LAB 2: Socket Programming using Python

Team Members:

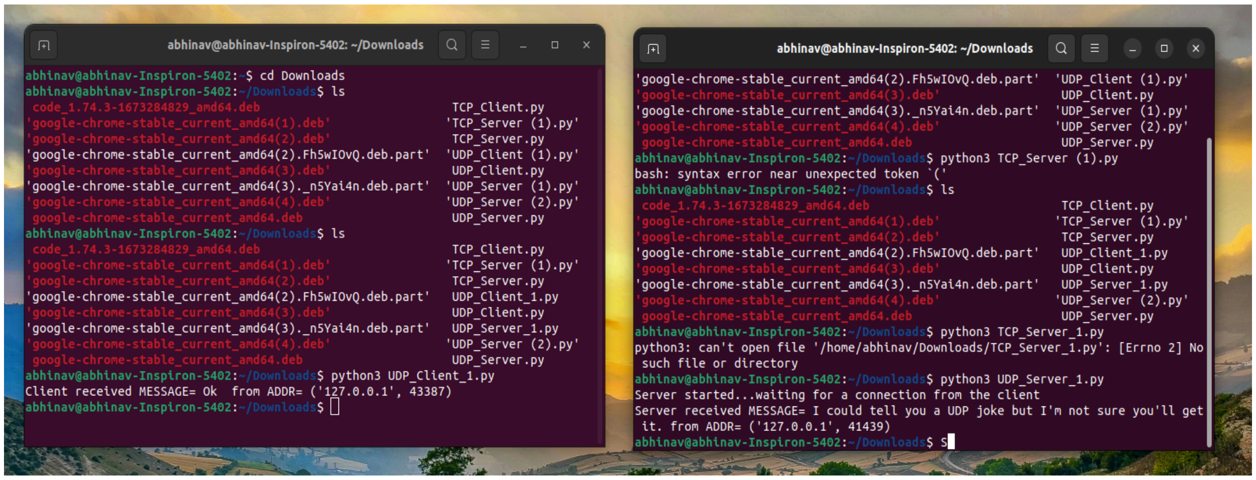
1. **Name :** ***Abhinav Reddy Gutha***
   Roll No. : 2103102

2. **Name :** ***Nidamanuri Sai Adarsh***
   Roll No. : 2103123

PART 1: UDP SOCKETS

1. Study the template provided for using UDP sockets in Python (files UDP_Server.py and UDP_Client.py). Make sure that you understand the purpose of UDP sockets, and the steps for creating and using these sockets. Run the Client and Server processes and observe the output.

Here's the output of the following UDP_Server and client python codes.



```
abhinav@abhinav-Inspiron-5402:~$ cd Downloads
abhinav@abhinav-Inspiron-5402:~/Downloads$ ls
 code_1.74.3-1673284829_amd64.deb                                  TCP_Client.py
'google-chrome-stable_current_amd64(1).deb'                       'TCP_Server (1).py'
'google-chrome-stable_current_amd64(2).deb'                        TCP_Server.py
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'         'UDP_Client (1).py'
'google-chrome-stable_current_amd64(3).deb'                        UDP_Client.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'         'UDP_Server (1).py'
'google-chrome-stable_current_amd64(4).deb'                       'UDP_Server (2).py'
 google-chrome-stable_current_amd64.deb                            UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ ls
 code_1.74.3-1673284829_amd64.deb                                  TCP_Client.py
'google-chrome-stable_current_amd64(1).deb'                       'TCP_Server (1).py'
'google-chrome-stable_current_amd64(2).deb'                        TCP_Server.py
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'          UDP_Client_1.py
'google-chrome-stable_current_amd64(3).deb'                        UDP_Client.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'          UDP_Server_1.py
'google-chrome-stable_current_amd64(4).deb'                       'UDP_Server (2).py'
 google-chrome-stable_current_amd64.deb                            UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Client_1.py
Client received MESSAGE= Ok  from ADDR= ('127.0.0.1', 43387)
abhinav@abhinav-Inspiron-5402:~/Downloads$ 
```

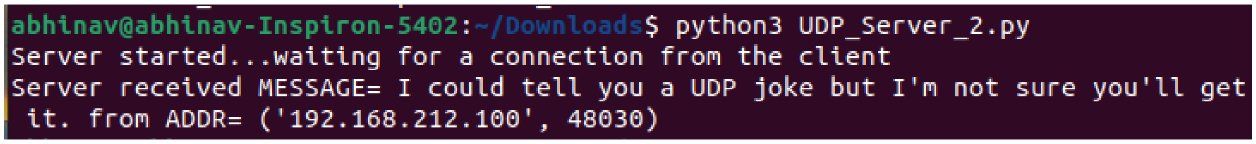
```
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'  'UDP_Client (1).py'
'google-chrome-stable_current_amd64(3).deb'                 UDP_Client.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'  'UDP_Server (1).py'
'google-chrome-stable_current_amd64(4).deb'                'UDP_Server (2).py'
 google-chrome-stable_current_amd64.deb                     UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Server (1).py
bash: syntax error near unexpected token `('
abhinav@abhinav-Inspiron-5402:~/Downloads$ ls
 code_1.74.3-1673284829_amd64.deb                           TCP_Client.py
'google-chrome-stable_current_amd64(1).deb'                'TCP_Server (1).py'
'google-chrome-stable_current_amd64(2).deb'                 TCP_Server.py
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'   UDP_Client_1.py
'google-chrome-stable_current_amd64(3).deb'                 UDP_Client.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'   UDP_Server_1.py
'google-chrome-stable_current_amd64(4).deb'                'UDP_Server (2).py'
 google-chrome-stable_current_amd64.deb                     UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Server_1.py
python3: can't open file '/home/abhinav/Downloads/TCP_Server_1.py': [Errno 2] No
 such file or directory
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Server_1.py
Server started...waiting for a connection from the client
Server received MESSAGE= I could tell you a UDP joke but I'm not sure you'll get
 it. from ADDR= ('127.0.0.1', 41439)
abhinav@abhinav-Inspiron-5402:~/Downloads$ S
```

In simple terms, what these two codes do is they communicate with the client, initially sending a message to the server saying “ I could tell you a UDP joke but I'm not sure you'll get it “. Then the server responds “OK” to the client. Here, both the server and client codes are running on the local host, not on different hosts.

2. Run the Client and the Server processes on different computers and check if they work as expected.

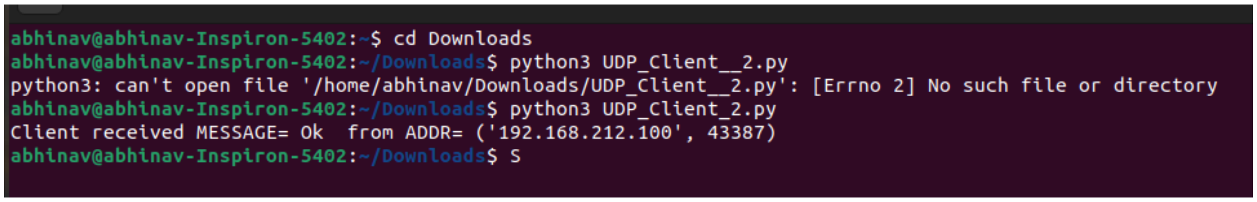
Yes, it’s working the same as expected. We need to change the IP address from local hosts to the respective computer’s IP address. Here’s the output of the following code:

Server:



```
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Server_2.py
Server started...waiting for a connection from the client
Server received MESSAGE= I could tell you a UDP joke but I'm not sure you'll get
 it. from ADDR= ('192.168.212.100', 48030)
```

Client:

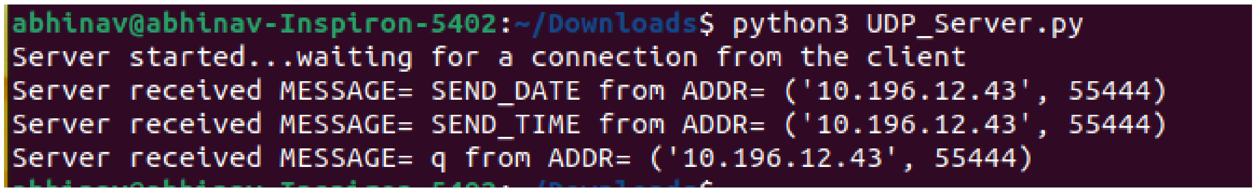


```
abhinav@abhinav-Inspiron-5402:~$ cd Downloads
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Client__2.py
python3: can't open file '/home/abhinav/Downloads/UDP_Client__2.py': [Errno 2] No such file or directory
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Client_2.py
Client received MESSAGE= Ok  from ADDR= ('192.168.212.100', 43387)
abhinav@abhinav-Inspiron-5402:~/Downloads$ S
```

3. Design and implement a Client-Server system that uses UDP sockets to do the following:
◦ The client sends the server a request. The request string can either be: “SEND_DATE” or “SEND_TIME”.
◦ The server runs in a infinite loop where it keeps waiting for requests. Whenever it sees a request, it responds by sending either the current DATE or the current TIME in (HH:MM:SS) format as specified in the request.
◦ When the Client receives a response, it prints it.
◦ The Client runs in a loop where it generates multiple such requests, and the time between successive requests varies randomly between 1-2 seconds.

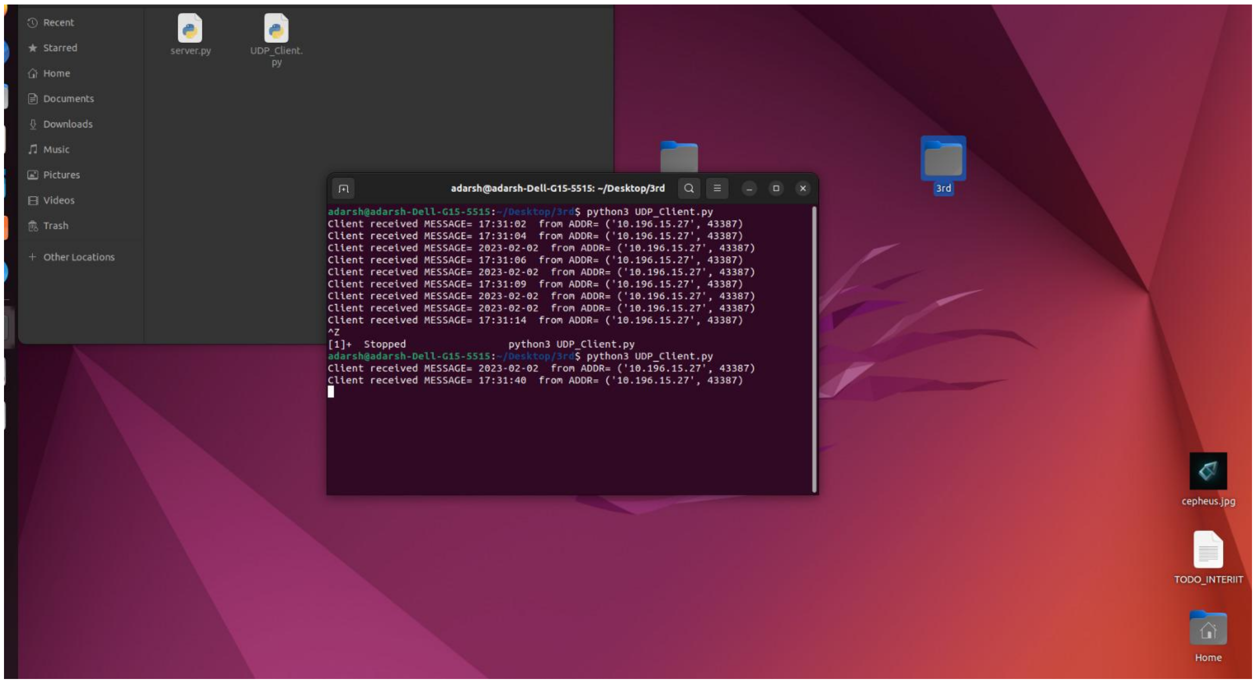
Here’s what our overall process looks like:

Server:



```
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 UDP_Server.py
Server started...waiting for a connection from the client
Server received MESSAGE= SEND_DATE from ADDR= ('10.196.12.43', 55444)
Server received MESSAGE= SEND_TIME from ADDR= ('10.196.12.43', 55444)
Server received MESSAGE= q from ADDR= ('10.196.12.43', 55444)
```

Client:

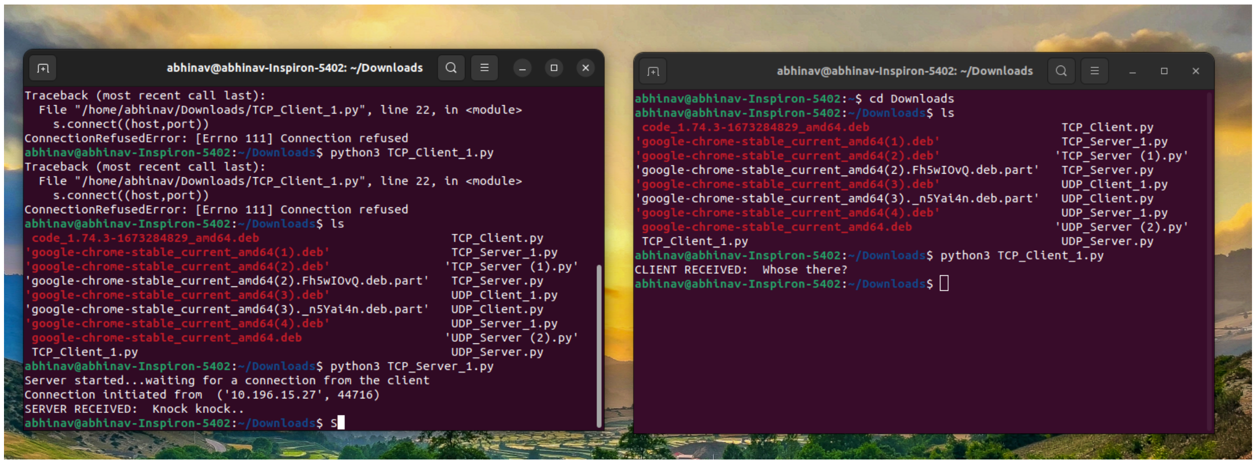


```
adarsh@adarsh-Dell-G15-5515: ~/Desktop/3rd
adarsh@adarsh-Dell-G15-5515:~/Desktop/3rd$ python3 UDP_Client.py
Client received MESSAGE= 17:31:02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 17:31:04  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 2023-02-02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 17:31:06  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 2023-02-02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 17:31:09  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 2023-02-02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 2023-02-02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 17:31:14  from ADDR= ('10.196.15.27', 43387)
^Z
[1]+  Stopped                 python3 UDP_Client.py
adarsh@adarsh-Dell-G15-5515:~/Desktop/3rd$ python3 UDP_Client.py
Client received MESSAGE= 2023-02-02  from ADDR= ('10.196.15.27', 43387)
Client received MESSAGE= 17:31:40  from ADDR= ('10.196.15.27', 43387)
```

PART 2: TCP SOCKETS

4. Study the template provided for using TCP sockets in Python (files TCP_Server.py and TCP_Client.py). Make sure that you understand the purpose of TCP sockets and the steps for creating and using these sockets. Observe the differences between UDP and TCP sockets and the steps for their use. Run the TCP Client and Server processes and observe the output.

We use SOCK_STREAM for TCP and SOCK_DGRAM for UDP in the socket function arguments. SOCK_DGRAM is a datagram oriented socket and SOCK_STREAM is a stream oriented socket regardless of the transport protocol used (Based on stackoverflow). TCP sockets ensure reliable transfer of data between the server and the client whereas UDP sockets will choose the best effort delivery option while transferring the data between the server and the client. Here’s what the output looks like:



abhinav@abhinav-Inspiron-5402: ~/Downloads

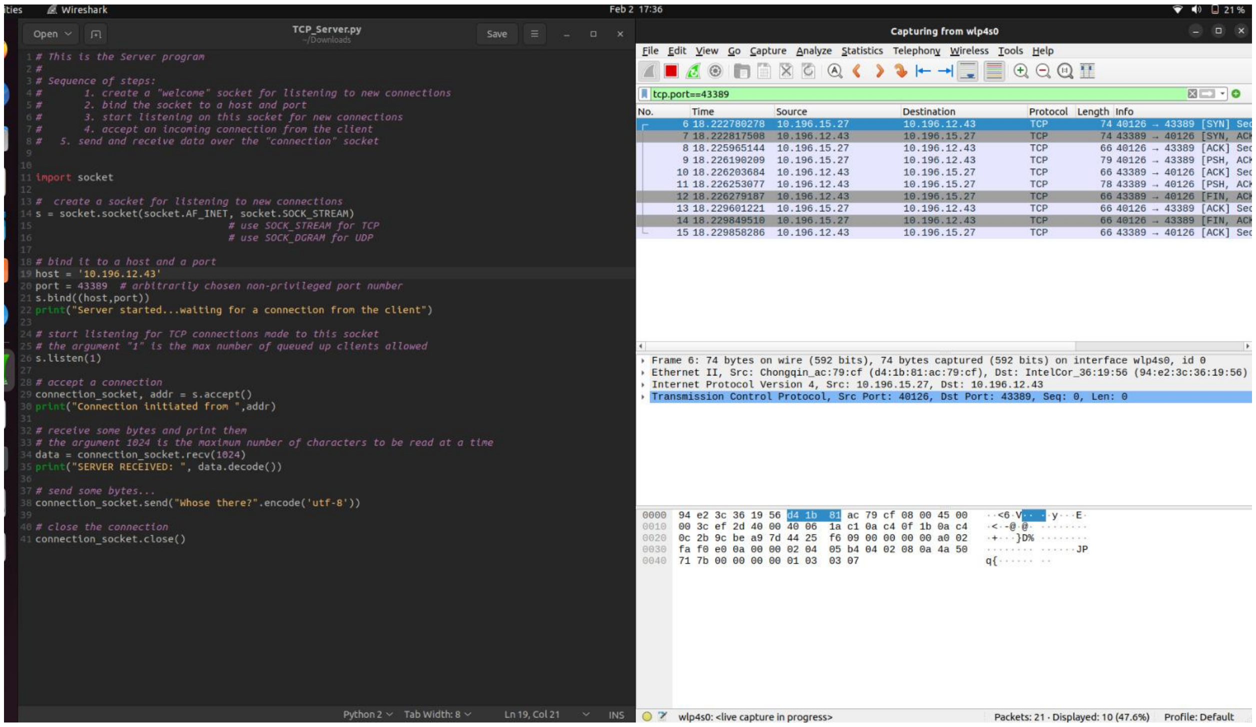
```
Traceback (most recent call last):
  File "/home/abhinav/Downloads/TCP_Client_1.py", line 22, in <module>
    s.connect((host,port))
ConnectionRefusedError: [Errno 111] Connection refused
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Client_1.py
Traceback (most recent call last):
  File "/home/abhinav/Downloads/TCP_Client_1.py", line 22, in <module>
    s.connect((host,port))
ConnectionRefusedError: [Errno 111] Connection refused
abhinav@abhinav-Inspiron-5402:~/Downloads$ ls
 code_1.74.3-1673284829_amd64.deb                                TCP_Client.py
'google-chrome-stable_current_amd64(1).deb'                      TCP_Server_1.py
'google-chrome-stable_current_amd64(2).deb'                     'TCP_Server (1).py'
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'        TCP_Server.py
'google-chrome-stable_current_amd64(3).deb'                      UDP_Client_1.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'        UDP_Client.py
'google-chrome-stable_current_amd64(4).deb'                      UDP_Server_1.py
 google-chrome-stable_current_amd64.deb                         'UDP_Server (2).py'
 TCP_Client_1.py                                                 UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Server_1.py
Server started...waiting for a connection from the client
Connection initiated from  ('10.196.15.27', 44716)
SERVER RECEIVED:  Knock knock..
abhinav@abhinav-Inspiron-5402:~/Downloads$ S
```

abhinav@abhinav-Inspiron-5402: ~/Downloads

```
abhinav@abhinav-Inspiron-5402:~$ cd Downloads
abhinav@abhinav-Inspiron-5402:~/Downloads$ ls
 code_1.74.3-1673284829_amd64.deb                                TCP_Client.py
'google-chrome-stable_current_amd64(1).deb'                      TCP_Server_1.py
'google-chrome-stable_current_amd64(2).deb'                     'TCP_Server (1).py'
'google-chrome-stable_current_amd64(2).Fh5wIOvQ.deb.part'        TCP_Server.py
'google-chrome-stable_current_amd64(3).deb'                      UDP_Client_1.py
'google-chrome-stable_current_amd64(3)._n5Yai4n.deb.part'        UDP_Client.py
'google-chrome-stable_current_amd64(4).deb'                      UDP_Server_1.py
 google-chrome-stable_current_amd64.deb                         'UDP_Server (2).py'
 TCP_Client_1.py                                                 UDP_Server.py
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Client_1.py
CLIENT RECEIVED:  Whose there?
abhinav@abhinav-Inspiron-5402:~/Downloads$ 
```

Initially there’s a connection established between the server and the client where the server sends a message to the client saying “Who’s there?”. Then the client responds “Knock Knock…”.

5. Start up Wireshark and apply a filter such that only the traffic generated by your Client and Server processes are displayed. Identify the messages used by TCP during the Handshake and the actual text sent by the two processes. Are the “contents” of the packet (the message strings) visible within Wireshark? (This is what we’d expect since the strings aren’t encrypted before sending.)



```python
# This is the Server program
#
# Sequence of steps:
#       1. create a "welcome" socket for listening to new connections
#       2. bind the socket to a host and port
#       3. start listening on this socket for new connections
#       4. accept an incoming connection from the client
#   5. send and receive data over the "connection" socket


import socket

#  create a socket for listening to new connections
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                                # use SOCK_STREAM for TCP
                                # use SOCK_DGRAM for UDP

# bind it to a host and a port
host = '10.196.12.43'
port = 43389  # arbitrarily chosen non-privileged port number
s.bind((host,port))
print("Server started...waiting for a connection from the client")

# start listening for TCP connections made to this socket
# the argument "1" is the max number of queued up clients allowed
s.listen(1)

# accept a connection
connection_socket, addr = s.accept()
print("Connection initiated from ",addr)

# receive some bytes and print them
# the argument 1024 is the maximum number of characters to be read at a time
data = connection_socket.recv(1024)
print("SERVER RECEIVED: ", data.decode())

# send some bytes...
connection_socket.send("Whose there?".encode('utf-8'))

# close the connection
connection_socket.close()
```

```
tcp.port==43389

No.  Time          Source        Destination   Protocol Length Info
  6 18.222780278 10.196.15.27  10.196.12.43  TCP      74 40126 → 43389 [SYN] Se
  7 18.222817508 10.196.12.43  10.196.15.27  TCP      74 43389 → 40126 [SYN, AC
  8 18.225965144 10.196.15.27  10.196.12.43  TCP      66 40126 → 43389 [ACK] Se
  9 18.226190209 10.196.15.27  10.196.12.43  TCP      79 40126 → 43389 [PSH, AC
 10 18.226203684 10.196.12.43  10.196.15.27  TCP      66 43389 → 40126 [ACK] Se
 11 18.22625307  10.196.12.43  10.196.15.27  TCP      78 43389 → 40126 [PSH, AC
 12 18.226279187 10.196.12.43  10.196.15.27  TCP      66 43389 → 40126 [FIN, AC
 13 18.229601221 10.196.15.27  10.196.12.43  TCP      66 40126 → 43389 [ACK] Se
 14 18.229849510 10.196.15.27  10.196.12.43  TCP      66 40126 → 43389 [FIN, AC
 15 18.229858286 10.196.12.43  10.196.15.27  TCP      66 43389 → 40126 [ACK] Se

Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlp4s0, id 0
Ethernet II, Src: Chongqin_ac:79:cf (d4:1b:81:ac:79:cf), Dst: IntelCor_36:19:56 (94:e2:3c:36:19:56)
Internet Protocol Version 4, Src: 10.196.15.27, Dst: 10.196.12.43
Transmission Control Protocol, Src Port: 40126, Dst Port: 43389, Seq: 0, Len: 0

0000  94 e2 3c 36 19 56 d4 1b  81 ac 79 cf 08 00 45 00   ··<6·V·· ··y···E·
0010  00 3c ef 2d 40 00 40 06  1a c1 0a c4 0f 1b 0a c4   ·<·-@·@· ········
0020  0c 2b 9c be a9 7d 44 25  f6 09 00 00 00 00 a0 02   ·+···}D% ········
0030  fa f0 e0 0a 00 00 02 04  05 b4 04 02 08 0a 4a 50   ········ ······JP
0040  71 7b 00 00 00 00 01 03  03 07                     q{······ ··

wlp4s0: <live capture in progress>    Packets: 21 · Displayed: 10 (47.6%)    Profile: Default
```

6. Design and implement a Client-Server system that uses TCP sockets to do the following:

◦ The client initiates communication with a Server by sending the server its name. (Choose some name for your client process). The Server remembers this name for the entire duration of the communication session.

◦ The client then runs in an infinite loop where its accepts a line of input from the user. The user is expected to enter a string consisting of two numbers and a simple arithmetic operation (separated by spaces), for example: “12 + 42” or “3.24 - 45” or “4.5 / -6” . If the input is not correctly formatted, a warning is displayed to the user. If correctly formatted, the Client sends this string to the Server.

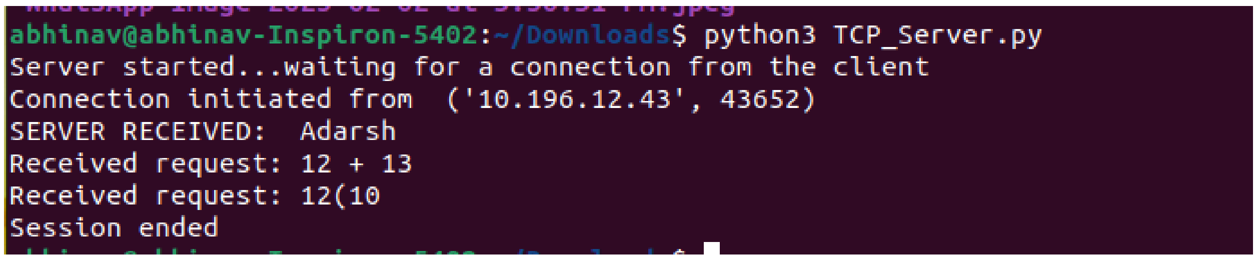
◦ The Server runs an infinite loop where it keeps waiting for requests from this client. Upon receiving a request, the server prints the received message, computes the answer by performing the arithmetic operation and sends it back as a string. The Client prints the answer it received from the server.

◦ **When the user wishes to stop, they enter “q”. The client process forwards the “q” to the Server upon which the server ends the communication session and prints “Session ended”. The Client processes stops.**

Here the server will have a function name perform_operation which evaluates the operation b/w two numbers provided by the client. Firstly the client establishes a connection with the server. When the client enters the input which has the thing of operation b/w two numbers, it evaluates and gives the output. When the input given is invalid, it returns as a invalid. When we quit, the client leaves the server.

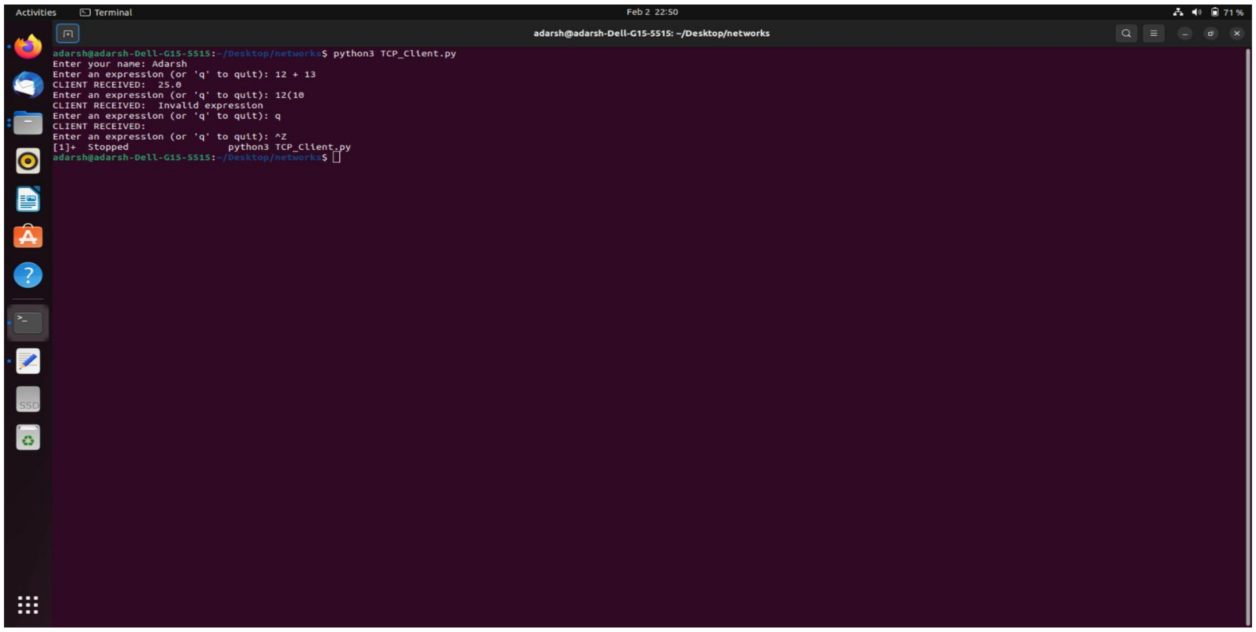
Here’s what the output looks like:

Server:



```
abhinav@abhinav-Inspiron-5402:~/Downloads$ python3 TCP_Server.py
Server started...waiting for a connection from the client
Connection initiated from  ('10.196.12.43', 43652)
SERVER RECEIVED:  Adarsh
Received request: 12 + 13
Received request: 12(10
Session ended
```

Client:



```
adarsh@adarsh-Dell-G15-5515:~/Desktop/networks$ python3 TCP_Client.py
Enter your name: Adarsh
Enter an expression (or 'q' to quit): 12 + 13
CLIENT RECEIVED:  25.0
Enter an expression (or 'q' to quit): 12(10
CLIENT RECEIVED:  Invalid expression
Enter an expression (or 'q' to quit): q
CLIENT RECEIVED:
Enter an expression (or 'q' to quit): ^Z
[1]+  Stopped                 python3 TCP_Client.py
adarsh@adarsh-Dell-G15-5515:~/Desktop/networks$ 
```

7. When you decide upon a “format” for the messages that can be correctly understood by the Client/Server processes, you have implicitly designed a “Protocol”. A protocol can be “Stateful” of “Stateless”.

◦ Is the application-layer protocol designed by you for question 3 Stateless?
◦ Is the application-layer protocol designed by you for question 6 Stateless?
◦ Is TCP a Stateless protocol?
◦ Is UDP a Stateless protocol?

A stateful protocol is a communication protocol in which the receiver may retain session state from previous requests. A stateless protocol is a communication protocol in which the receiver must not retain session state from previous requests. Based on this, we can say that

1. For question 3, it is a stateless protocol because it is datagram oriented communication, it may not retain session state. Every instance is independent of others.
2. For question 6, it is not a stateless protocol as we can observe in question 6 that it retains its session state from previous requests due to the stream line oriented communication.
3. TCP is not a stateless protocol.
4. UDP is a stateless protocol.

PART 3: Multiple Clients

8. For systems such as those described in question 6, think of how you could modify the Server so that it can handle multiple clients at the same time. For each connection (using TCP sockets), the Server should provide the same service as described. There are several ways to implement this. One way is to use multiple threads for the Server, one per connection.

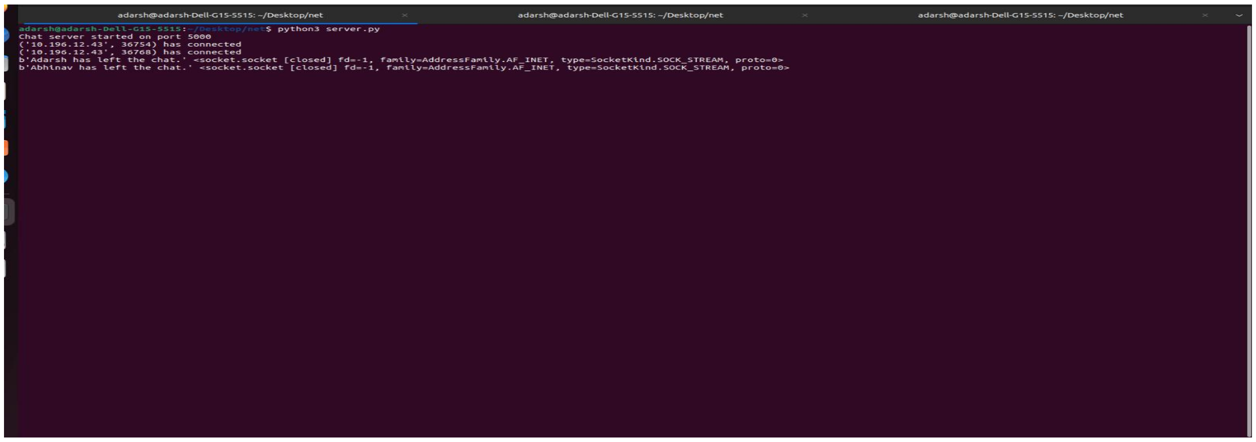
In this question, we wish to design such a multi-Client system to implement a “Chat Room” as follows:

◦ The Client process acts like a chat window. It takes user input, sends appropriate requests to the Server, and displays the messages sent by the server to the human user.
◦ The Server process acts like a chat room manager. It allows client processes to login to the chat room (each client needs to have a unique name). The server keeps track of all the clients that are currently logged in. Whenever any interesting event happens, (such as a new user logging in or leaving the chat room) the status is broadcast to all connected clients. Also, whatever each user types is broadcast to all
clients.
◦ At the beginning, the user is requested for a “login name”. The client process then sends a login request to the chat room (Server) with this name.
◦ After logging in, whatever lines the user types is broadcast to all clients along with the sender’s name. The following lines show an example of output that might be displayed to two different clients. Client 1 is the first to join the chat room.

Here we collect names of the clients initially and we add two threadings (One for input and one for output). Once the host enters his name, now he can chat whatever he wants to convey to the other clients. Once a client enters “quit”, then the server removes this client from the chat room.

Here’s what the output looks like:

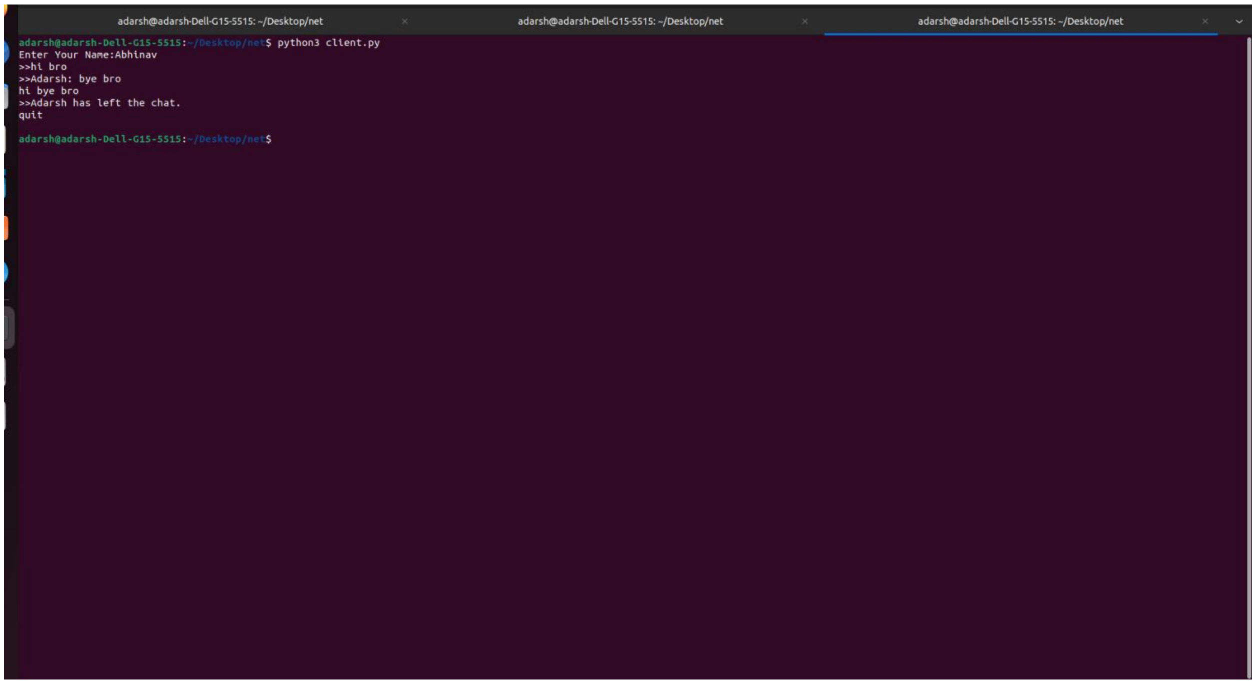
Server:



```
adarsh@adarsh-Dell-G15-5515:~/Desktop/net$ python3 server.py
Chat server started on port 5000
('10.196.12.43', 36754) has connected
('10.196.12.43', 36768) has connected
b'Adarsh has left the chat.' <socket.socket [closed] fd=-1, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0>
b'Abhinav has left the chat.' <socket.socket [closed] fd=-1, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0>
```

adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net

Client-1:

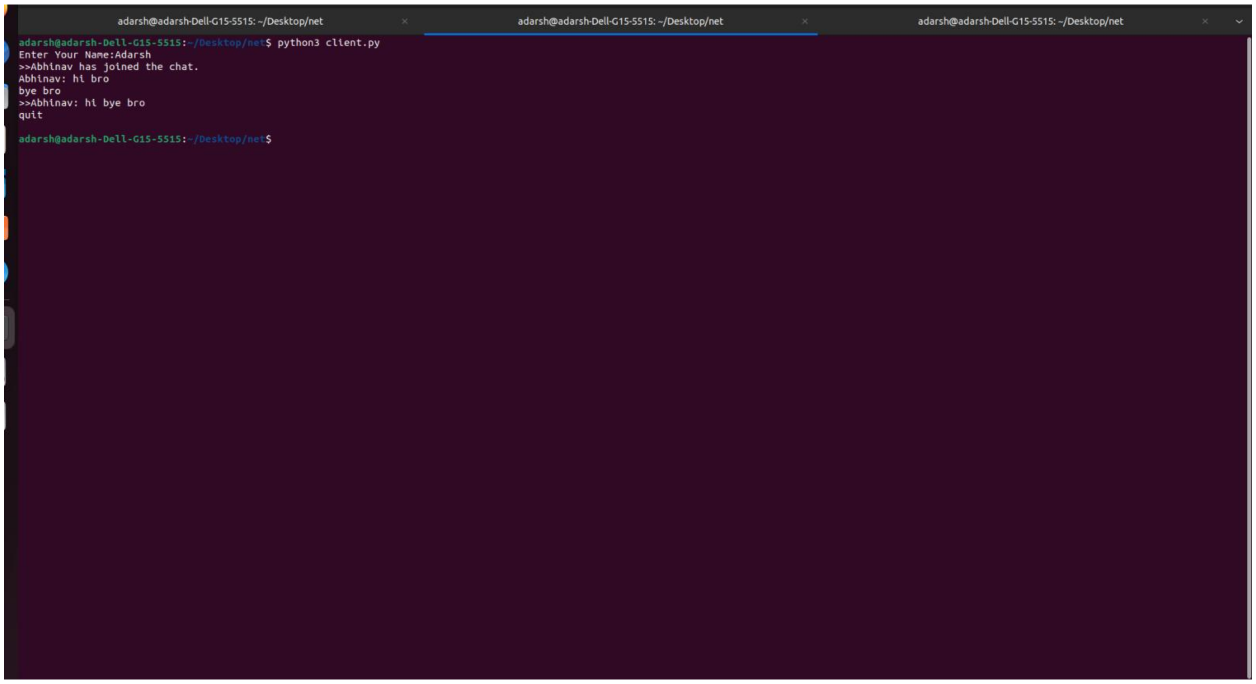


adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net

```
adarsh@adarsh-Dell-G15-5515:~/Desktop/net$ python3 client.py
Enter Your Name:Abhinav
>>hi bro
>>Adarsh: bye bro
hi bye bro
>>Adarsh has left the chat.
quit

adarsh@adarsh-Dell-G15-5515:~/Desktop/net$
```

Client-2:


adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net
adarsh@adarsh-Dell-G15-5515: ~/Desktop/net

```
adarsh@adarsh-Dell-G15-5515:~/Desktop/net$ python3 client.py
Enter Your Name:Adarsh
>>Abhinav has joined the chat.
Abhinav: hi bro
bye bro
>>Abhinav: hi bye bro
quit

adarsh@adarsh-Dell-G15-5515:~/Desktop/net$
```