client color, name, and the current date-time to the message to be sent.
- we send the message using `send()` method and we make a way to exit out of the problem, by just inputting `'q'` character in the place of the message.

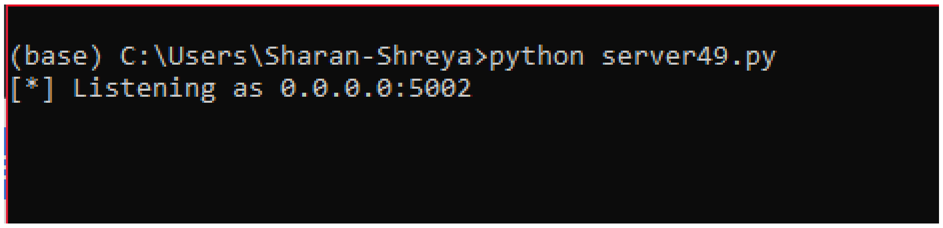
6. Experiment Results and Analysis

1. Results:

Now Going on to the Demonstration:

Server:

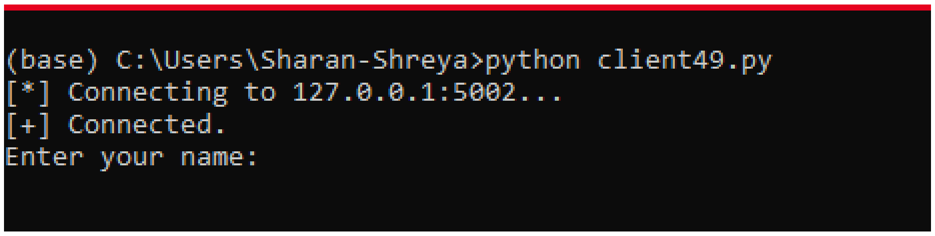
Figure a:



```
(base) C:\Users\Sharan-Shreya>python server49.py
[*] Listening as 0.0.0.0:5002
```

Client no 1:

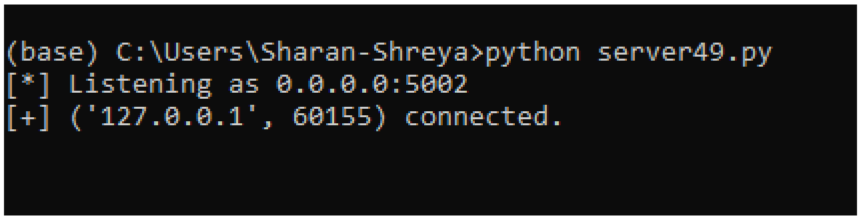
Figure b:



```
(base) C:\Users\Sharan-Shreya>python client49.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name:
```

Server:

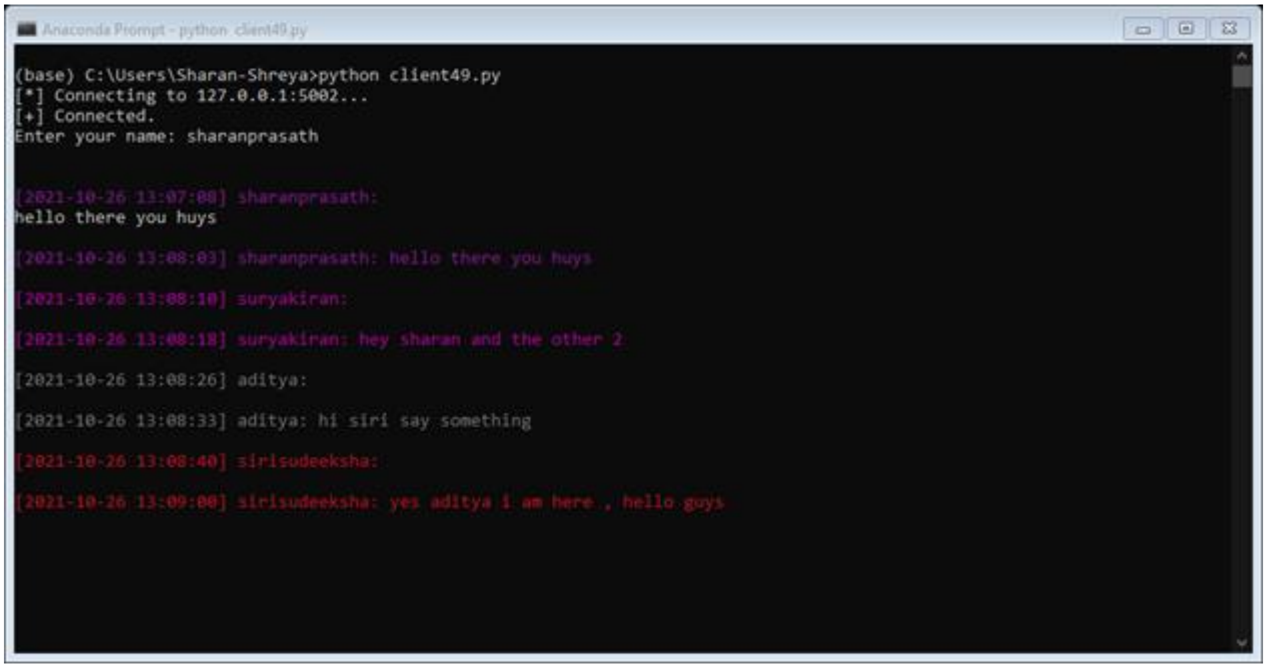
Figure c



```
(base) C:\Users\Sharan-Shreya>python server49.py
[*] Listening as 0.0.0.0:5002
[+] ('127.0.0.1', 60155) connected.
```

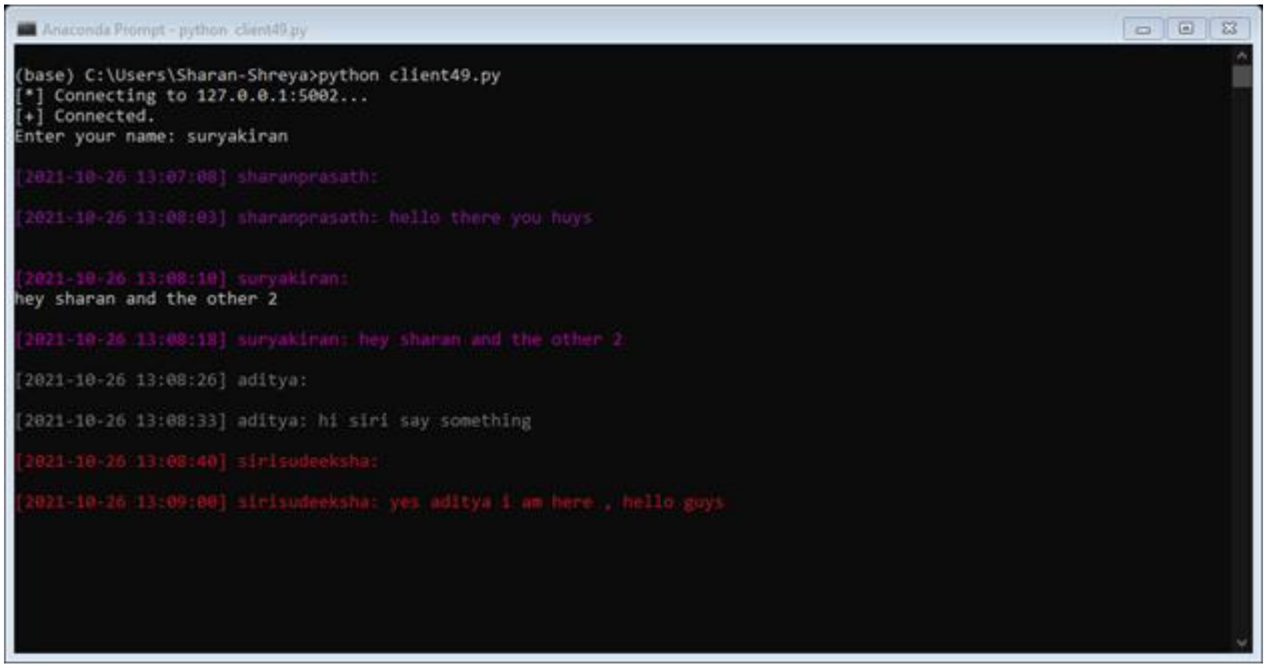
Chat Room:

CLIENT 1:



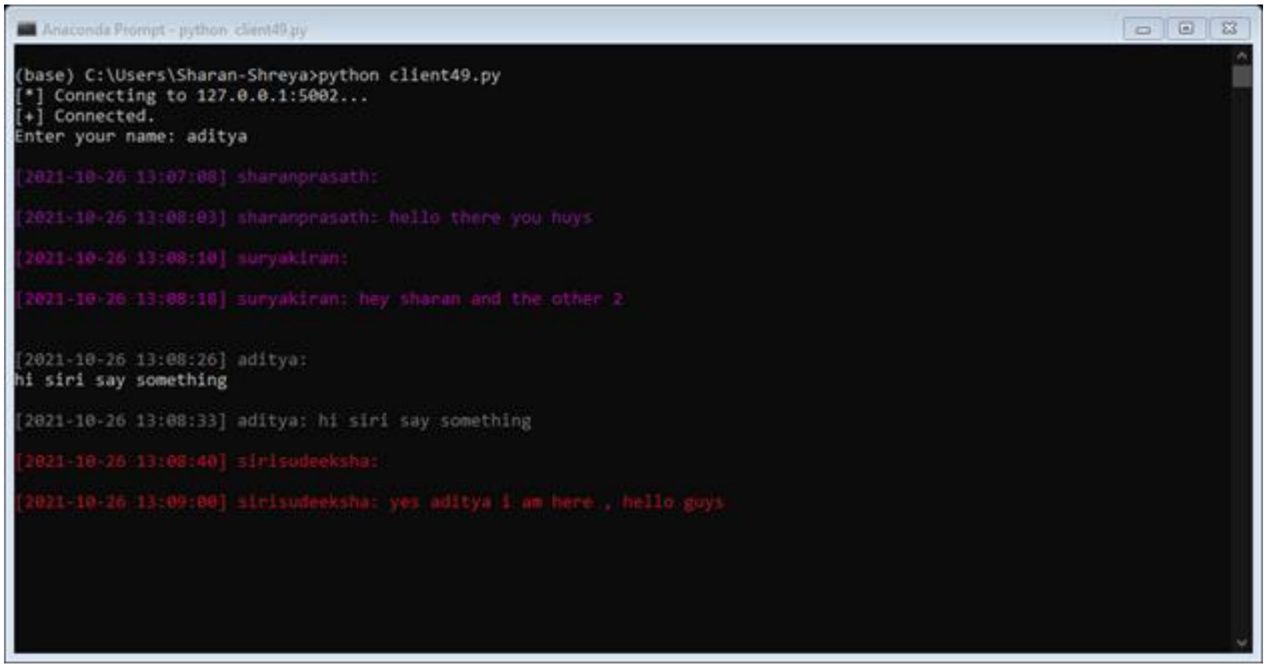
```
(base) C:\Users\Sharan-Shreya>python client49.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: sharanprasath


[2021-10-26 13:07:00] sharanprasath:
hello there you huys

[2021-10-26 13:08:03] sharanprasath: hello there you huys

[2021-10-26 13:08:10] suryakiran:

[2021-10-26 13:08:18] suryakiran: hey sharan and the other 2

[2021-10-26 13:08:26] aditya:

[2021-10-26 13:08:33] aditya: hi siri say something

[2021-10-26 13:08:40] sirisudeeksha:

[2021-10-26 13:09:00] sirisudeeksha: yes aditya i am here , hello guys
```

CLIENT 2:



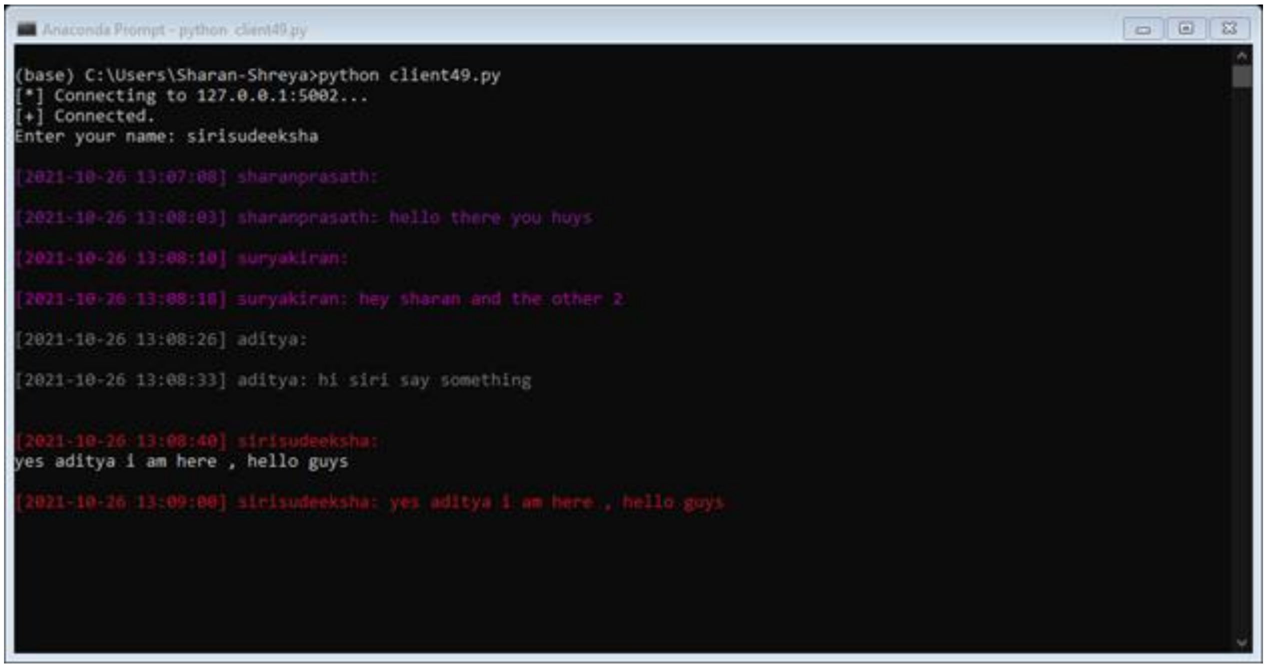
```
(base) C:\Users\Sharan-Shreya>python client49.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: suryakiran

[2021-10-26 13:07:08] sharanprasath:

[2021-10-26 13:08:03] sharanprasath: hello there you huys


[2021-10-26 13:08:10] suryakiran:
hey sharan and the other 2

[2021-10-26 13:08:18] suryakiran: hey sharan and the other 2

[2021-10-26 13:08:26] aditya:

[2021-10-26 13:08:33] aditya: hi siri say something

[2021-10-26 13:08:40] sirisudeeksha:

[2021-10-26 13:09:00] sirisudeeksha: yes aditya i am here , hello guys
```

CLIENT 3:



```
(base) C:\Users\Sharan-Shreya>python client49.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: aditya

[2021-10-26 13:07:08] sharanprasath:

[2021-10-26 13:08:03] sharanprasath: hello there you huys

[2021-10-26 13:08:10] suryakiran:

[2021-10-26 13:08:18] suryakiran: hey sharan and the other 2


[2021-10-26 13:08:26] aditya:
hi siri say something

[2021-10-26 13:08:33] aditya: hi siri say something

[2021-10-26 13:08:40] sirisudeeksha:

[2021-10-26 13:09:00] sirisudeeksha: yes aditya i am here , hello guys
```

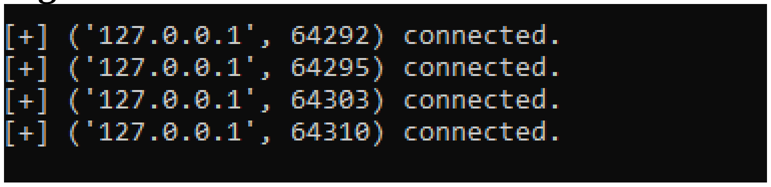
CLIENT 4:



```
(base) C:\Users\Sharan-Shreya>python client49.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: sirisudeeksha

[2021-10-26 13:07:08] sharanprasath:

[2021-10-26 13:08:03] sharanprasath: hello there you huys

[2021-10-26 13:08:10] suryakiran:

[2021-10-26 13:08:18] suryakiran: hey sharan and the other 2

[2021-10-26 13:08:26] aditya:

[2021-10-26 13:08:33] aditya: hi siri say something


[2021-10-26 13:08:40] sirisudeeksha:
yes aditya i am here , hello guys

[2021-10-26 13:09:00] sirisudeeksha: yes aditya i am here , hello guys
```

Now at server side:

Figure D:



```
[+] ('127.0.0.1', 64292) connected.
[+] ('127.0.0.1', 64295) connected.
[+] ('127.0.0.1', 64303) connected.
[+] ('127.0.0.1', 64310) connected.
```

2. Results Analysis:

- Alright, now that both code for the server and the client is completed,the demonstration was done above.
- Firstly, run one and only one server instance as shown in **figure a.**
- the server is listening for upcoming client connections,try to run one client instance as shown in **figure b**

- Now the client is connected to the server and prompted for a username, to make sure it's connected, get back to the server console and you'll see indeed it's connected as shown in **figure c or D.**

- Note we're on localhost (`127.0.0.1`) address for now, as it's the same machine, but if you want to connect from other machines in the same network, you can do that as well, just make sure to change `SERVER_HOST` in client code from `127.0.0.1` to the server's private IP address.

- Let's run another client so we can chat:as shown in Client one, two , three, and four in the image above.

- As you can see, each client have a color so we can distinguish between users

3. Conclusion :

- Now after the Experiment or project we have build a Chat room application that accepts multiple connected clients using built-in's sockets and threading libraries in Python.

- Every message sent from a particular client is sent to all other clients. Note the colors are changed whenever you re-execute the `client.py` script.

- Also Every time a client sends a message in the chat room a notification sound is sent so that other clients may know there is a new message sent by another client.

- This Experiment explains exactly how a socket programming works with server and multiple clients.

7. References:

GeekforGeeks,W3schools,Python Documentation.