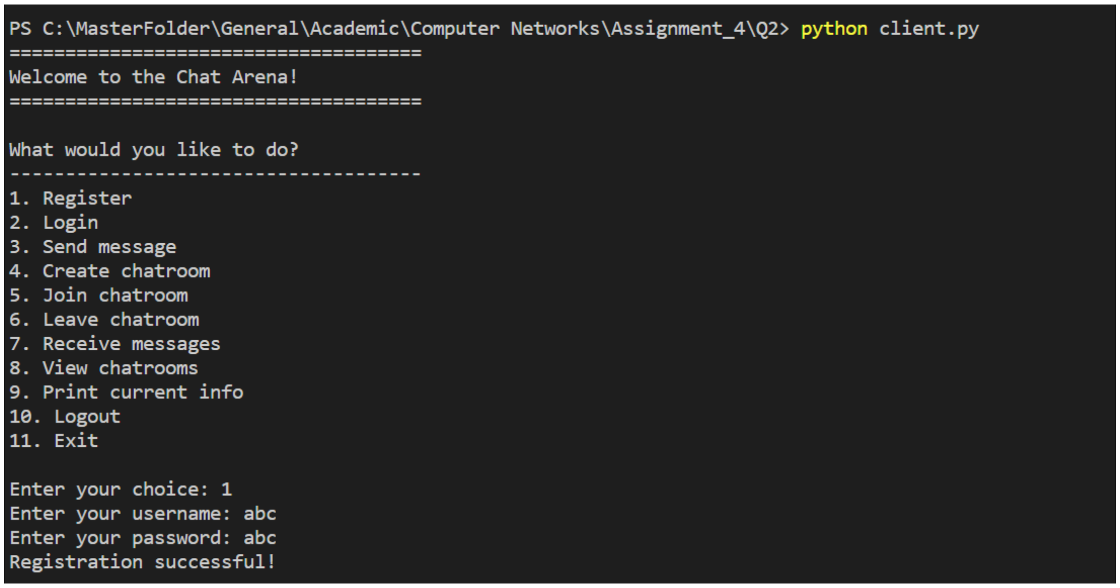
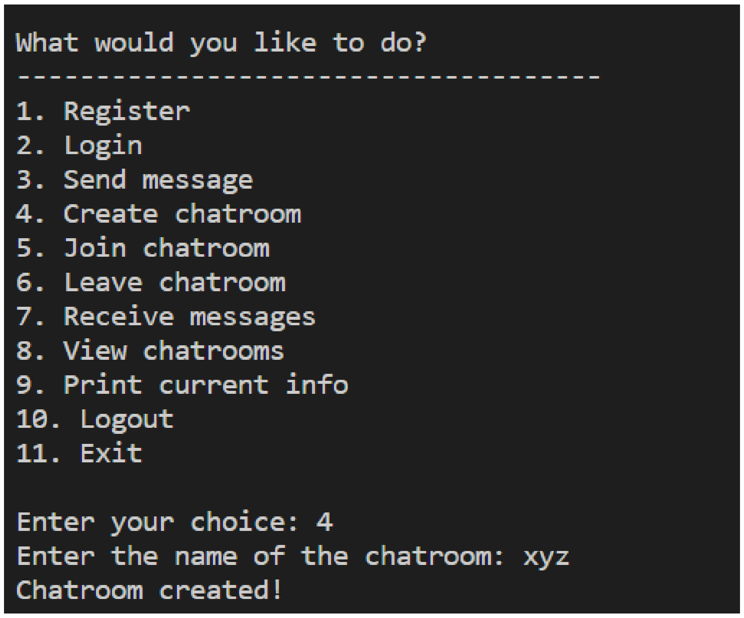
.Chat Interface:



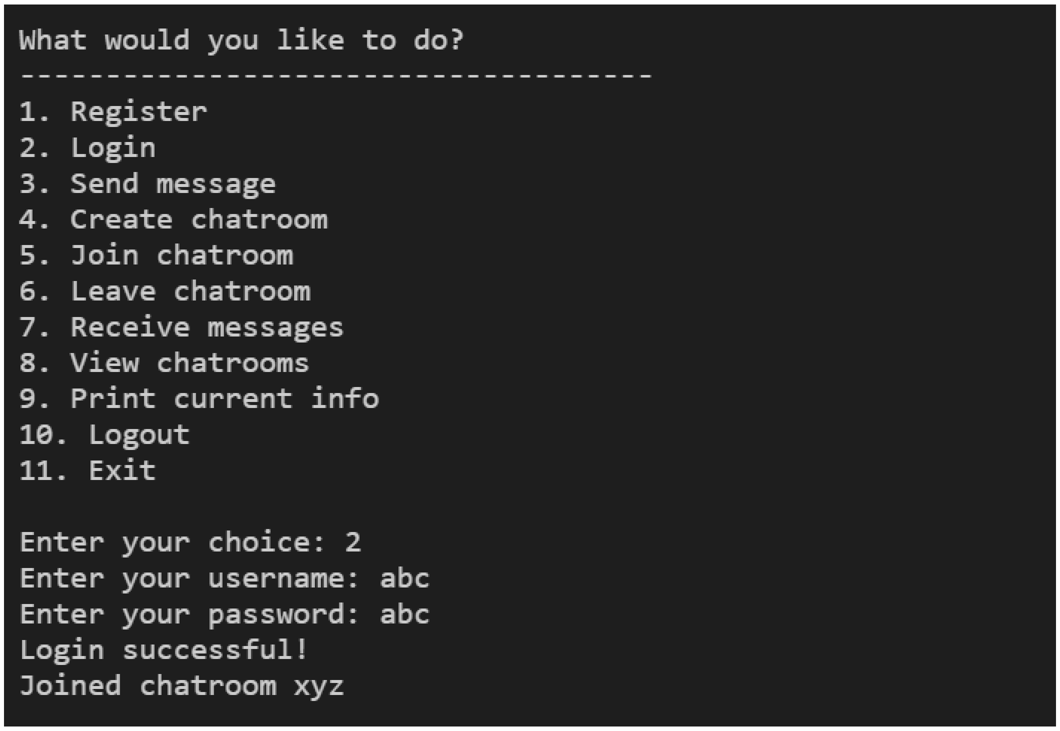
```
PS C:\MasterFolder\General\Academic\Computer Networks\Assignment_4\Q2> python client.py
=====================================
Welcome to the Chat Arena!
=====================================

What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 1
Enter your username: abc
Enter your password: abc
Registration successful!
```

Client side interface on registration



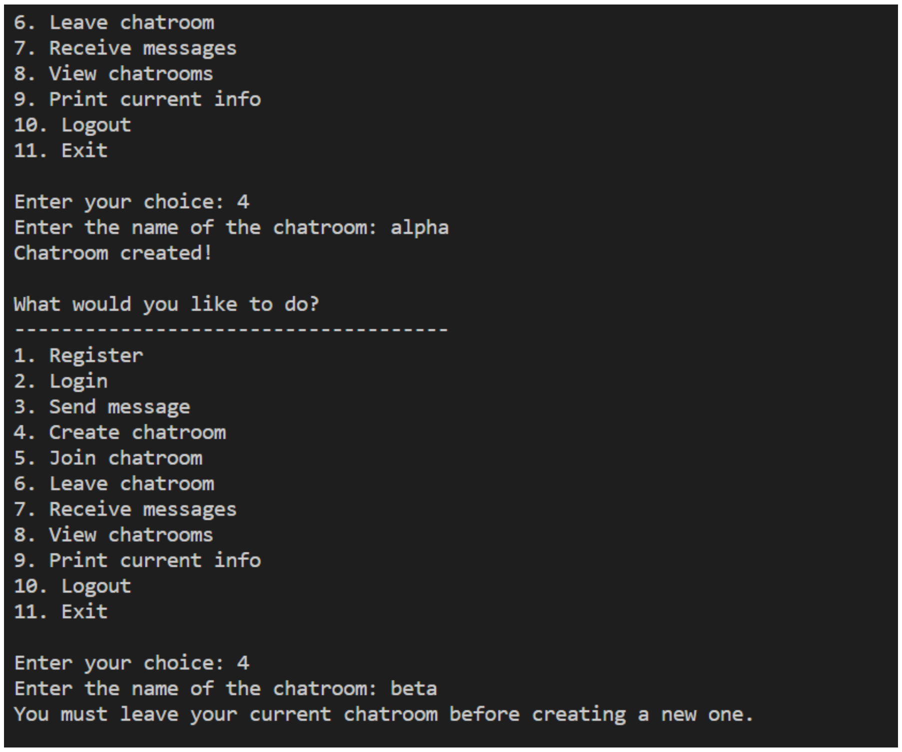
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 4
Enter the name of the chatroom: xyz
Chatroom created!
```

New chat room creation



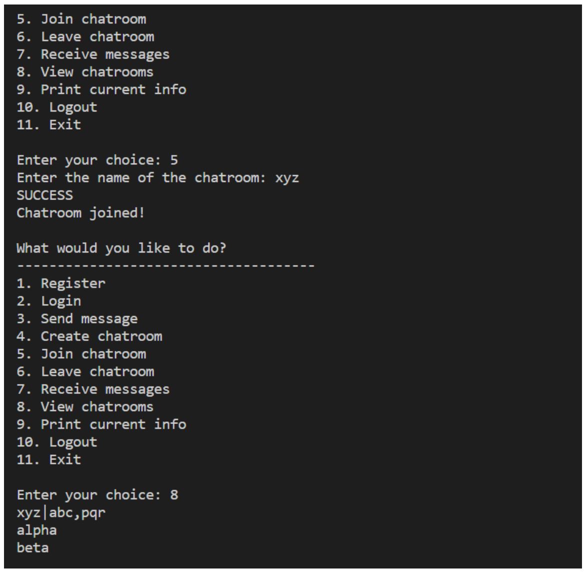
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 2
Enter your username: abc
Enter your password: abc
Login successful!
Joined chatroom xyz
```

Client abc logged in from another terminal



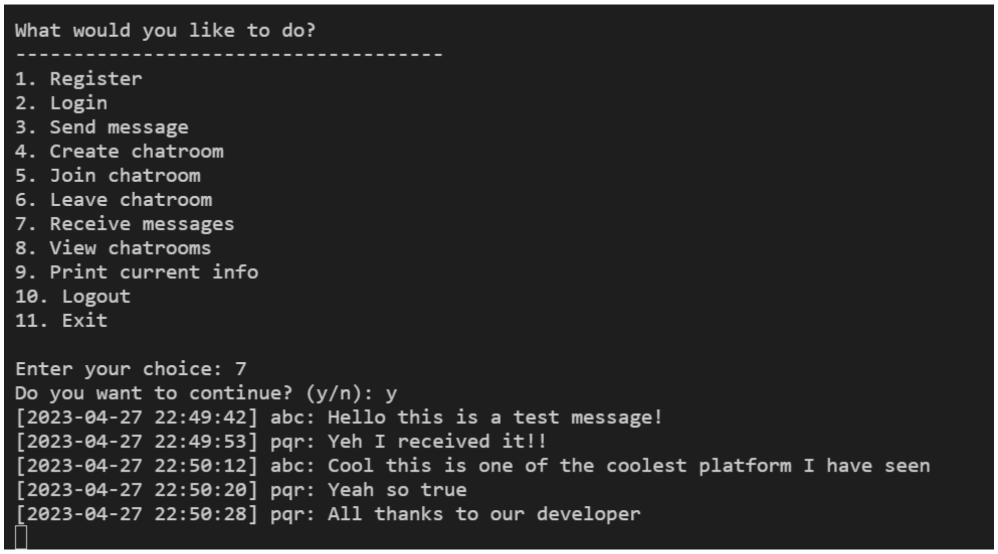
```
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 4
Enter the name of the chatroom: alpha
Chatroom created!

What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 4
Enter the name of the chatroom: beta
You must leave your current chatroom before creating a new one.
```

Leaving the chatroom before creating another



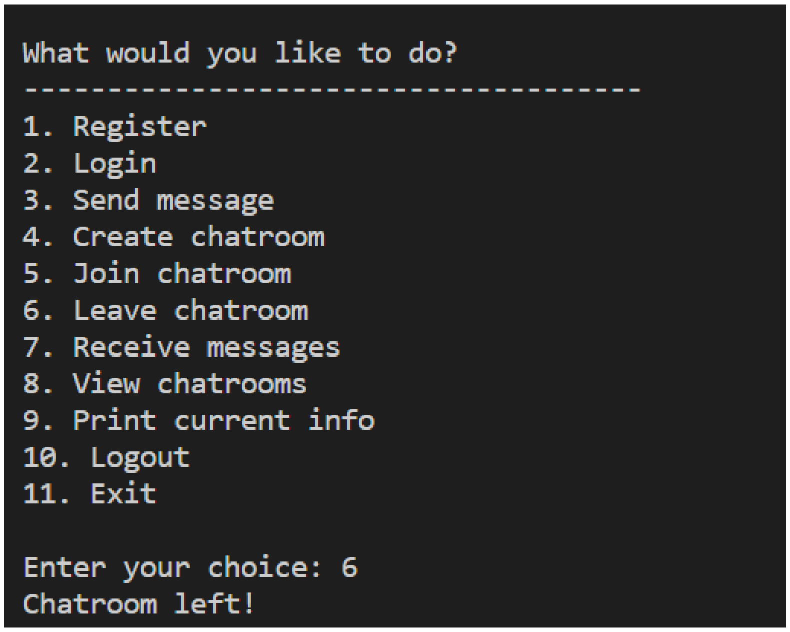
```
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 5
Enter the name of the chatroom: xyz
SUCCESS
Chatroom joined!

What would you like to do?
------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 8
xyz|abc,pqr
alpha
beta
```

View current chatrooms and participants in it



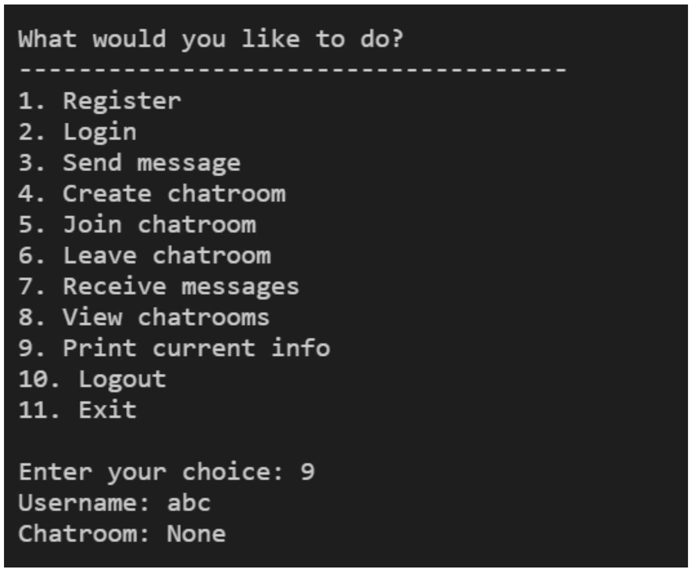
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 7
Do you want to continue? (y/n): y
[2023-04-27 22:49:42] abc: Hello this is a test message!
[2023-04-27 22:49:53] pqr: Yeh I received it!!
[2023-04-27 22:50:12] abc: Cool this is one of the coolest platform I have seen
[2023-04-27 22:50:20] pqr: Yeah so true
[2023-04-27 22:50:28] pqr: All thanks to our developer
```

Client abc receiving messages in real-time, also there is a timeout of 1 min, after which it will be prompted whether to continue or not



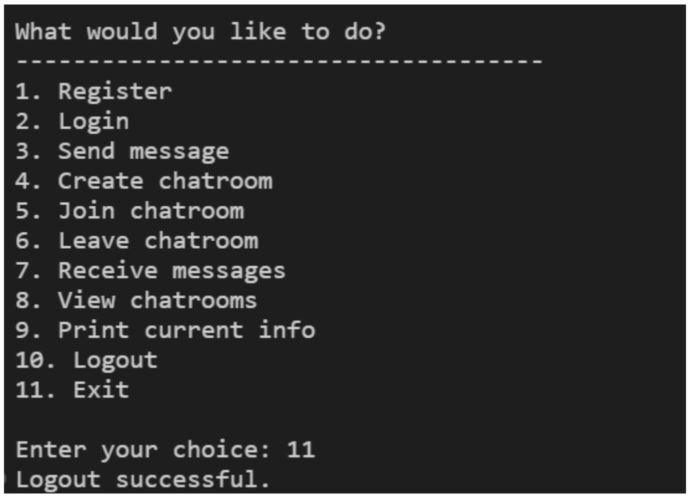
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 6
Chatroom left!
```

Chatroom left



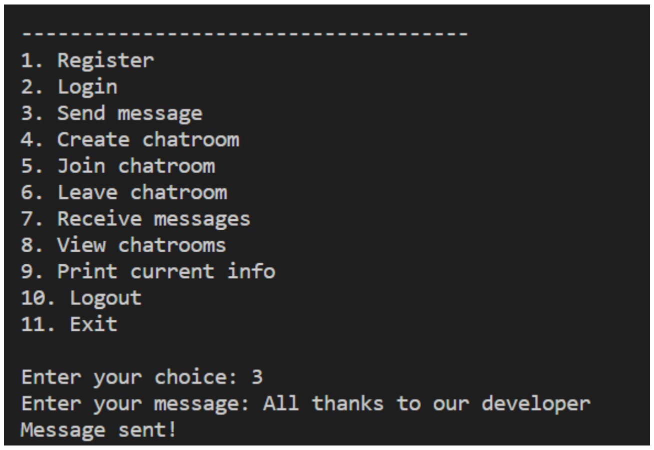
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 9
Username: abc
Chatroom: None
```

After leaving the chatroom, this is reflected in other terminal as well



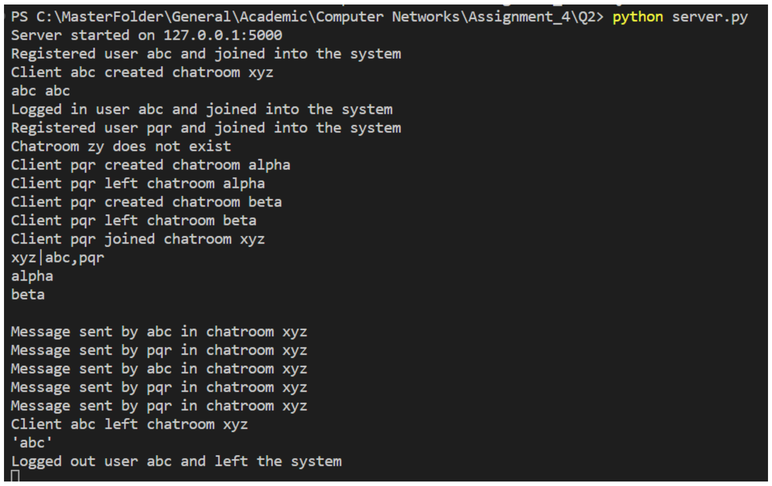
```
What would you like to do?
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 11
Logout successful.
```

Logout from the application



```
-------------------------------------
1. Register
2. Login
3. Send message
4. Create chatroom
5. Join chatroom
6. Leave chatroom
7. Receive messages
8. View chatrooms
9. Print current info
10. Logout
11. Exit

Enter your choice: 3
Enter your message: All thanks to our developer
Message sent!
```

Message sent!!



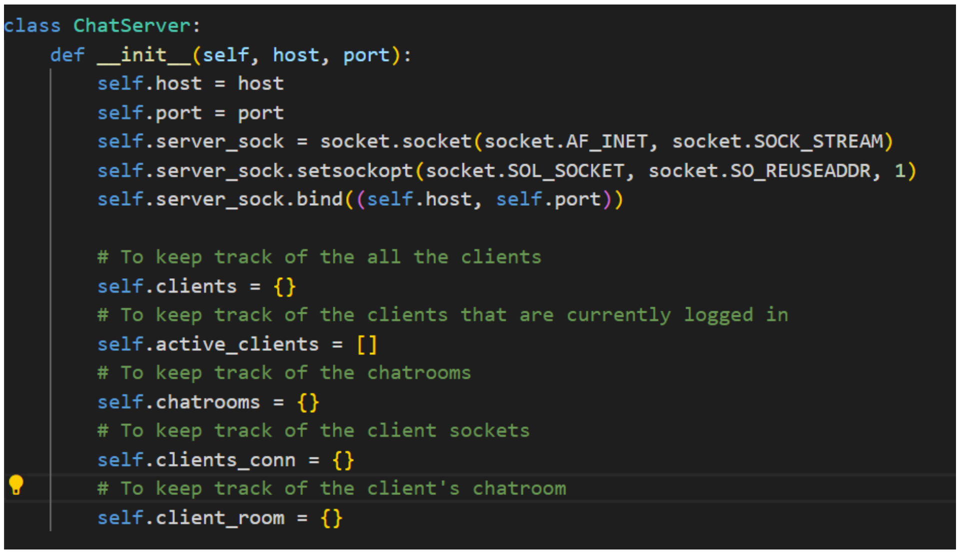
```
PS C:\MasterFolder\General\Academic\Computer Networks\Assignment_4\Q2> python server.py
Server started on 127.0.0.1:5000
Registered user abc and joined into the system
Client abc created chatroom xyz
abc abc
Logged in user abc and joined into the system
Registered user pqr and joined into the system
Chatroom zy does not exist
Client pqr created chatroom alpha
Client pqr left chatroom alpha
Client pqr created chatroom beta
Client pqr left chatroom beta
Client pqr joined chatroom xyz
xyz|abc,pqr
alpha
beta

Message sent by abc in chatroom xyz
Message sent by pqr in chatroom xyz
Message sent by abc in chatroom xyz
Message sent by pqr in chatroom xyz
Message sent by pqr in chatroom xyz
Client abc left chatroom xyz
'abc'
Logged out user abc and left the system
```

Server Logs telling about the various events

Server Side Code:

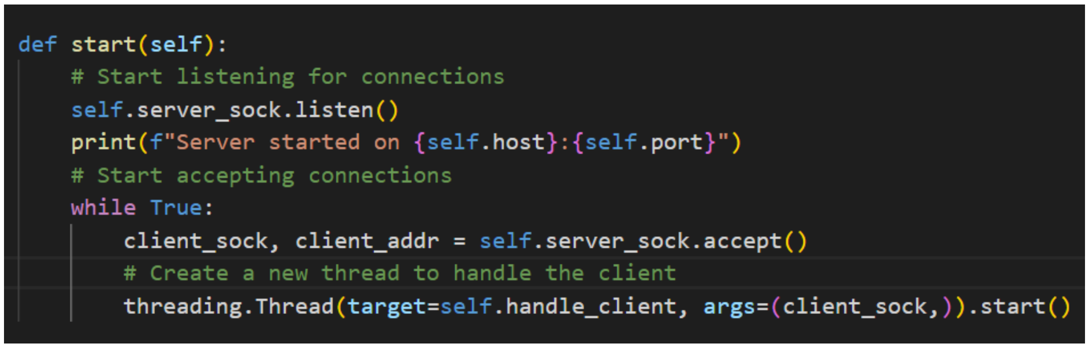
- ➢ The ChatServer class is a Python class that uses the socket module to create a chat server.

- ➢ It has an init method that takes two arguments - a host and a port number. These are used to create a socket object with the socket.AF_INET and socket.SOCK_STREAM parameters, which sets up a TCP/IP socket connection.



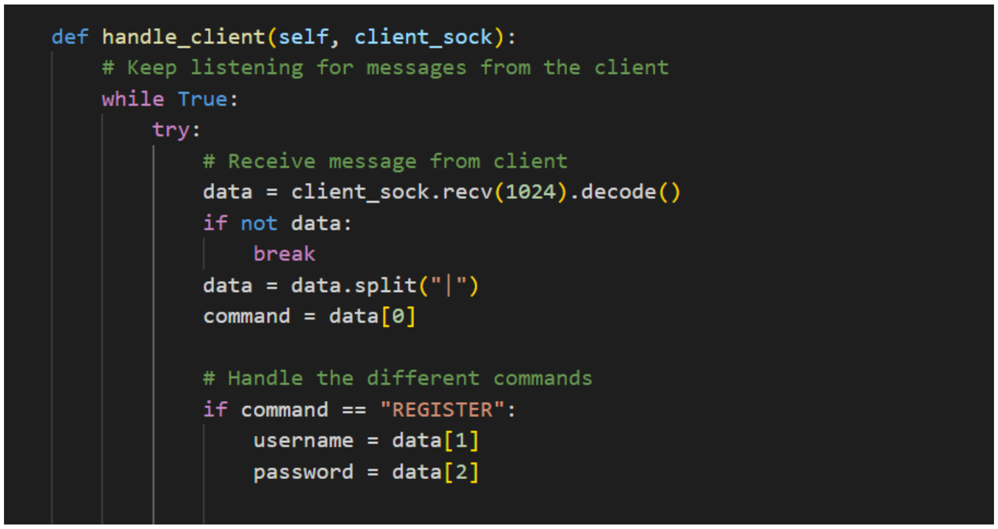
```python
class ChatServer:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_sock.bind((self.host, self.port))

        # To keep track of the all the clients
        self.clients = {}
        # To keep track of the clients that are currently logged in
        self.active_clients = []
        # To keep track of the chatrooms
        self.chatrooms = {}
        # To keep track of the client sockets
        self.clients_conn = {}
        # To keep track of the client's chatroom
        self.client_room = {}
```

- ➢ The class also initializes several instance variables, including a dictionary of registered clients (self.clients), a list of currently active clients (self.active_clients), a dictionary of chatrooms (self.chatrooms), a dictionary of client connections (self.clients_conn), and a dictionary of client-to-chatroom mappings (self.client_room).



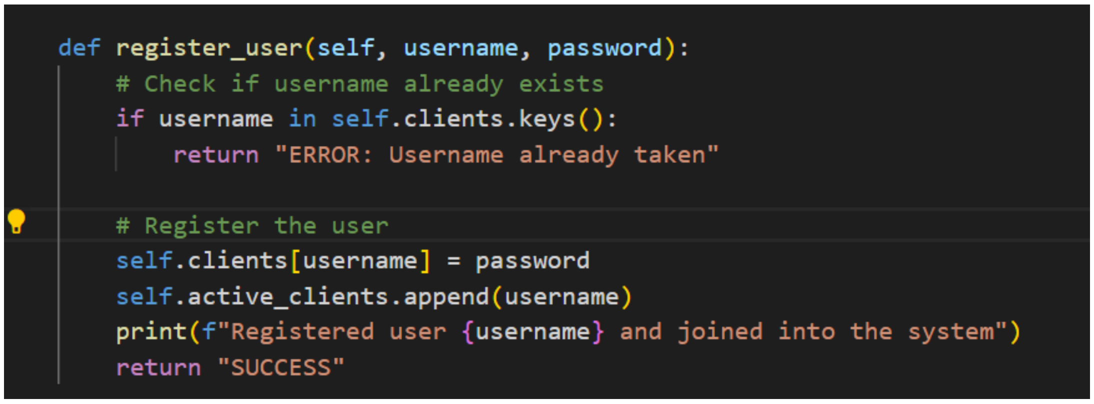
```python
def start(self):
    # Start listening for connections
    self.server_sock.listen()
    print(f"Server started on {self.host}:{self.port}")
    # Start accepting connections
    while True:
        client_sock, client_addr = self.server_sock.accept()
        # Create a new thread to handle the client
        threading.Thread(target=self.handle_client, args=(client_sock,)).start()
```

- ➢ The **start method** listens for incoming client connections and starts a new thread to handle each connection. When a new client connects, the handle_client method is called with the client socket as an argument.



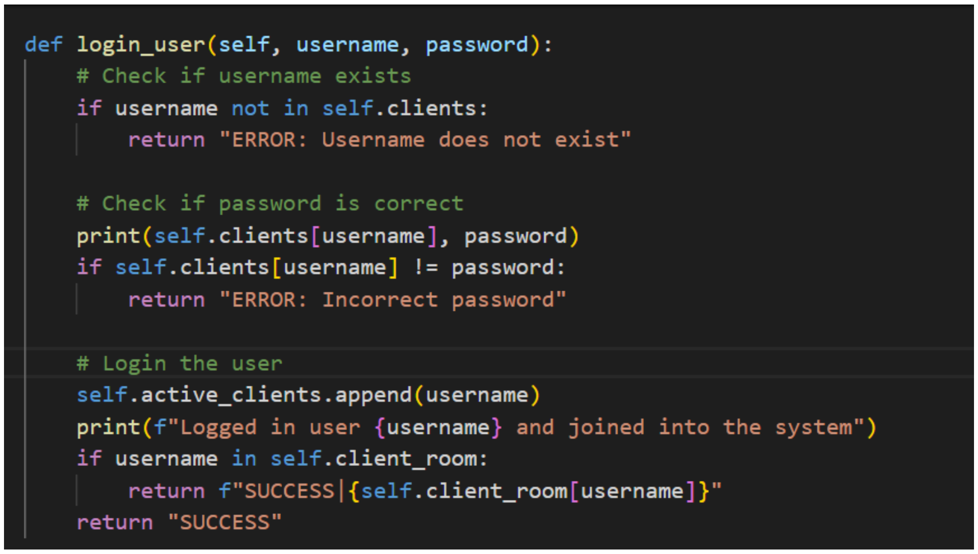
```python
def handle_client(self, client_sock):
    # Keep listening for messages from the client
    while True:
        try:
            # Receive message from client
            data = client_sock.recv(1024).decode()
            if not data:
                break
            data = data.split("|")
            command = data[0]

            # Handle the different commands
            if command == "REGISTER":
                username = data[1]
                password = data[2]
```

➢ The **handle_client** method is responsible for processing client messages and updating the server state accordingly. It does this by receiving data from the client socket, parsing the command, and calling the appropriate method to handle the command.



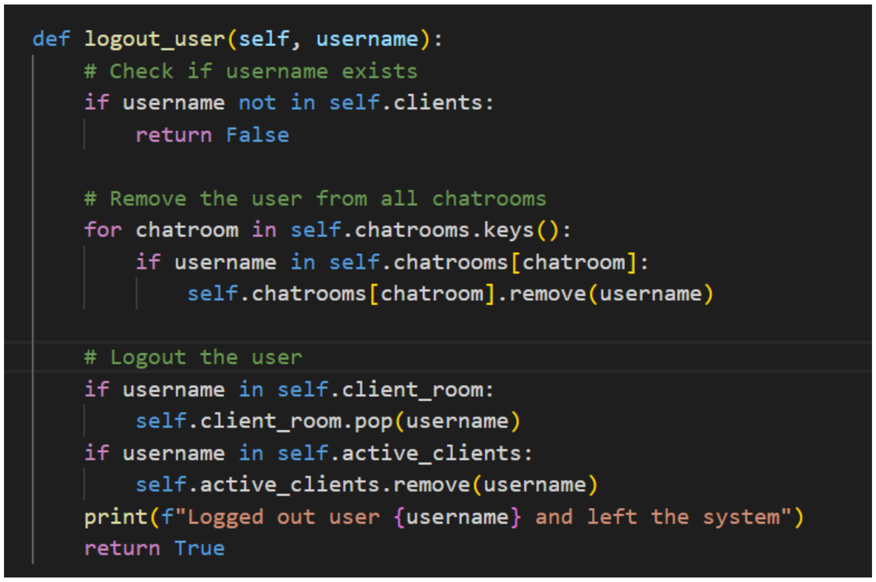
```python
def register_user(self, username, password):
    # Check if username already exists
    if username in self.clients.keys():
        return "ERROR: Username already taken"

    # Register the user
    self.clients[username] = password
    self.active_clients.append(username)
    print(f"Registered user {username} and joined into the system")
    return "SUCCESS"
```

➢ The **register_user** method takes a username and password and adds the user to the self.clients dictionary. If the username is already taken, it returns an error message. If the registration is successful, it adds the username to the self.active_clients list.



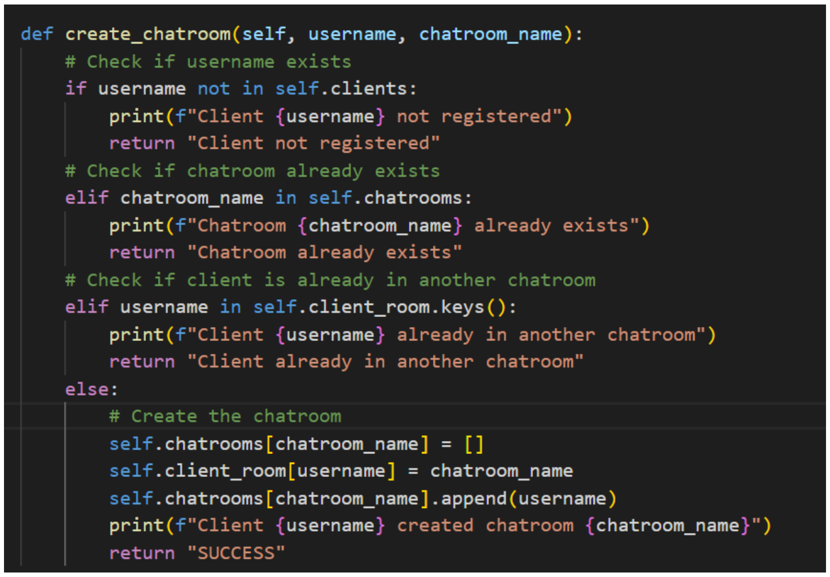
```python
def login_user(self, username, password):
    # Check if username exists
    if username not in self.clients:
        return "ERROR: Username does not exist"

    # Check if password is correct
    print(self.clients[username], password)
    if self.clients[username] != password:
        return "ERROR: Incorrect password"

    # Login the user
    self.active_clients.append(username)
    print(f"Logged in user {username} and joined into the system")
    if username in self.client_room:
        return f"SUCCESS|{self.client_room[username]}"
    return "SUCCESS"
```

➢ The **login_user** method takes a username and password and checks if they match an existing user in the self.clients dictionary. If there is a match, it adds the user to the self.active_clients list and returns a success message. If there is no match, it returns an error message.



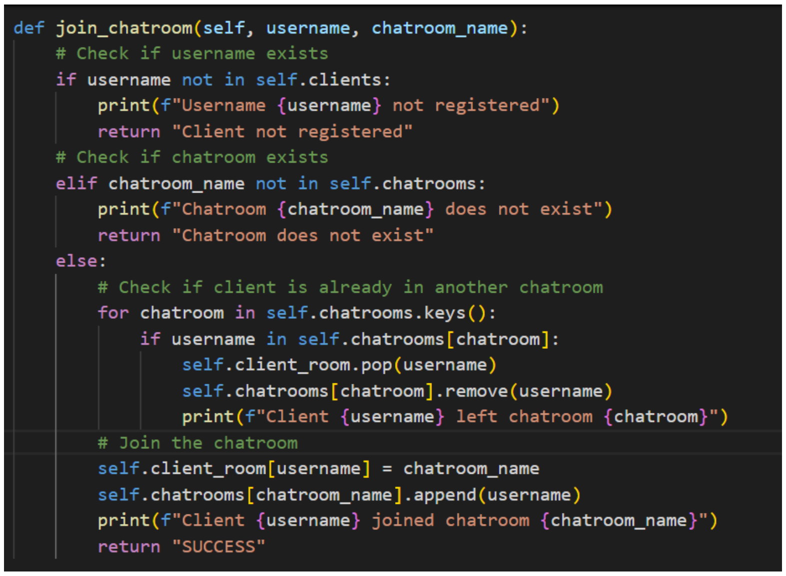
```python
def logout_user(self, username):
    # Check if username exists
    if username not in self.clients:
        return False

    # Remove the user from all chatrooms
    for chatroom in self.chatrooms.keys():
        if username in self.chatrooms[chatroom]:
            self.chatrooms[chatroom].remove(username)

    # Logout the user
    if username in self.client_room:
        self.client_room.pop(username)
    if username in self.active_clients:
        self.active_clients.remove(username)
    print(f"Logged out user {username} and left the system")
    return True
```

➢ The **logout_user** method removes a user from the self.active_clients list and removes them from any chatrooms they are in. If the user is not logged in, it returns an error message.



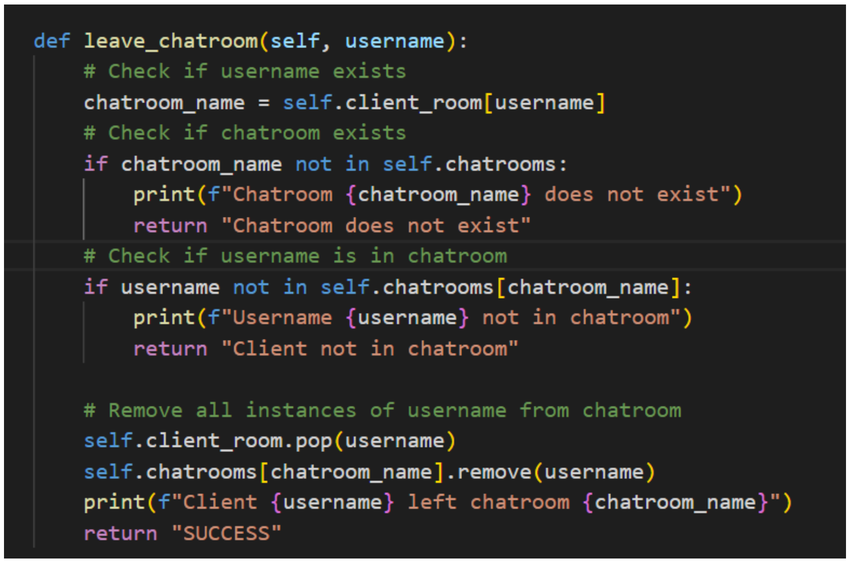
```python
def create_chatroom(self, username, chatroom_name):
    # Check if username exists
    if username not in self.clients:
        print(f"Client {username} not registered")
        return "Client not registered"
    # Check if chatroom already exists
    elif chatroom_name in self.chatrooms:
        print(f"Chatroom {chatroom_name} already exists")
        return "Chatroom already exists"
    # Check if client is already in another chatroom
    elif username in self.client_room.keys():
        print(f"Client {username} already in another chatroom")
        return "Client already in another chatroom"
    else:
        # Create the chatroom
        self.chatrooms[chatroom_name] = []
        self.client_room[username] = chatroom_name
        self.chatrooms[chatroom_name].append(username)
        print(f"Client {username} created chatroom {chatroom_name}")
        return "SUCCESS"
```

➢ The **create_chatroom** method creates a new chatroom with a given name and adds it to the self.chatrooms dictionary. If the chatroom name already exists, it returns an error message. If the user is already in another chatroom or not registered, it returns an error message. Otherwise, it adds the user to the chatroom and updates the self.client_room dictionary.

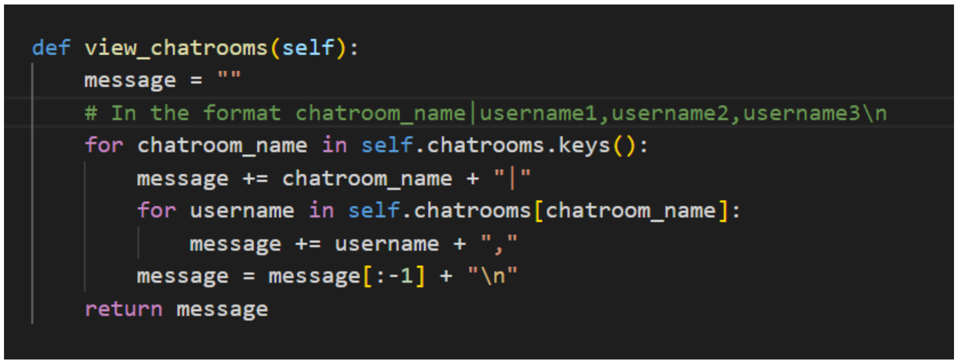


```python
def join_chatroom(self, username, chatroom_name):
    # Check if username exists
    if username not in self.clients:
        print(f"Username {username} not registered")
        return "Client not registered"
    # Check if chatroom exists
    elif chatroom_name not in self.chatrooms:
        print(f"Chatroom {chatroom_name} does not exist")
        return "Chatroom does not exist"
    else:
        # Check if client is already in another chatroom
        for chatroom in self.chatrooms.keys():
            if username in self.chatrooms[chatroom]:
                self.client_room.pop(username)
                self.chatrooms[chatroom].remove(username)
                print(f"Client {username} left chatroom {chatroom}")
        # Join the chatroom
        self.client_room[username] = chatroom_name
        self.chatrooms[chatroom_name].append(username)
        print(f"Client {username} joined chatroom {chatroom_name}")
        return "SUCCESS"
```

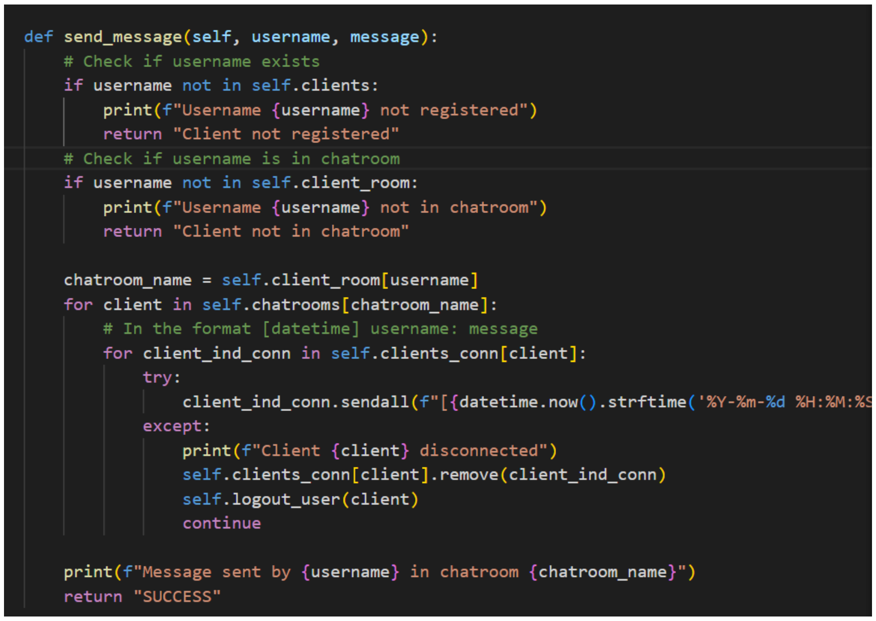
➢ The **join_chatroom** method adds a user to an existing chatroom. If the chatroom does not exist or the user is already in another chatroom or not registered, it returns an error message. Otherwise, it adds the user to the chatroom and updates the self.client_room dictionary.


```python
def leave_chatroom(self, username):
    # Check if username exists
    chatroom_name = self.client_room[username]
    # Check if chatroom exists
    if chatroom_name not in self.chatrooms:
        print(f"Chatroom {chatroom_name} does not exist")
        return "Chatroom does not exist"
    # Check if username is in chatroom
    if username not in self.chatrooms[chatroom_name]:
        print(f"Username {username} not in chatroom")
        return "Client not in chatroom"

    # Remove all instances of username from chatroom
    self.client_room.pop(username)
    self.chatrooms[chatroom_name].remove(username)
    print(f"Client {username} left chatroom {chatroom_name}")
    return "SUCCESS"
```

➢ The **leave_chatroom** method removes a user from a chatroom. If the user is not in a chatroom or not registered, it returns an error message. Otherwise, it removes the user from the chatroom and updates the self.client_room dictionary.

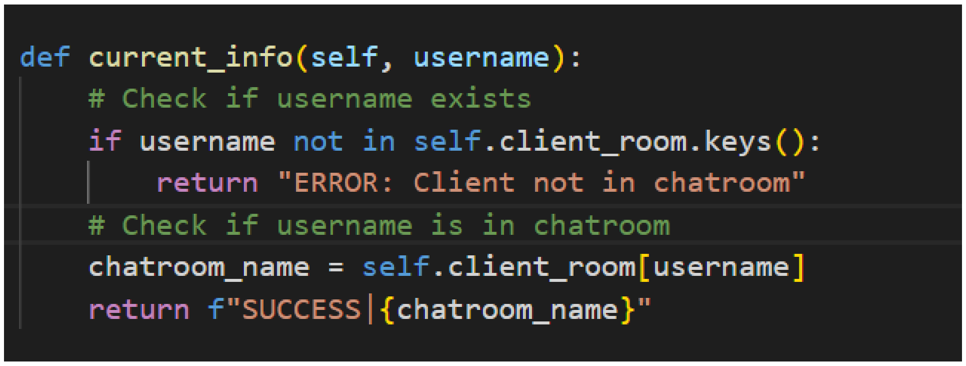

```python
def view_chatrooms(self):
    message = ""
    # In the format chatroom_name|username1,username2,username3\n
    for chatroom_name in self.chatrooms.keys():
        message += chatroom_name + "|"
        for username in self.chatrooms[chatroom_name]:
            message += username + ","
        message = message[:-1] + "\n"
    return message
```

➢ The **view_chatrooms** method returns a list of all chatrooms in the self.chatrooms dictionary.



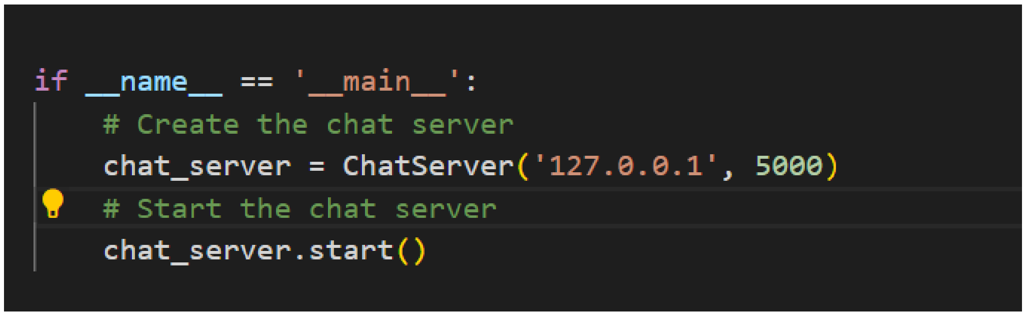
```python
def send_message(self, username, message):
    # Check if username exists
    if username not in self.clients:
        print(f"Username {username} not registered")
        return "Client not registered"
    # Check if username is in chatroom
    if username not in self.client_room:
        print(f"Username {username} not in chatroom")
        return "Client not in chatroom"

    chatroom_name = self.client_room[username]
    for client in self.chatrooms[chatroom_name]:
        # In the format [datetime] username: message
        for client_ind_conn in self.clients_conn[client]:
            try:
                client_ind_conn.sendall(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S
            except:
                print(f"Client {client} disconnected")
                self.clients_conn[client].remove(client_ind_conn)
                self.logout_user(client)
                continue

    print(f"Message sent by {username} in chatroom {chatroom_name}")
    return "SUCCESS"
```

➢ The **send_message** method takes a username and a message and sends the message to all clients in the same chatroom as the user. If the user is not in a chatroom or not registered, it returns an error message.



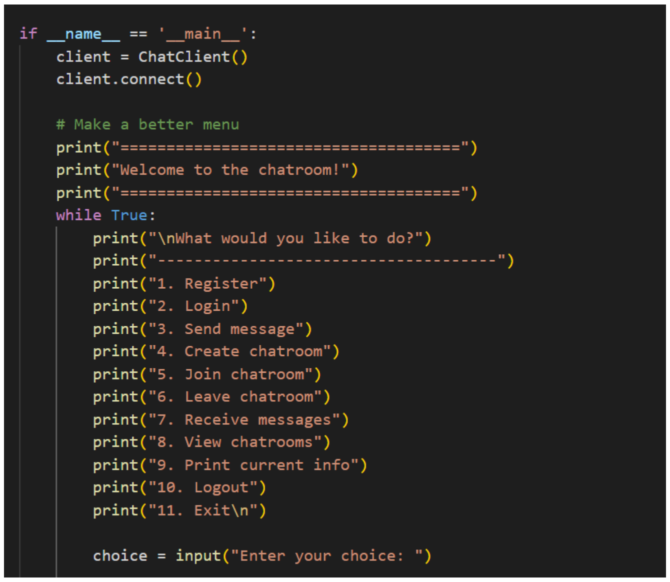
```python
def current_info(self, username):
    # Check if username exists
    if username not in self.client_room.keys():
        return "ERROR: Client not in chatroom"
    # Check if username is in chatroom
    chatroom_name = self.client_room[username]
    return f"SUCCESS|{chatroom_name}"
```

➢ The **current_info** method takes a username and returns the chatroom that the user is currently in, if any. If the user is not in a chatroom or not registered, it returns an error message.



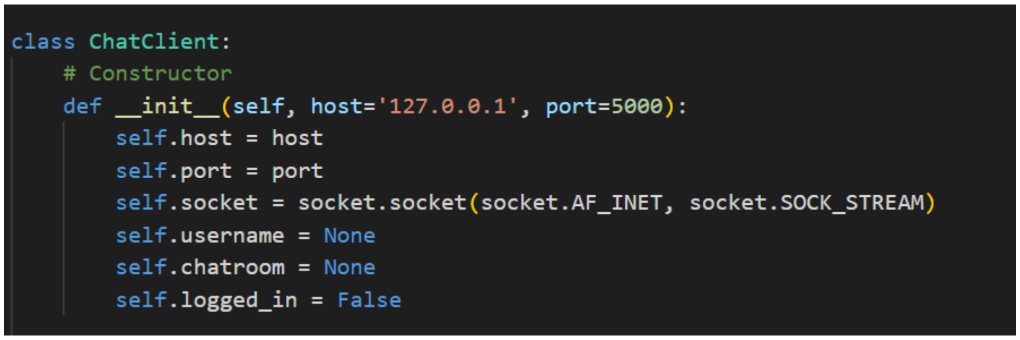
```python
if __name__ == '__main__':
    # Create the chat server
    chat_server = ChatServer('127.0.0.1', 5000)
    # Start the chat server
    chat_server.start()
```

Client Side Code:



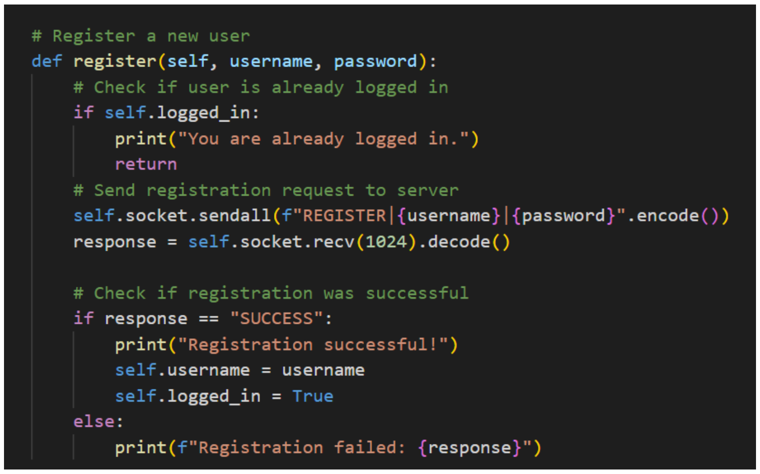
```python
if __name__ == '__main__':
    client = ChatClient()
    client.connect()

    # Make a better menu
    print("====================================")
    print("Welcome to the chatroom!")
    print("====================================")
    while True:
        print("\nWhat would you like to do?")
        print("------------------------------------")
        print("1. Register")
        print("2. Login")
        print("3. Send message")
        print("4. Create chatroom")
        print("5. Join chatroom")
        print("6. Leave chatroom")
        print("7. Receive messages")
        print("8. View chatrooms")
        print("9. Print current info")
        print("10. Logout")
        print("11. Exit\n")

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

➢ The ChatClient class is a Python class that uses the socket module to create a client that can connect to a chat server.



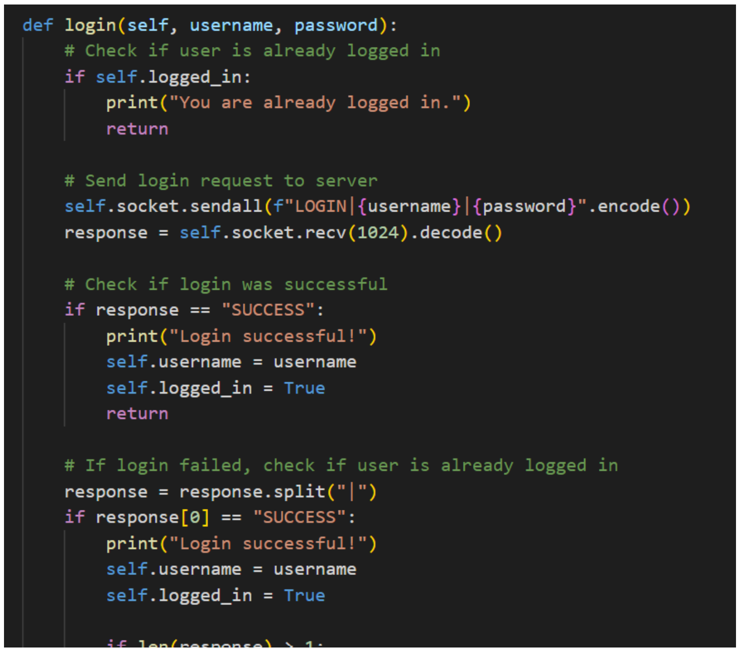
```python
class ChatClient:
    # Constructor
    def __init__(self, host='127.0.0.1', port=5000):
        self.host = host
        self.port = port
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.username = None
        self.chatroom = None
        self.logged_in = False
```

➢ It has an **init** method that takes two arguments - a host and a port number. These are used to create a socket object with the socket.AF_INET and socket.SOCK_STREAM parameters, which sets up a TCP/IP socket connection to the server.



```python
# Register a new user
def register(self, username, password):
    # Check if user is already logged in
    if self.logged_in:
        print("You are already logged in.")
        return
    # Send registration request to server
    self.socket.sendall(f"REGISTER|{username}|{password}".encode())
    response = self.socket.recv(1024).decode()

    # Check if registration was successful
    if response == "SUCCESS":
        print("Registration successful!")
        self.username = username
        self.logged_in = True
    else:
        print(f"Registration failed: {response}")
```

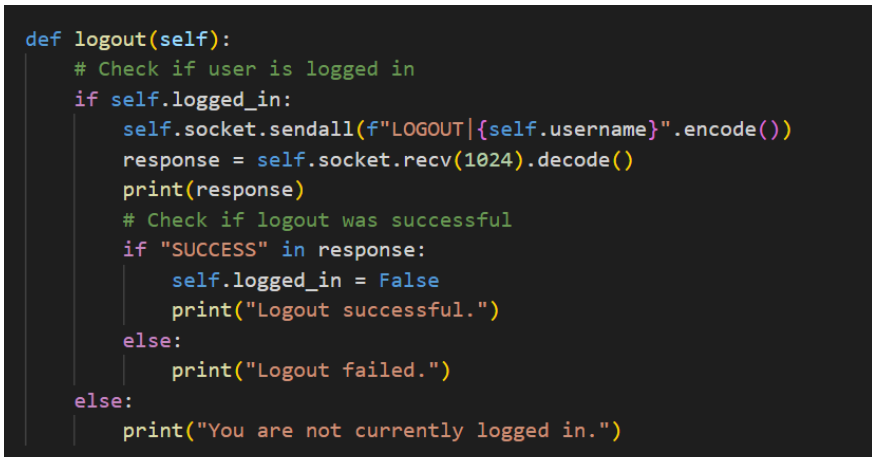
➢ The class also initializes several instance variables, including the client socket (self.client_socket), the username of the client (self.username), and the chatroom the client is currently in (self.chatroom).



```python
def login(self, username, password):
    # Check if user is already logged in
    if self.logged_in:
        print("You are already logged in.")
        return

    # Send login request to server
    self.socket.sendall(f"LOGIN|{username}|{password}".encode())
    response = self.socket.recv(1024).decode()

    # Check if login was successful
    if response == "SUCCESS":
        print("Login successful!")
        self.username = username
        self.logged_in = True
        return

    # If login failed, check if user is already logged in
    response = response.split("|")
    if response[0] == "SUCCESS":
        print("Login successful!")
        self.username = username
        self.logged_in = True

        if len(response) > 1:
```

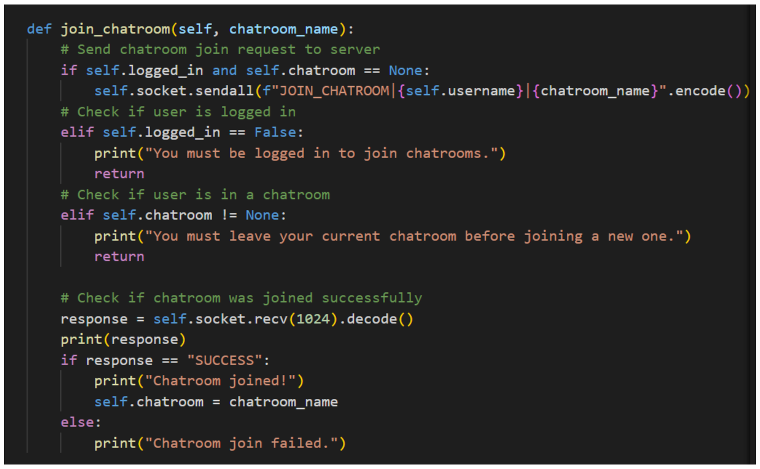
➢ The **connect method** connects the client socket to the server at the specified host and port number.

➢ The **login** method takes a username and password and sends them to the server to authenticate the user. If the authentication is successful, it sets the username and chatroom instance variables and returns a success message. If there is an error, it returns an error message.



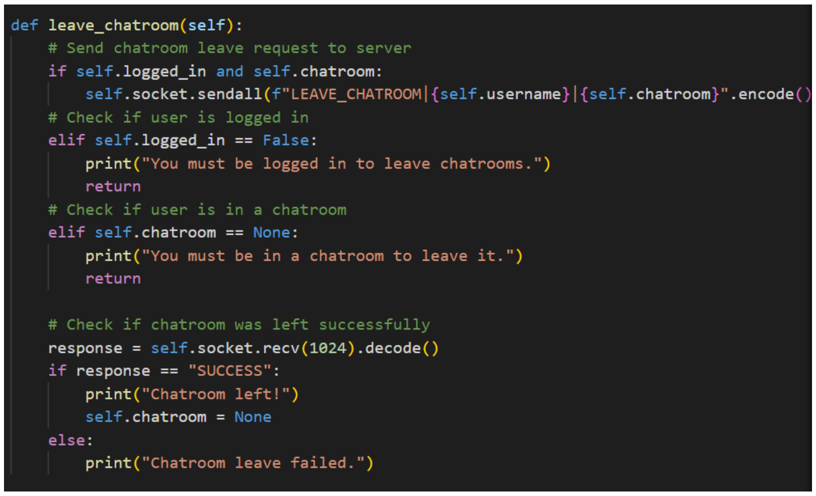
```python
def logout(self):
    # Check if user is logged in
    if self.logged_in:
        self.socket.sendall(f"LOGOUT|{self.username}".encode())
        response = self.socket.recv(1024).decode()
        print(response)
        # Check if logout was successful
        if "SUCCESS" in response:
            self.logged_in = False
            print("Logout successful.")
        else:
            print("Logout failed.")
    else:
        print("You are not currently logged in.")
```

➢ The **logout** method sends a message to the server to logout the user and closes the client socket.



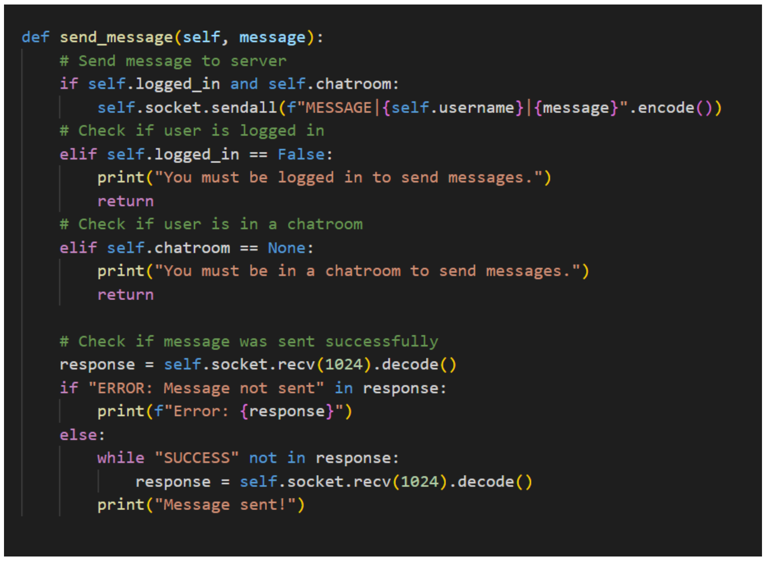
```python
def join_chatroom(self, chatroom_name):
    # Send chatroom join request to server
    if self.logged_in and self.chatroom == None:
        self.socket.sendall(f"JOIN_CHATROOM|{self.username}|{chatroom_name}".encode())
    # Check if user is logged in
    elif self.logged_in == False:
        print("You must be logged in to join chatrooms.")
        return
    # Check if user is in a chatroom
    elif self.chatroom != None:
        print("You must leave your current chatroom before joining a new one.")
        return

    # Check if chatroom was joined successfully
    response = self.socket.recv(1024).decode()
    print(response)
    if response == "SUCCESS":
        print("Chatroom joined!")
        self.chatroom = chatroom_name
    else:
        print("Chatroom join failed.")
```

➢ The **join_chatroom** method takes a chatroom name and sends a message to the server to add the user to the chatroom. If the chatroom name does not exist or the user is already in another chatroom, it returns an error message. Otherwise, it updates the chatroom instance variable and returns a success message.



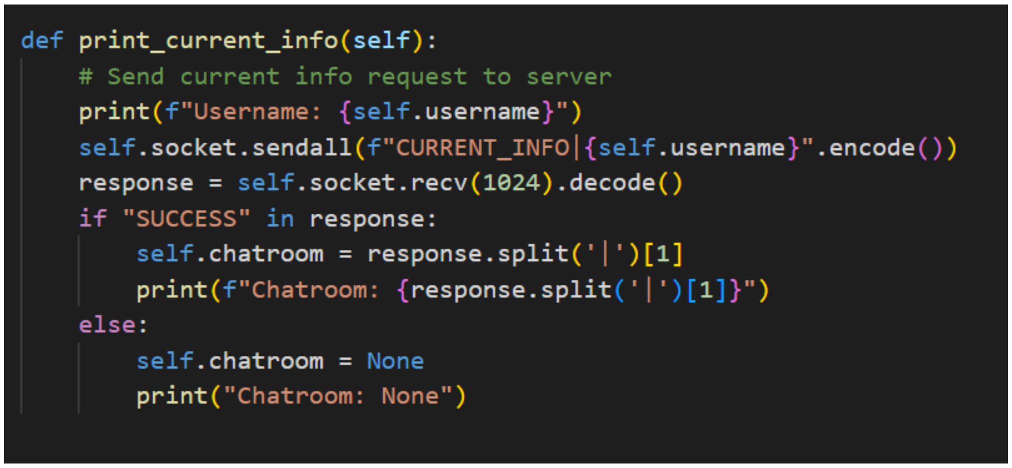
```python
def leave_chatroom(self):
    # Send chatroom leave request to server
    if self.logged_in and self.chatroom:
        self.socket.sendall(f"LEAVE_CHATROOM|{self.username}|{self.chatroom}".encode(
    # Check if user is logged in
    elif self.logged_in == False:
        print("You must be logged in to leave chatrooms.")
        return
    # Check if user is in a chatroom
    elif self.chatroom == None:
        print("You must be in a chatroom to leave it.")
        return

    # Check if chatroom was left successfully
    response = self.socket.recv(1024).decode()
    if response == "SUCCESS":
        print("Chatroom left!")
        self.chatroom = None
    else:
        print("Chatroom leave failed.")
```

➢ The **leave_chatroom** method sends a message to the server to remove the user from the chatroom. If the user is not in a chatroom, it returns an error message. Otherwise, it updates the chatroom instance variable and returns a success message.



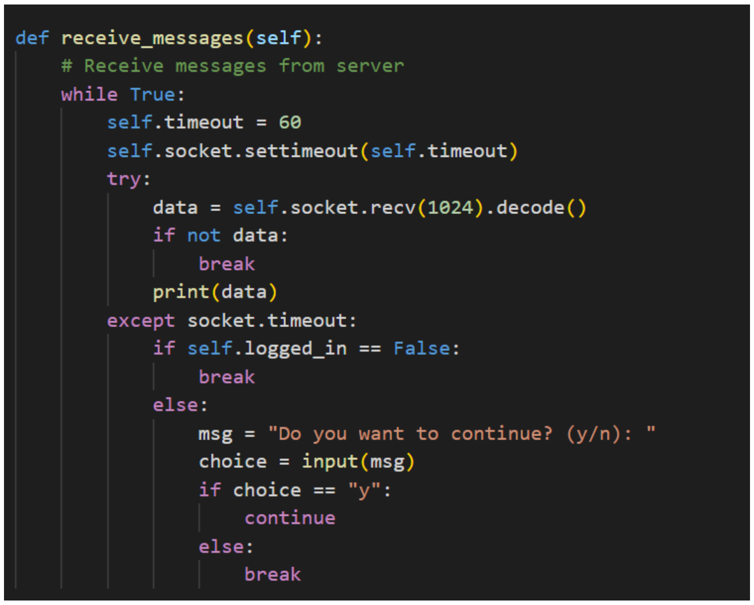
```python
def send_message(self, message):
    # Send message to server
    if self.logged_in and self.chatroom:
        self.socket.sendall(f"MESSAGE|{self.username}|{message}".encode())
    # Check if user is logged in
    elif self.logged_in == False:
        print("You must be logged in to send messages.")
        return
    # Check if user is in a chatroom
    elif self.chatroom == None:
        print("You must be in a chatroom to send messages.")
        return

    # Check if message was sent successfully
    response = self.socket.recv(1024).decode()
    if "ERROR: Message not sent" in response:
        print(f"Error: {response}")
    else:
        while "SUCCESS" not in response:
            response = self.socket.recv(1024).decode()
        print("Message sent!")
```

➢ The **send_message** method takes a message and sends it to the server to broadcast to all clients in the same chatroom as the user. If the user is not in a chatroom, it returns an error message.



```python
def print_current_info(self):
    # Send current info request to server
    print(f"Username: {self.username}")
    self.socket.sendall(f"CURRENT_INFO|{self.username}".encode())
    response = self.socket.recv(1024).decode()
    if "SUCCESS" in response:
        self.chatroom = response.split('|')[1]
        print(f"Chatroom: {response.split('|')[1]}")
    else:
        self.chatroom = None
        print("Chatroom: None")
```

➢ The **current_info** method returns the current chatroom the user is in, if any. If the user is not in a chatroom, it returns an error message.



```python
def receive_messages(self):
    # Receive messages from server
    while True:
        self.timeout = 60
        self.socket.settimeout(self.timeout)
        try:
            data = self.socket.recv(1024).decode()
            if not data:
                break
            print(data)
        except socket.timeout:
            if self.logged_in == False:
                break
            else:
                msg = "Do you want to continue? (y/n): "
                choice = input(msg)
                if choice == "y":
                    continue
                else:
                    break
```

➢ The **receive_messages** method listens for incoming messages from the server and prints them to the console. It runs on a separate thread so that the client can send and receive messages simultaneously. It also has a timeout which is set to 1 min. It will prompt the user whether they want to continue receiving the messages after the timeout.