CHAT ROOM IN PYTHON

In this article, you'll be learning how to build a chat room in python. It's one of the very simple projects one can build in python with just a few lines of code. Without much ado, let's dive into the tutorial.

A simple chat room in python is built using the concepts of *socket programming* and *multithreading*. We'll be building a chatroom that has one server and many clients can connect to that server and request for its services concurrently. Before starting with the code, we'll have a quick insight into these concepts for a better understanding of the project; if you are already familiar with these concepts feel free to skip to the coding section.

Client-Server Architecture:

A server is a program or a device that manages and delivers network resources and services for the client to consume. A client is a program or device that requests for services or resources from the server. There can be more than one client that requests services from a server. A client initiates the communication session with the servers which await upon incoming requests.

Socket Programming:

Sockets are interior endpoints of a communication channel across which the two connected devices(nodes) can exchange data; in our scenario, one of the nodes will be a server and the other will be a client. The IP address of the device and the port number it is communicating from together form a socket. This means that a single device can have multiple sockets based on the port numbers being used.
The server will receive incoming requests from clients wanting to communicate and the connected clients will be able to communicate with each other via the server.
To achieve socket programming in python, we need to import the *socket* module which comes with various built-in methods required for creating sockets and help them associate with one another.
In this tutorial, we'll be using the following socket methods:

Builtin Method	Usage
socket.socket()	Used to create a socket; required on both server and client end to create a socket. It takes in two parameters; address family and the type.
socket.accept()	Used by the server to accept a connection; it returns a pair of values ('socket object', 'address of the socket present on the other

	end')
socket.bind()	Used by the server to bind to the address that is specified as a parameter
socket.connect()	Used by the client to connect to the address specified as a parameter
socket.listen()	Used to enable the server to accept connections
socket.close()	Used to mark the socket as closed

Multithreading:

Multithreading is a way to achieve parallelism and concurrency using the concept of *threads.*
We will use the concept of multithreading to create multiple threads within one python program. In this way, many clients can communicate with the server 'concurrently' without waiting for the previous client to complete its communication.
Multithreading in python is achieved by importing the *threading* module.

Let's start coding:

As stated earlier, we need to set up a server program and a client program, therefore we'll be writing two scripts.

server.py

1. We'll first import the required modules.

```
import socket
import threading
```

2. We'll declare the required constants.

```
clients={}
addresses={}

BUFSIZ=1024 #Buffer size of the message being transmitted
PORT=33000  #Port number across which the communication is to happen
#The ip address of the device
HOST=socket.gethostbyname(socket.gethostname())
ADDR=(HOST,PORT) #A tuple of the ip address and the port number
#Encoding/Decoding format for the message to be transmitted as
```

```
    FORMAT='utf-8'
    DISCONNECT_MSG="!DISCONNECT" #Disconnection message for the client
    # Creating a socket at the server side, and binding it with the
    specified address
    SERVER=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    SERVER.bind(ADDR)
```

3. We will define a function to accept the incoming connection request of the clients.

```
def accept_connections():
    while True:
      #To accept the connection
      client,client_addr=SERVER.accept()

      #Log the connection details on the server side
      print(f"[NEW CONNECTION] {client_addr} connected.")

      #Welcoming the client.
      client.send(bytes("Hello!"+"Please type in your name and press
      enter.",FORMAT))

      #Adding the client address to the addresses dictionary for future
      use
      addresses[client]=client_addr

      #Creating a new thread for the specific client
      thread=threading.Thread(target=handle_client, args=(client,))
      #Starting the new thread
      thread.start()
```

4. We will define a function to handle each client separately/concurrently. This is the function that is executed, each time a new thread is started.

```
def handle_client(client):
  #Receive the name of the client and decode it using the specified
   format
   name=client.recv(BUFSIZ).decode(FORMAT)

   #Send a welcoming message to the client and instructions on how to
   quit the chatroom
```

```
    welcome=f"Welcome {name}, If you ever want to quit, type
      "+DISCONNECT_MSG+" for a clean disconnection."
    client.send(bytes(welcome,FORMAT))

    #Broadcast the message to the already connected clients that we have a
    new connection.
    msg=f"{name} has joined the chatroom!"
    broadcast(bytes(msg,FORMAT))

    #Add the new client to the clients dictionary
    clients[client]=name

    #Infinite loop, waiting for the client's messages to broadcast. The
    loop will end once the client sends the disconnection message
    while True:
        #Receive any incoming message from a client
        clmsg=client.recv(BUFSIZ)

        #As long as the client does not send the disconnection message,
        broadcast it to the other connected clients.
        if clmsg!=bytes(DISCONNECT_MSG,FORMAT):
            broadcast(clmsg,name+": ")

        #If the client sends the disconnection message, close that
         connection, delete the client form the clients dictionary, notify
         other active clients that the specific client has disconnected,
         and break the while loop
        else:
            client.close()
            del clients[client]
            broadcast(bytes(f"{name} has left the chatroom!",FORMAT))
            break
```

5. We will define a function to broadcast messages to active clients.

```
#Broadcast the message to all the clients in the clients dictionary
def broadcast(msg,prefix=""):
    for sock in clients:
```

```
        sock.send(bytes(prefix,FORMAT)+msg)
```

6. Code to start the server and listen for incoming connections

```
if __name__=="__main__":
    SERVER.listen(5)
    print("Waiting for connection... ")
    ACCEPT_THREAD=threading.Thread(target=accept_connections)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()
```

This completes the server.py script, now we'll write the script for the client-side.

client.py

1. We'll first import the required modules

```
import socket
import threading
import time
```

2. We'll declare the required constants

```
BUFSIZ = 1024 #Buffer size of the message being transmitted
PORT = 33000  #Port number across which the communication is to happen
(should be same as the server's)

# The ip address of the device
HOST = socket.gethostbyname(socket.gethostname())
#A tuple of the ip address and the port number
ADDR = (HOST, PORT)
#Encoding/Decoding format for the message to be transmitted as
FORMAT = 'utf-8'
#Disconnection message for the client
DISCONNECT_MSG = "!DISCONNECT"

# Creating a socket at the client-side, and connecting it to the server
with the specified address
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(ADDR)
```

3. We'll define a function that will receive the incoming messages from the server

```
def receive():
    #Infinite loop that receive messages from the server until the client
    leaves the chat
    while True:
        try:
            msg = client_socket.recv(BUFSIZ).decode(FORMAT)
            print(msg)
        except OSError:
            break
```

4. A function that will send messages to be broadcasted via the server

```
def send():
    #Input and send the username to the server
    my_username = input("Username: ")
    username = my_username.encode(FORMAT)
    client_socket.send(username)

    #An infinite loop that will send the client's messages to the server,
    it will end once the client enters the disconnection message
    while True:
        msg=input(f"")
        #Unless the message is not a disconnection message, it will be
        encoded and sent to the server
        if msg!=DISCONNECT_MSG:
            msg=msg.encode(FORMAT)
            client_socket.send(msg)
        #If the message is the disconnection message, It will notify the
        server, close the socket connection, and break out of the while
        loop
        else:
            msg = msg.encode(FORMAT)
            client_socket.send(msg)
            client_socket.close()
            break
```

5. A main function to start the *receive* and *send* threads for the client concurrently

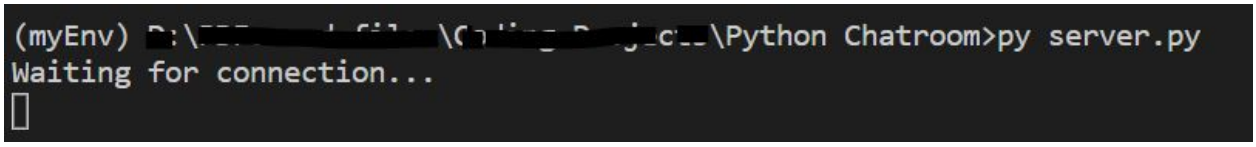
```
def main():
```

```
    receive_thread = threading.Thread(target=receive)
    receive_thread.start()

    send_thread = threading.Thread(target=send)
    send_thread.start()

    time.sleep(1)
    while threading.activeCount() > 1:
        pass


if __name__ == '__main__':
    main()
```

This completes our client-side script.

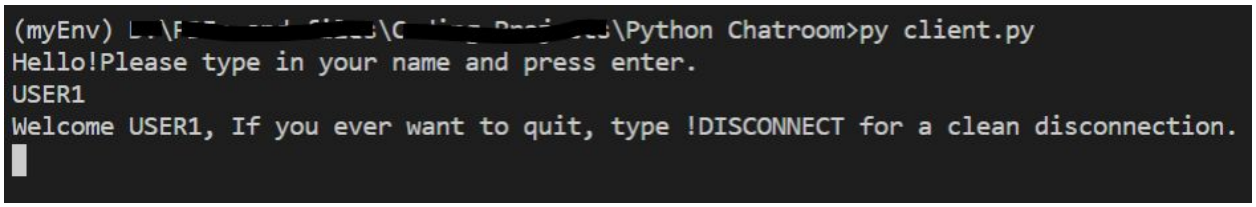
Chatroom in action:

Now let's run both the scripts and see our chatroom in action
Run the server script first

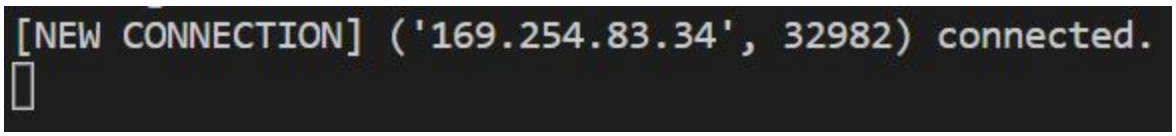


The server has successfully run and is waiting for client connections.

Now let's run the client script



The client has connected successfully and the server is notified



Now let's connect one more client.

```
[redacted]>py client.py
Hello!Please type in your name and press enter.
USER2
Welcome USER2, If you ever want to quit, type !DISCONNECT for a clean disconnection.
```

The server logs this connection as:

```
[NEW CONNECTION] ('169.254.83.34', 32982) connected.

[NEW CONNECTION] ('169.254.83.34', 33048) connected.
```

And USER1 is also notified

```
Welcome USER1, If you ever want to quit, type !DISCONNECT for a clean disconnection.
USER2 has joined the chatroom!
```

Similarly, there can be many clients.

```
USER3 has joined the c | USER3 has joined the chatroom | Welcome USER3, If you ever want to quit, typ
hatroom!               | !                             | e !DISCONNECT for a clean disconnection.
USER3: HI I'm user3    | USER3: HI I'm user3           | HI I'm user3
USER2: I'm user2       | I'm user2                     | USER3: HI I'm user3
I'm user1              | USER2: I'm user2              | USER2: I'm user2
USER1: I'm user1       | USER1: I'm user1              | USER1: I'm user1
test msg1              | USER1: test msg1              | USER1: test msg1
USER1: test msg1       | test msg 2                    | USER2: test msg 2
USER2: test msg 2      | USER2: test msg 2             | test msg 3
USER3: test msg 3      | USER3: test msg 3             | USER3: test msg 3
```

If one of the clients disconnect.

```
USER2 has left the chatroom! | !DISCONNECT | USER2 has left the chatroom!
```

Congratulations, you've successfully made a chatroom in python.