

CN Lab Report

Week 5

PES2UG19CS412
Suhan B Revankar

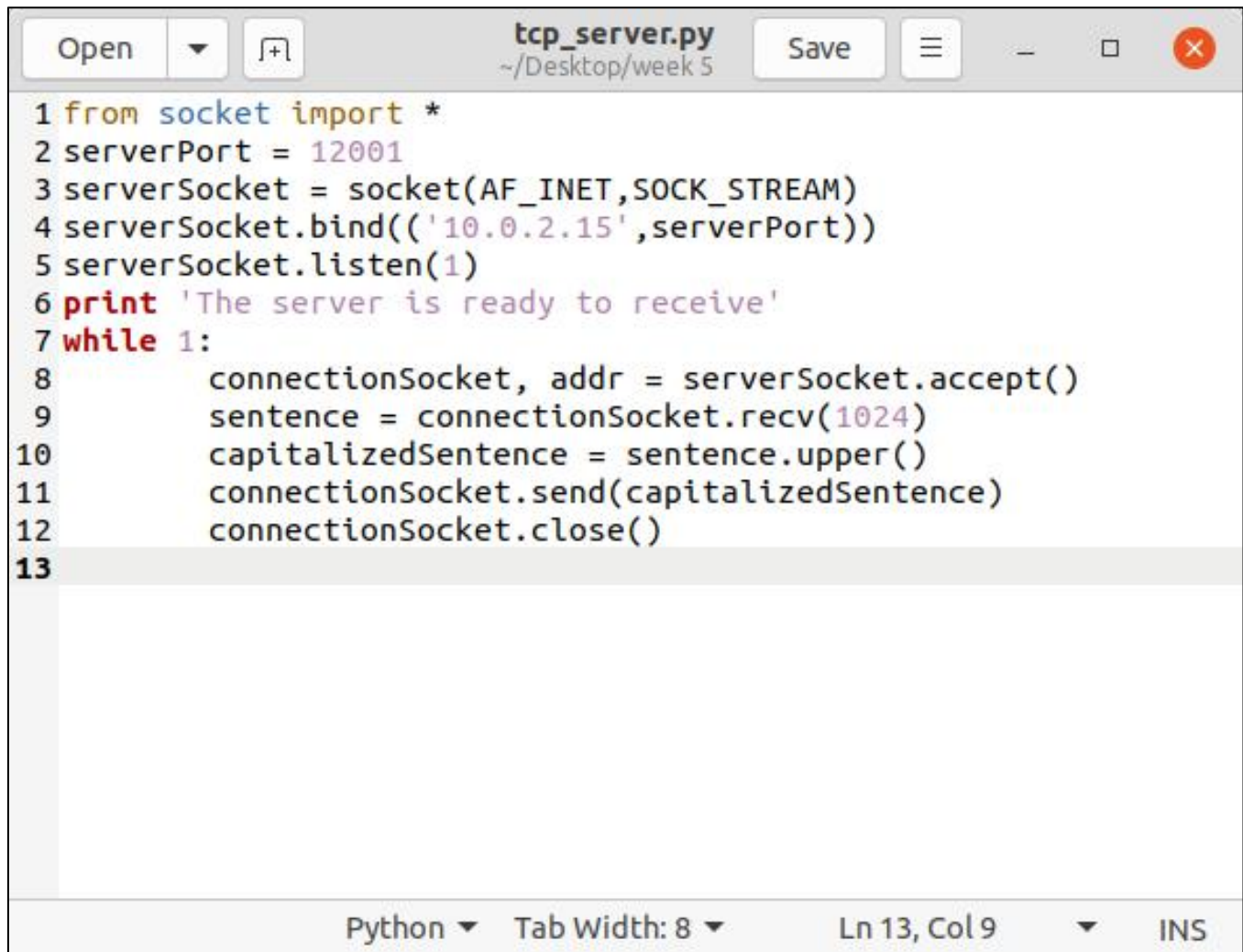
1. Socket Programming

1. Create an application that will
 - a. Convert lowercase letters to uppercase
 - e.g. [a...z] to [A...Z]
 - code will not change any special characters, e.g. &*!
 - b. If the character is in uppercase, the program must not alter
2. Create Socket API both for client and server.
3. Must take the server address and port from the Command Line Interface (CLI).

1.1 TCP Connection

- A TCP connection can be made between two machines with the help of a socket interface using the socket library on Python3.
- To create a TCP socket interface, the type of socket needs to be set as `SOCK_STREAM`.
- The type of addresses needs to be set as `AF_INET` which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using the `bind()` function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

1.1.1 TCP Server

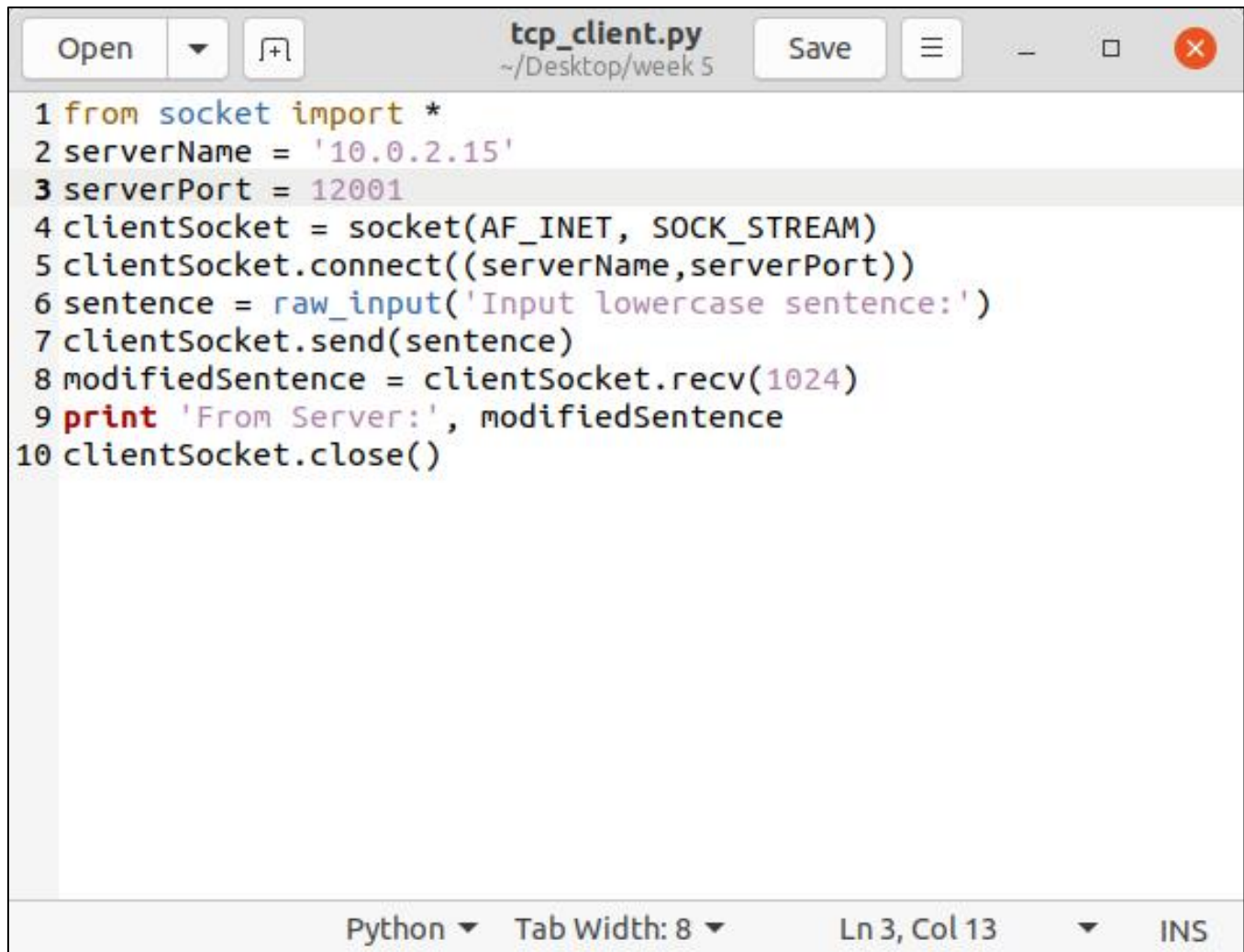


The image shows a code editor window titled "tcp_server.py" with a subtitle "~/Desktop/week 5". The editor has a menu bar with "Open", "Save", and a hamburger menu icon. The code is written in Python and implements a simple TCP server. It imports the socket module, sets a server port to 12001, binds the socket to the IP address 10.0.2.15, and listens for connections. A print statement indicates the server is ready to receive. A while loop then accepts connections, receives data, capitalizes it, sends it back, and closes the connection. The status bar at the bottom shows "Python", "Tab Width: 8", "Ln 13, Col 9", and "INS".

```
1 from socket import *
2 serverPort = 12001
3 serverSocket = socket(AF_INET, SOCK_STREAM)
4 serverSocket.bind(('10.0.2.15', serverPort))
5 serverSocket.listen(1)
6 print 'The server is ready to receive'
7 while 1:
8     connectionSocket, addr = serverSocket.accept()
9     sentence = connectionSocket.recv(1024)
10    capitalizedSentence = sentence.upper()
11    connectionSocket.send(capitalizedSentence)
12    connectionSocket.close()
13
```

Python ▾ Tab Width: 8 ▾ Ln 13, Col 9 ▾ INS

1.1.2 TCP Client

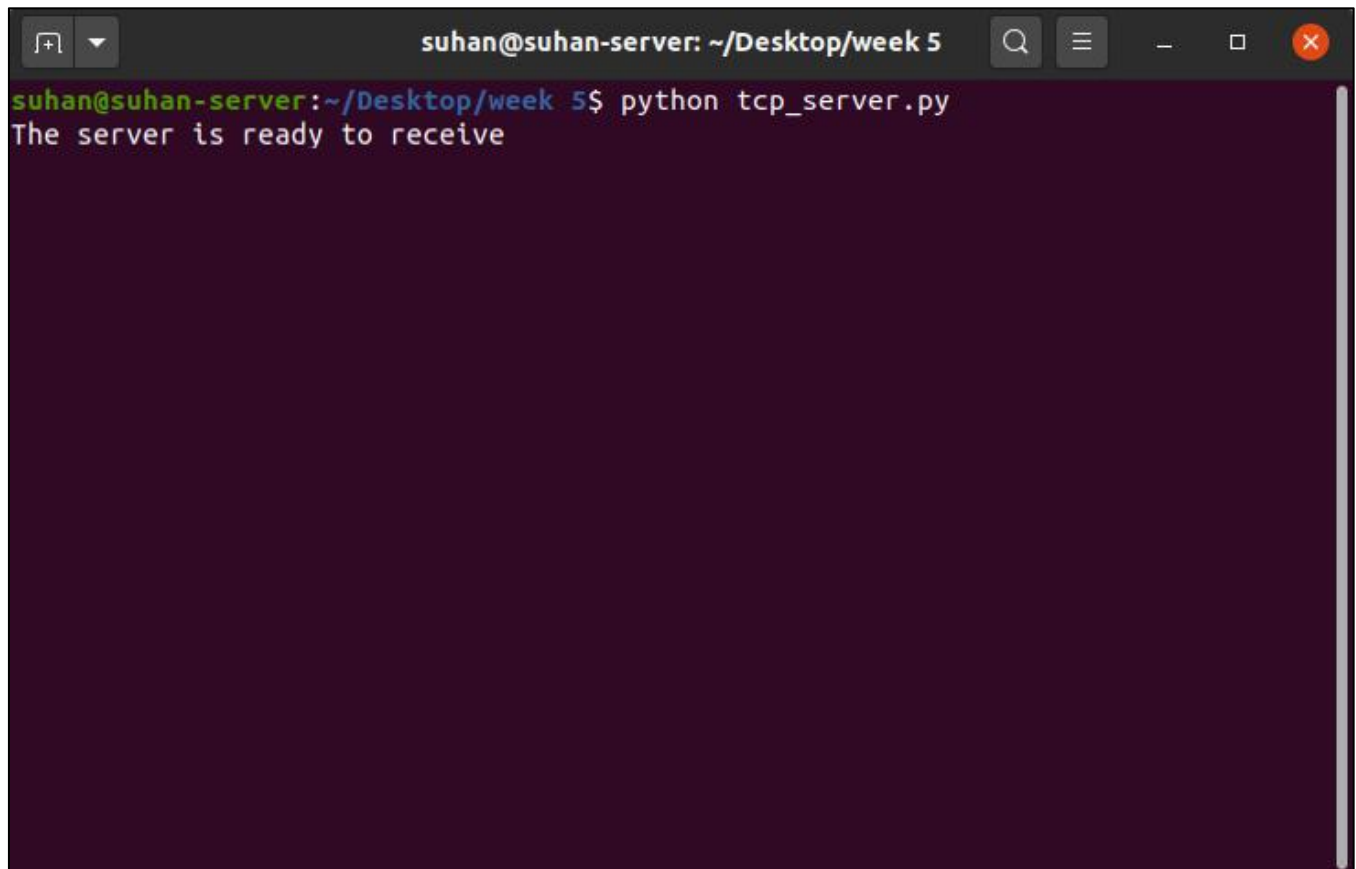


The image shows a code editor window titled "tcp_client.py" with a file path of "~/Desktop/week 5". The editor contains a Python script for a TCP client. The script imports the socket module, sets the server name to '10.0.2.15' and the server port to 12001. It then creates a client socket, connects to the server, sends a sentence entered by the user, receives a modified sentence from the server, and prints it. Finally, it closes the client socket. The status bar at the bottom indicates the language is Python, tab width is 8, and the cursor is at line 3, column 13.

```
1 from socket import *
2 serverName = '10.0.2.15'
3 serverPort = 12001
4 clientSocket = socket(AF_INET, SOCK_STREAM)
5 clientSocket.connect((serverName,serverPort))
6 sentence = raw_input('Input lowercase sentence:')
7 clientSocket.send(sentence)
8 modifiedSentence = clientSocket.recv(1024)
9 print 'From Server:', modifiedSentence
10 clientSocket.close()
```

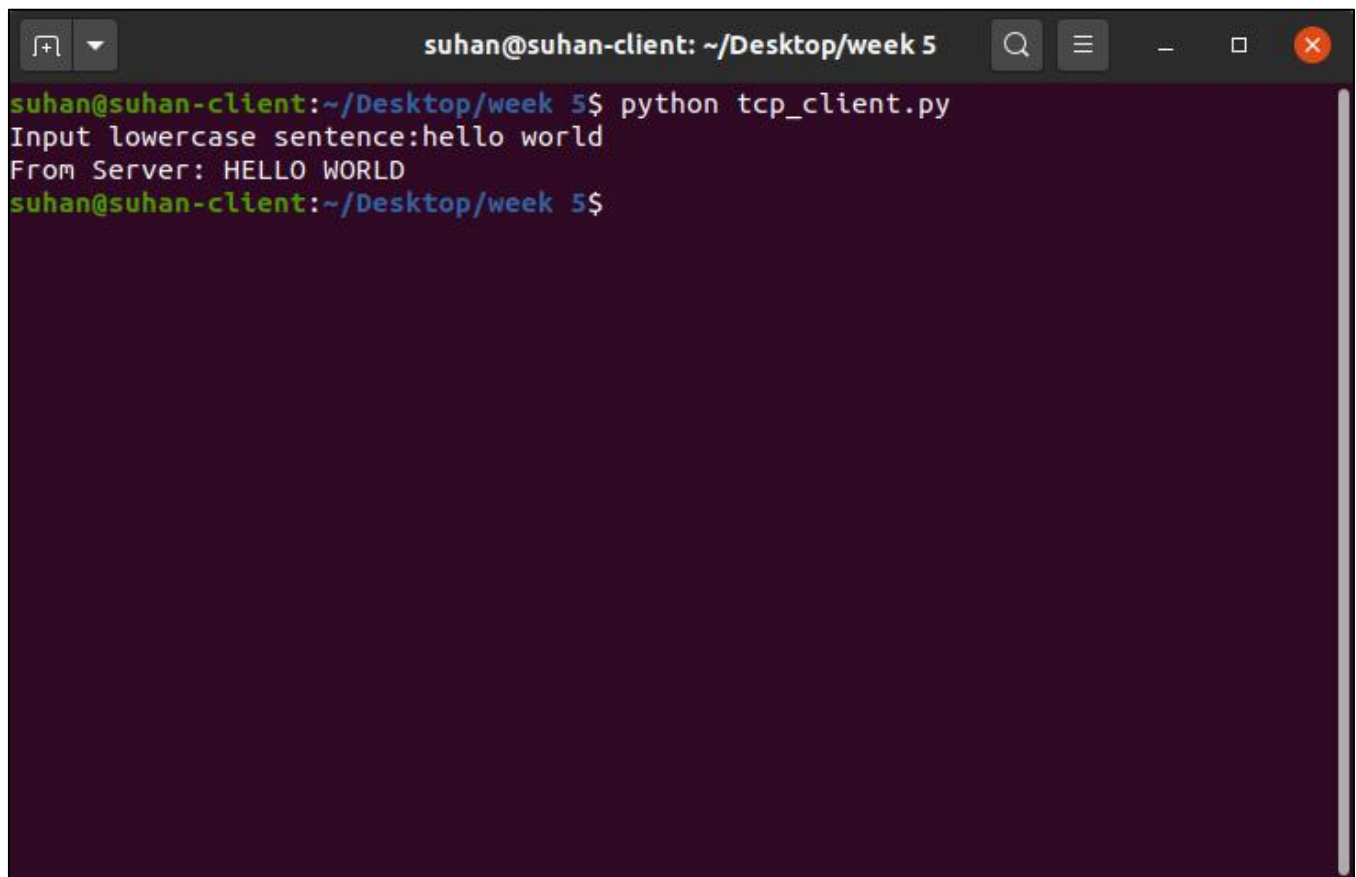
Python ▾ Tab Width: 8 ▾ Ln 3, Col 13 ▾ INS

1.1.3 TCP Connection between Server and Client

A terminal window with a dark background and light-colored text. The window title bar shows 'suhan@suhan-server: ~/Desktop/week 5' and standard window control buttons. The terminal content shows a command prompt where 'python tcp_server.py' has been executed, resulting in the output 'The server is ready to receive'.

```
suhan@suhan-server: ~/Desktop/week 5  
suhan@suhan-server:~/Desktop/week 5$ python tcp_server.py  
The server is ready to receive
```

TCP Server

A terminal window with a dark purple background. The title bar at the top shows the user 'suhan' on a machine named 'suhan-client' in the directory '~/Desktop/week 5'. The terminal contains the following text: a prompt 'suhan@suhan-client:~/Desktop/week 5\$' followed by the command 'python tcp_client.py', the user input 'hello world', and the server response 'HELLO WORLD'.

```
suhan@suhan-client: ~/Desktop/week 5
suhan@suhan-client:~/Desktop/week 5$ python tcp_client.py
Input lowercase sentence:hello world
From Server: HELLO WORLD
suhan@suhan-client:~/Desktop/week 5$
```

TCP Client

1.1.4 Wireshark Capture for TCP Connection

The screenshot shows a Wireshark capture of a TCP connection. The packet list on the left shows a sequence of packets: a SYN packet (No. 1), a SYN-ACK packet (No. 2), an ACK packet (No. 3), a PSH-ACK packet (No. 4), and two more ACK packets (Nos. 6 and 8). The packet details pane for packet 4 is selected, showing the TCP header and data. The header includes the sequence number 1, acknowledgment number 12, and window size 502. The data field shows 11 bytes of data.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	TCP	76	48472 -> 12001 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1419844658 TSecr=0 WS=128
2	0.000078674	10.0.2.15	10.0.2.4	TCP	76	12001 -> 48472 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1855977978 TSecr=1419844658
3	0.000504402	10.0.2.4	10.0.2.15	TCP	68	48472 -> 12001 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1419844659 TSecr=1855977978
4	4.569398095	10.0.2.4	10.0.2.15	TCP	79	48472 -> 12001 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=11 TSval=1419849228 TSecr=1855977978
5	4.569442988	10.0.2.15	10.0.2.4	TCP	68	12001 -> 48472 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSval=1855982547 TSecr=1419849228
6	4.569646072	10.0.2.15	10.0.2.4	TCP	79	12001 -> 48472 [PSH, ACK] Seq=1 Ack=12 Win=65152 Len=11 TSval=1855982548 TSecr=1419849228
7	4.569767709	10.0.2.15	10.0.2.4	TCP	68	12001 -> 48472 [FIN, ACK] Seq=12 Ack=12 Win=65152 Len=0 TSval=1855982548 TSecr=1419849228
8	4.569821288	10.0.2.4	10.0.2.15	TCP	68	48472 -> 12001 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=1419849229 TSecr=1855982548

Stream index: 0
[TCP Segment Len: 11]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 1471992233
[Next Sequence Number: 12 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 2650745303
1000 = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
Checksum: 0x8215 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[SEQ/ACK analysis]
[Timestamps]
Data (11 bytes)
Data: 68656c66f20776f726c64
[Length: 11]

0020 0a 00 02 0f bd 58 2e e1 57 bc d1 a9 9d ff 29 d7 ...X

Transmission Control Protocol (tcp), 32 bytes

Packets: 10 · Displayed: 10 (100.0%) · Dropped: 0 (0.0%) Profile: Default

The screenshot shows a Wireshark capture of a TCP connection. The packet list on the left shows a sequence of packets: a SYN packet (No. 1), a SYN-ACK packet (No. 2), an ACK packet (No. 3), a PSH-ACK packet (No. 4), and two more ACK packets (Nos. 6 and 8). The packet details pane for packet 4 is selected, showing the TCP header and data. The header includes the sequence number 1, acknowledgment number 12, and window size 509. The data field shows 11 bytes of data.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	TCP	76	48472 -> 12001 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1419844658 TSecr=0 WS=128
2	0.000078674	10.0.2.15	10.0.2.4	TCP	76	12001 -> 48472 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1855977978 TSecr=1419844658
3	0.000504402	10.0.2.4	10.0.2.15	TCP	68	48472 -> 12001 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1419844659 TSecr=1855977978
4	4.569398095	10.0.2.4	10.0.2.15	TCP	79	48472 -> 12001 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=11 TSval=1419849228 TSecr=1855977978
5	4.569442988	10.0.2.15	10.0.2.4	TCP	68	12001 -> 48472 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSval=1855982547 TSecr=1419849228
6	4.569646072	10.0.2.15	10.0.2.4	TCP	79	12001 -> 48472 [PSH, ACK] Seq=1 Ack=12 Win=65152 Len=11 TSval=1855982548 TSecr=1419849228
7	4.569767709	10.0.2.15	10.0.2.4	TCP	68	12001 -> 48472 [FIN, ACK] Seq=12 Ack=12 Win=65152 Len=0 TSval=1855982548 TSecr=1419849228
8	4.569821288	10.0.2.4	10.0.2.15	TCP	68	48472 -> 12001 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TSval=1419849229 TSecr=1855982548

Transmission Control Protocol, Src Port: 12001, Dst Port: 48472, Seq: 1, Ack: 12, Len: 11
Source Port: 12001
Destination Port: 48472
[Stream index: 0]
[TCP Segment Len: 11]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 2650745303
[Next Sequence Number: 12 (relative sequence number)]
Acknowledgment Number: 12 (relative ack number)
Acknowledgment number (raw): 1471992244
1000 = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 509
[Calculated window size: 65152]
[Window size scaling factor: 128]
Checksum: 0x1844 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[SEQ/ACK analysis]
[Timestamps]
Data (11 bytes)
TCP payload
Data: 68656c66f20776f726c64
[Length: 11]

0020 0a 00 02 04 2e e1 bd 58 9d ff 29 d7 57 bc d1 b4 ...X

Transmission Control Protocol (tcp), 32 bytes

Packets: 10 · Displayed: 10 (100.0%) · Dropped: 0 (0.0%) Profile: Default

1.2 UDP Connection

- A UDP connection can be made between two machines with the help of a socket interface using the socketlibrary on Python3.
- To create a UDP socket interface, the type of socket needs to be set as SOCK_DGRAM.
- The type of addresses needs to be set as AF_INET which corresponds to IPv4.
- Once the server socket application is created, it needs to be hosted and hence needs to bind to a host IP and port number using the bind()function.
- Similarly, the client socket application needs to connect to a host using the IP address and port number.
- The socket can now listen for incoming connections as well as send messages to connected host machines.

1.2.1 UDP Server

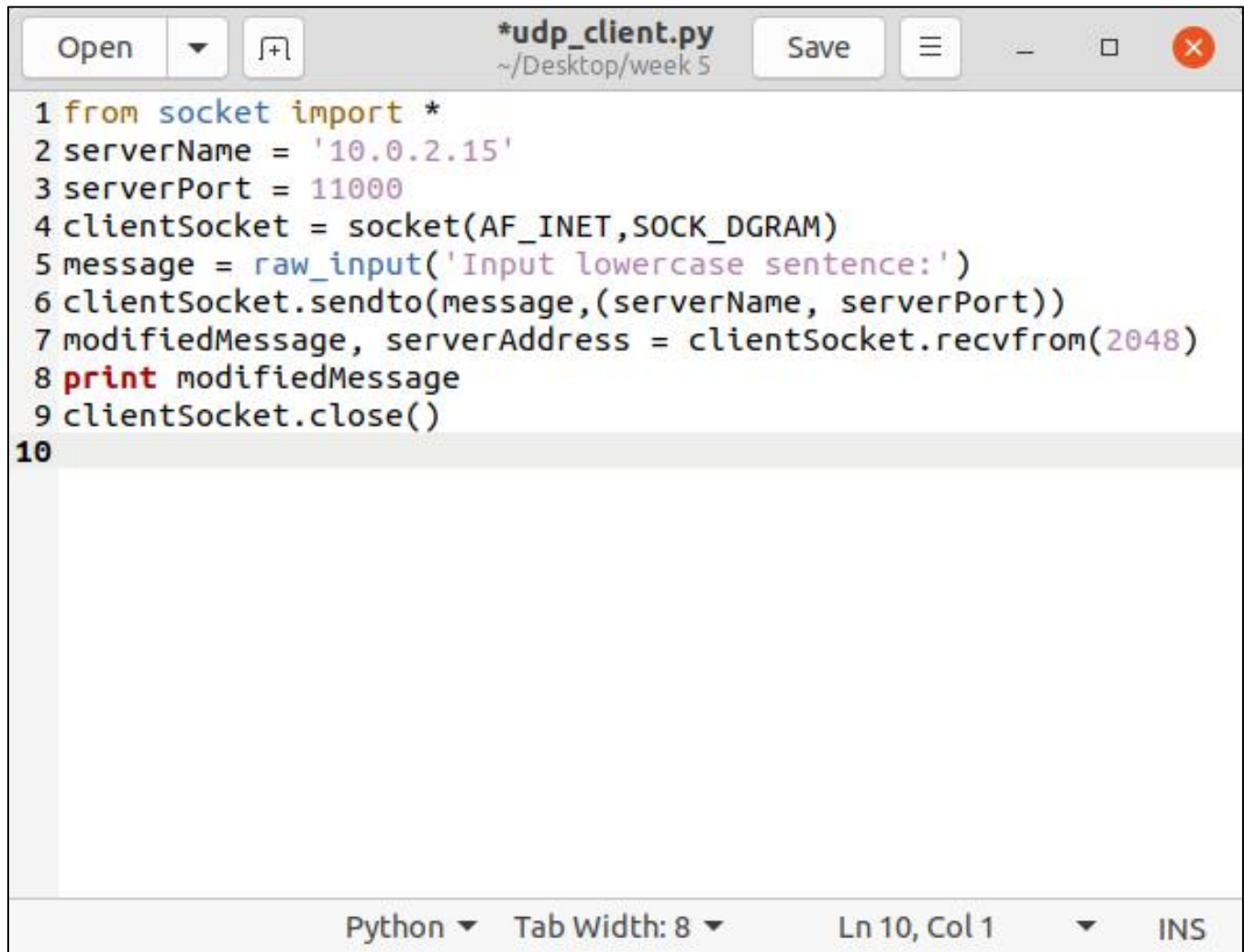


The image shows a code editor window titled "udp_server.py" with a file path of "~/Desktop/week 5". The editor contains a Python script for a UDP server. The script imports the socket module, sets a server port to 11000, creates a socket object, binds it to the IP address '10.0.2.15' and the port, prints a message, and enters a while loop to receive and respond to client messages. The status bar at the bottom indicates the editor is in Python mode with a tab width of 8, and the cursor is at line 10, column 9 in insert mode.

```
1 from socket import *
2 serverPort = 11000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('10.0.2.15', serverPort))
5 print "The server is ready to receive"
6 while 1:
7     message, clientAddress = serverSocket.recvfrom(2048)
8     modifiedMessage = message.upper()
9     serverSocket.sendto(modifiedMessage, clientAddress)
10
```

Python ▾ Tab Width: 8 ▾ Ln 10, Col 9 ▾ INS

1.2.2 UDP Client

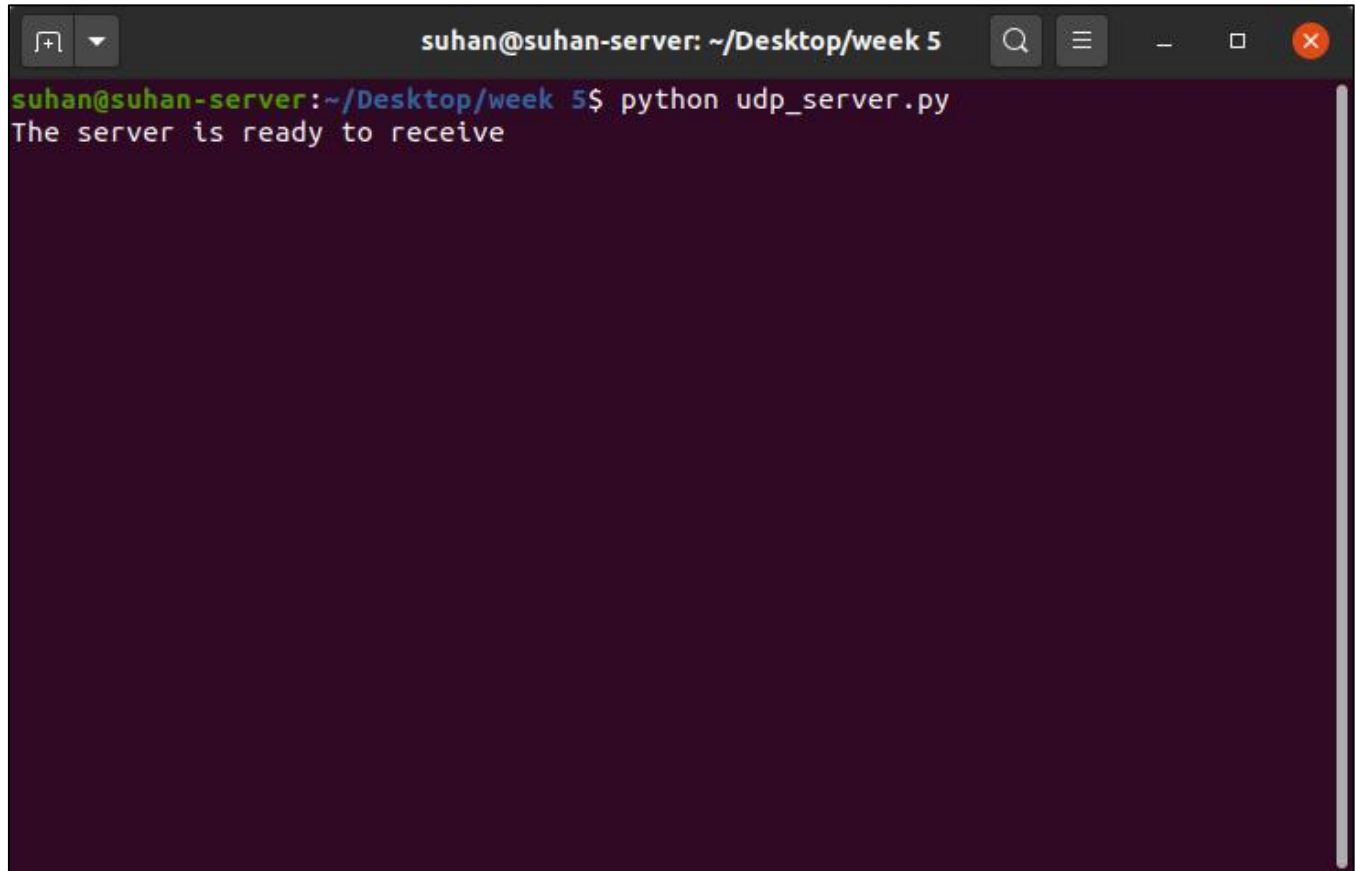


The image shows a code editor window with a title bar containing the filename `*udp_client.py` and the path `~/Desktop/week 5`. The editor has buttons for 'Open', 'Save', and a menu icon. The code is as follows:

```
1 from socket import *
2 serverName = '10.0.2.15'
3 serverPort = 11000
4 clientSocket = socket(AF_INET, SOCK_DGRAM)
5 message = raw_input('Input lowercase sentence:')
6 clientSocket.sendto(message, (serverName, serverPort))
7 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
8 print modifiedMessage
9 clientSocket.close()
10
```

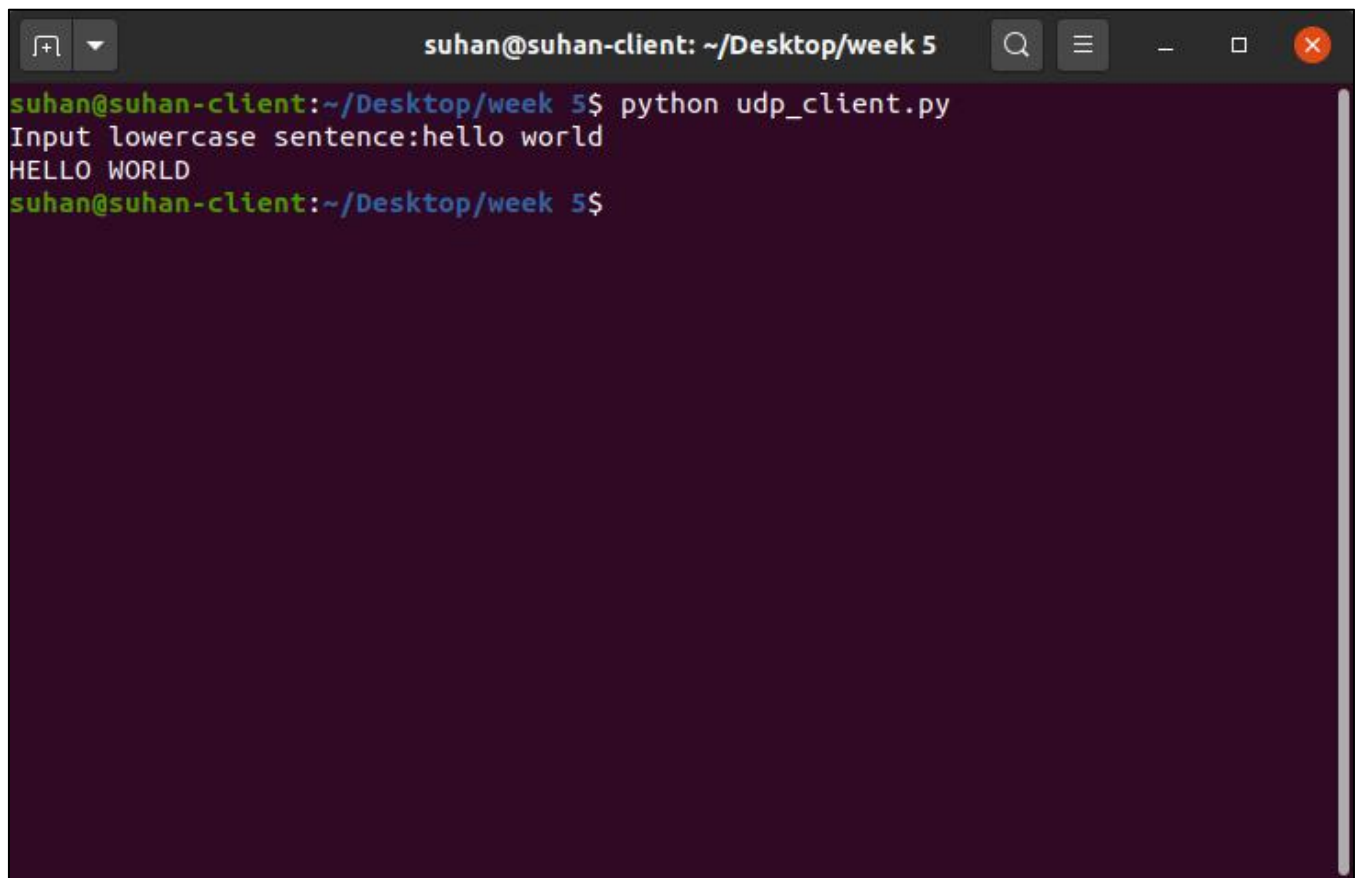
The status bar at the bottom indicates the language is 'Python', the tab width is '8', the cursor is at 'Ln 10, Col 1', and the input mode is 'INS'.

1.2.3 UDP Connection between Server and Client

A terminal window with a dark background and light text. The title bar at the top reads 'suhan@suhan-server: ~/Desktop/week 5'. The terminal content shows a prompt 'suhan@suhan-server:~/Desktop/week 5\$' followed by the command 'python udp_server.py'. Below the command, the output 'The server is ready to receive' is displayed. The window has standard Linux window controls (minimize, maximize, close) on the right side of the title bar.

```
suhan@suhan-server: ~/Desktop/week 5
suhan@suhan-server:~/Desktop/week 5$ python udp_server.py
The server is ready to receive
```

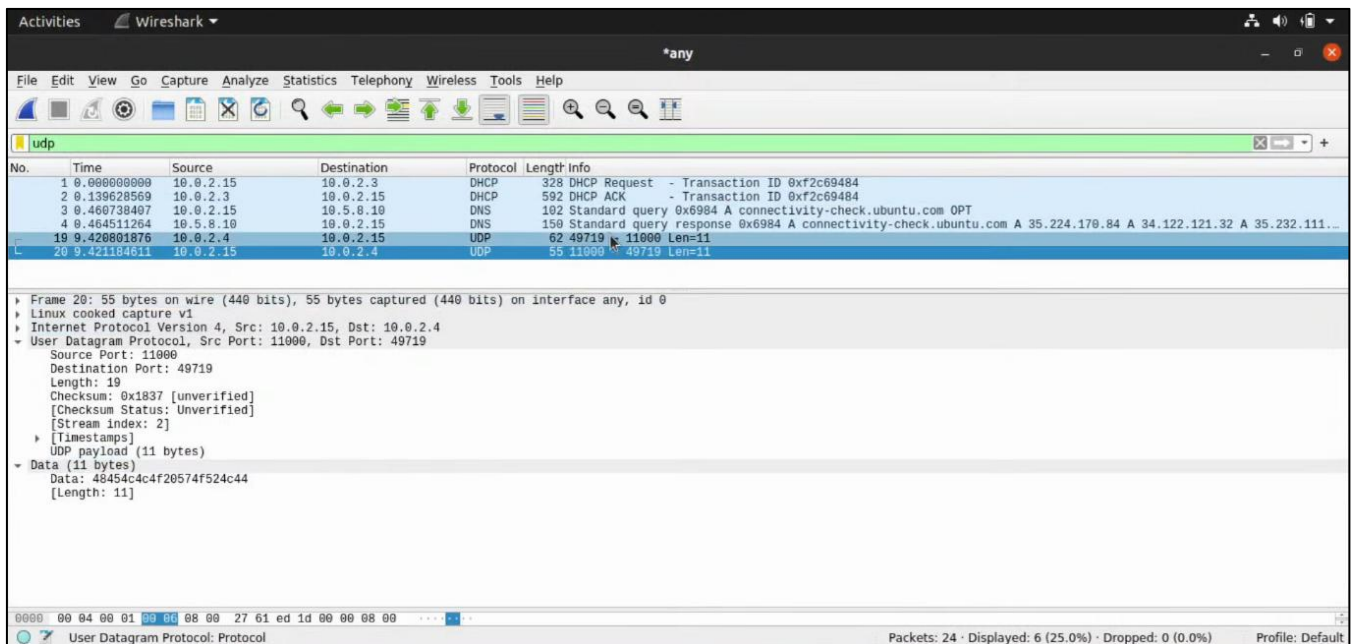
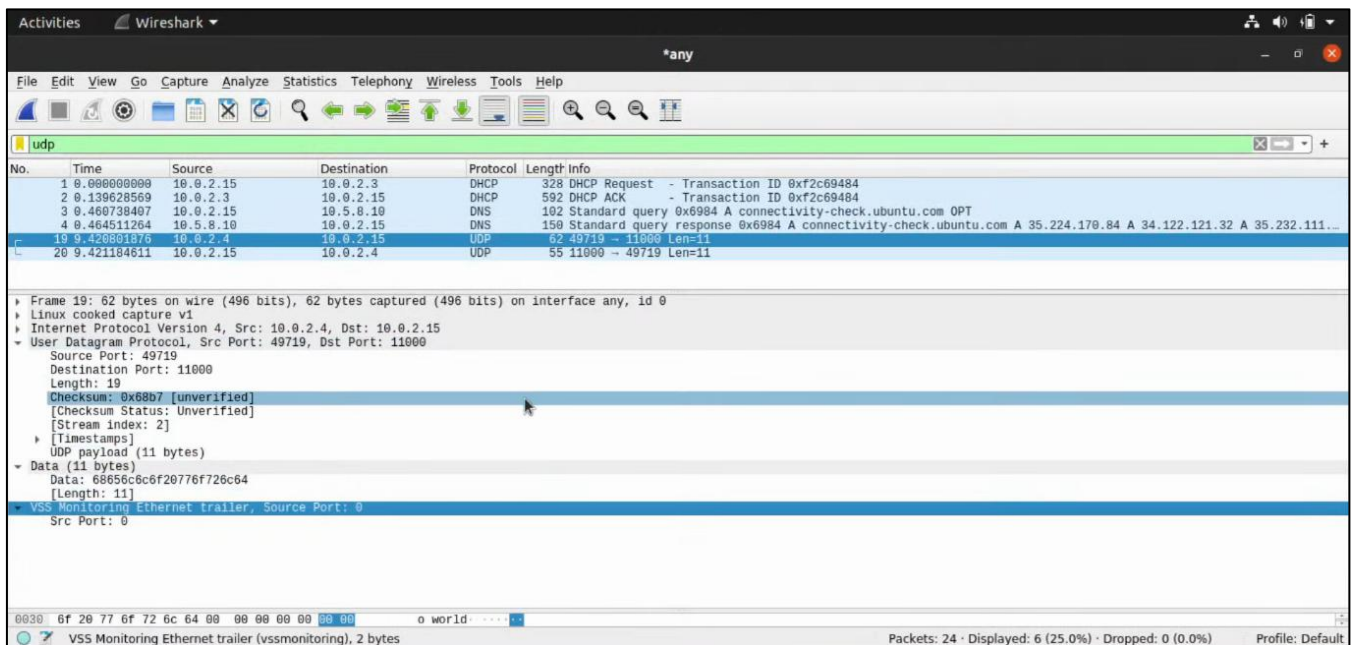
UDP Server

A terminal window with a dark purple background. The title bar at the top shows the user 'suhan' on a machine named 'suhan-client' in the directory '~/Desktop/week 5'. The terminal contains the following text: a prompt 'suhan@suhan-client:~/Desktop/week 5\$' followed by the command 'python udp_client.py', the prompt 'Input lowercase sentence:' followed by the input 'hello world', the output 'HELLO WORLD', and a final prompt 'suhan@suhan-client:~/Desktop/week 5\$'.

```
suhan@suhan-client: ~/Desktop/week 5
suhan@suhan-client:~/Desktop/week 5$ python udp_client.py
Input lowercase sentence:hello world
HELLO WORLD
suhan@suhan-client:~/Desktop/week 5$
```

UDP Client

1.2.4 Wireshark Capture for UDP Connection

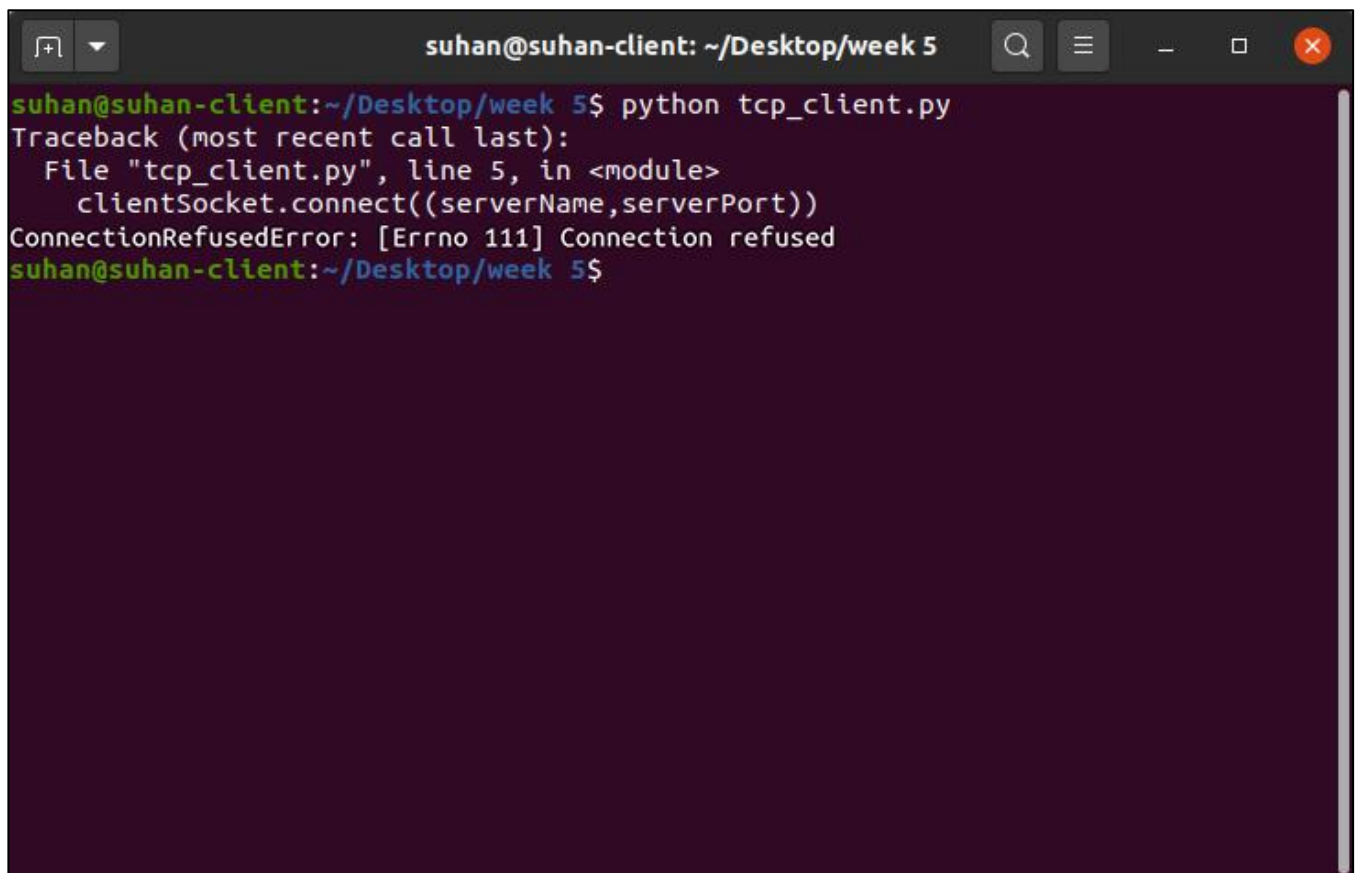


1.3 Problems

Q1. Suppose you run *TCPClient* before you run *TCPServer*. What happens? Why?

Answer – This will lead to a **ConnectionRefusedError**, since the *server socket application* we are trying to connect to has not been initiated and is not listening for connections on the given port number. Hence, any connection requests sent by a client machine at that IP and port number immediately fail since the connection gets refused.

A TCP connection can be established between two socket interfaces only when a host machine listens to requests on a given IP address and port number and accepts connections made by another machine at the same address and port.

A terminal window titled 'suhan@suhan-client: ~/Desktop/week 5' with standard window controls. The terminal shows a command 'python tcp_client.py' being executed, which results in a 'ConnectionRefusedError: [Errno 111] Connection refused'. The error message is displayed in a light blue font against a dark purple background.

```
suhan@suhan-client:~/Desktop/week 5$ python tcp_client.py
Traceback (most recent call last):
  File "tcp_client.py", line 5, in <module>
    clientSocket.connect((serverName,serverPort))
ConnectionRefusedError: [Errno 111] Connection refused
suhan@suhan-client:~/Desktop/week 5$
```

Q2. *Suppose you run UDPClient before you run UDPServer. What happens? Why?*

Answer – No error will be obtained since *UDP does not require a prior connection to be set up between the host machines for data transfer to begin*. It is a connectionless protocol which transfers packets of data to a destination IP and port number without verifying the existence of the connection. Hence, it is prone to data integrity issues such as loss of packets.

If any packets of data are sent before the server is executed, the packets are lost forever and will not reach the server socket application. However, if any packets of data are sent after the server is executed, the client will be able to send packets to a destination server and also receive response packets in return.

Q3. *What happens if you use different port numbers for the client and server sides?*

Answer – This will lead to a **ConnectionRefusedError** for a TCP connection, since the *server socket application we are trying to connect to is not listening for requests at the same port number as the one the client socket application is trying to connect with*. Hence, the connection between the two socket interfaces is never setup and the connection is downright refused.

However, on a UDP connection, *since no prior connection is required to be established between the host machines for data transfer to take place, no error as such is obtained*. Any messages sent by the client are lost since the destination server does not exist.

2. Task 2: Web Server

In this assignment, you will develop a simple Web server in Python that is capable of processing only one request. Specifically, your Web server will

For this assignment, the companion Web site provides the skeleton code for your server.

Your job is to complete the code, run your server, and then test your server by sending requests from browsers running on different hosts. If you run your server on a host that already has a web server .

Web server running on it, then you should use a different port than port 80 for your Web server.

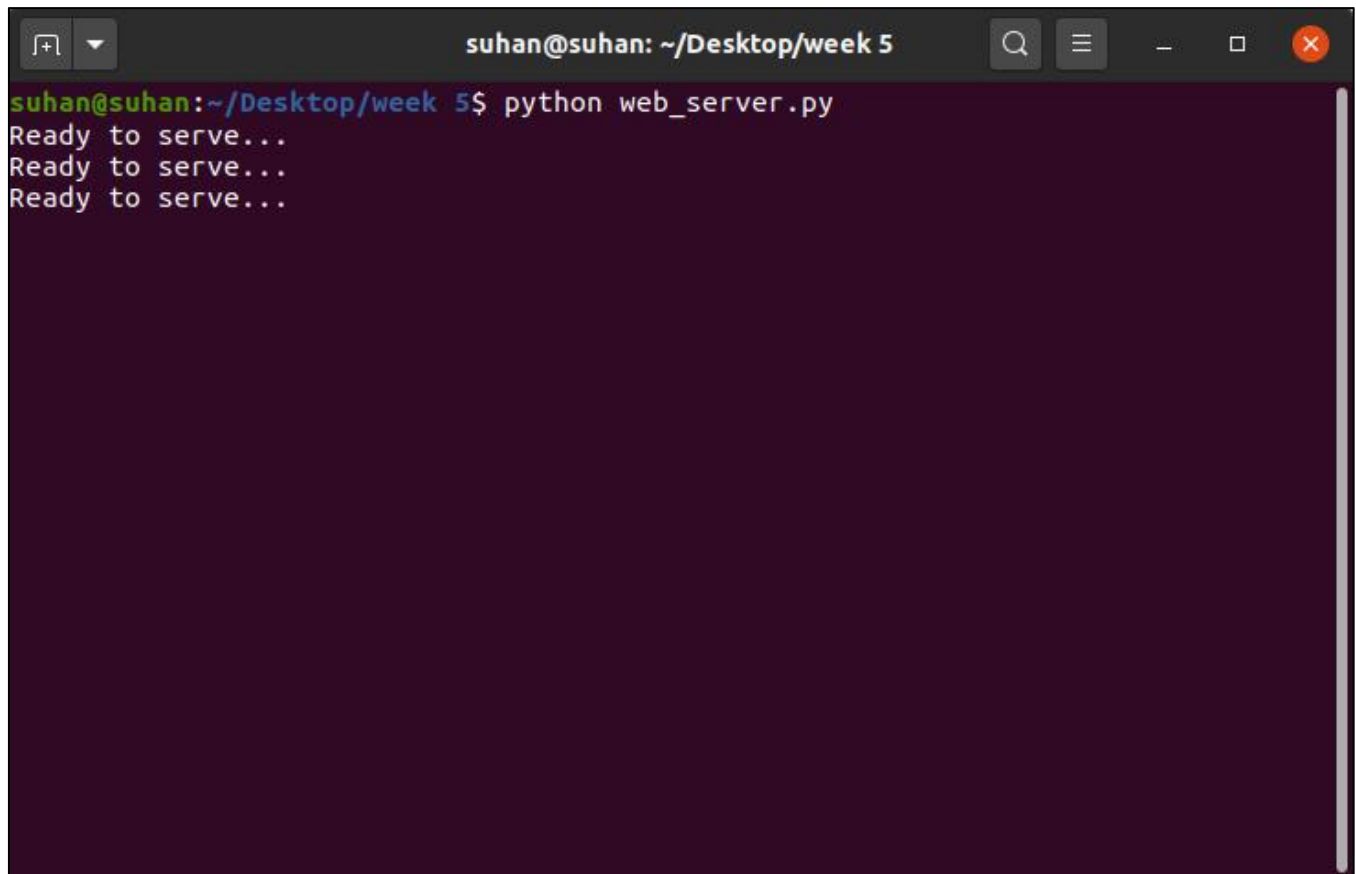
a) create a connection socket when contacted by a client (browser);

```
web_server.py
~/Desktop/week 5
Save

1 # Import socket module
2 from socket import *
3
4 # Create a TCP server socket
5 # (AF_INET is used for IPv4 protocols)
6 # (SOCK_STREAM is used for TCP)
7
8 serverSocket = socket(AF_INET, SOCK_STREAM)
9
10 # Assign a port number
11 serverPort = 3000
12
13 # Bind the socket to server address and server port
14 serverSocket.bind(("10.0.2.15", serverPort))
15
16 # Listen to at most 1 connection at a time
17 serverSocket.listen(1)
18
19 # Server should be up and running and listening to the incoming connections
20 while True:
21     print 'Ready to serve...'
22
23     # Set up a new connection from the client
24     connectionSocket, addr = serverSocket.accept()
25
26     # If an exception occurs during the execution of try clause
27     # the rest of the clause is skipped
28     # If the exception type matches the word after except
29     # the except clause is executed
30     try:
31         # Receives the request message from the client
32         message = connectionSocket.recv(1024)
33         # Extract the path of the requested object from the message
34         # The path is the second part of HTTP header, identified by [1]
35         filename = message.split()[1]
36
37         # Because the extracted path of the HTTP request includes
38         # a character '/', we read the path from the second character
39         f = open(filename[1:])
40         # Store the entire content of the requested file in a temporary buffer
41         outputdata = f.read()
42         # Send the HTTP response header line to the connection socket
43         connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")
44
45         # Send the content of the requested file to the connection socket
46         for i in range(0, len(outputdata)):
47             connectionSocket.send(outputdata[i])
48             connectionSocket.send("\r\n")
49
50         # Close the client connection socket
51         connectionSocket.close()
52
53     except IOError:
54         # Send HTTP response message for file not found
55         connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
56         connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n")
57         # Close the client connection socket
58         connectionSocket.close()
59
60 serverSocket.close()
61
```

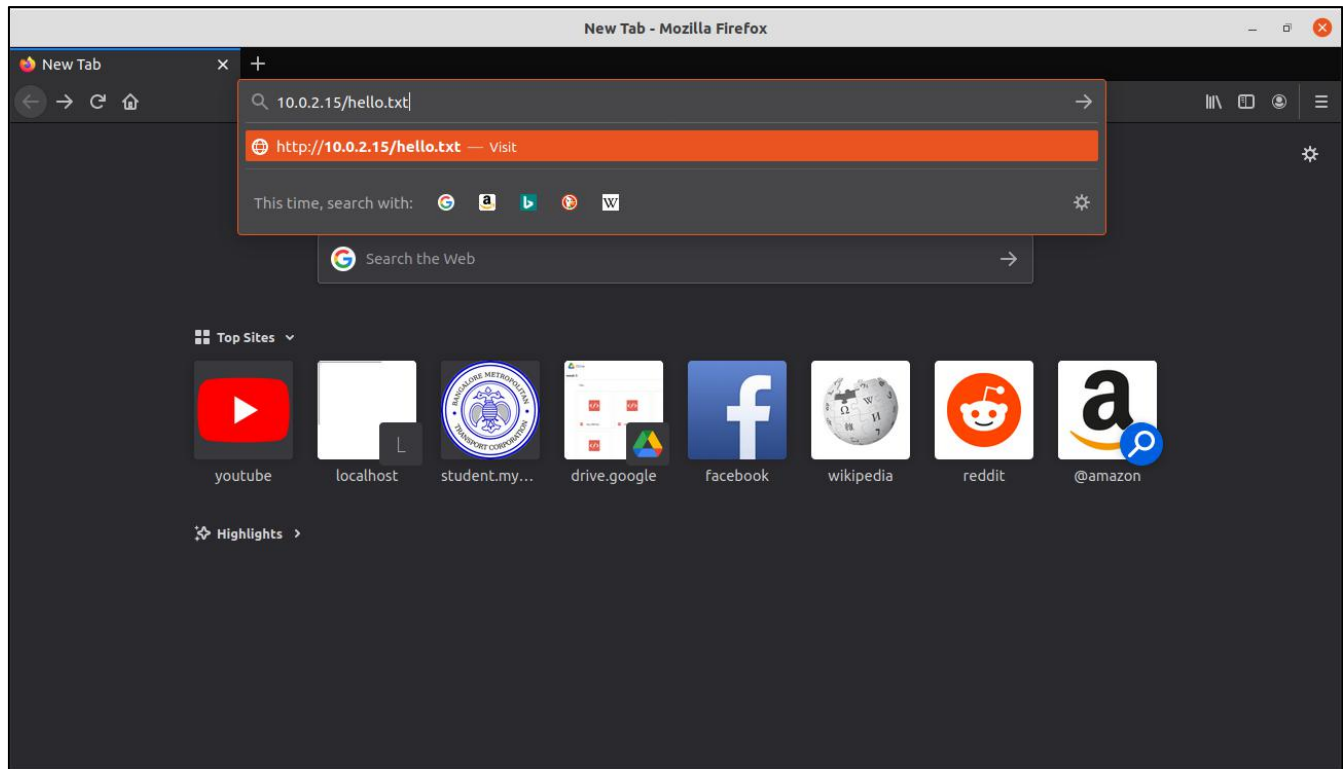
Python Tab Width: 8 Ln 1, Col 1 INS

b) receive the HTTP request from this connection;

A terminal window with a dark background and light text. The window title bar shows 'suhan@suhan: ~/Desktop/week 5' and standard window controls. The terminal content shows the command 'python web_server.py' being executed, followed by three lines of 'Ready to serve...' output.

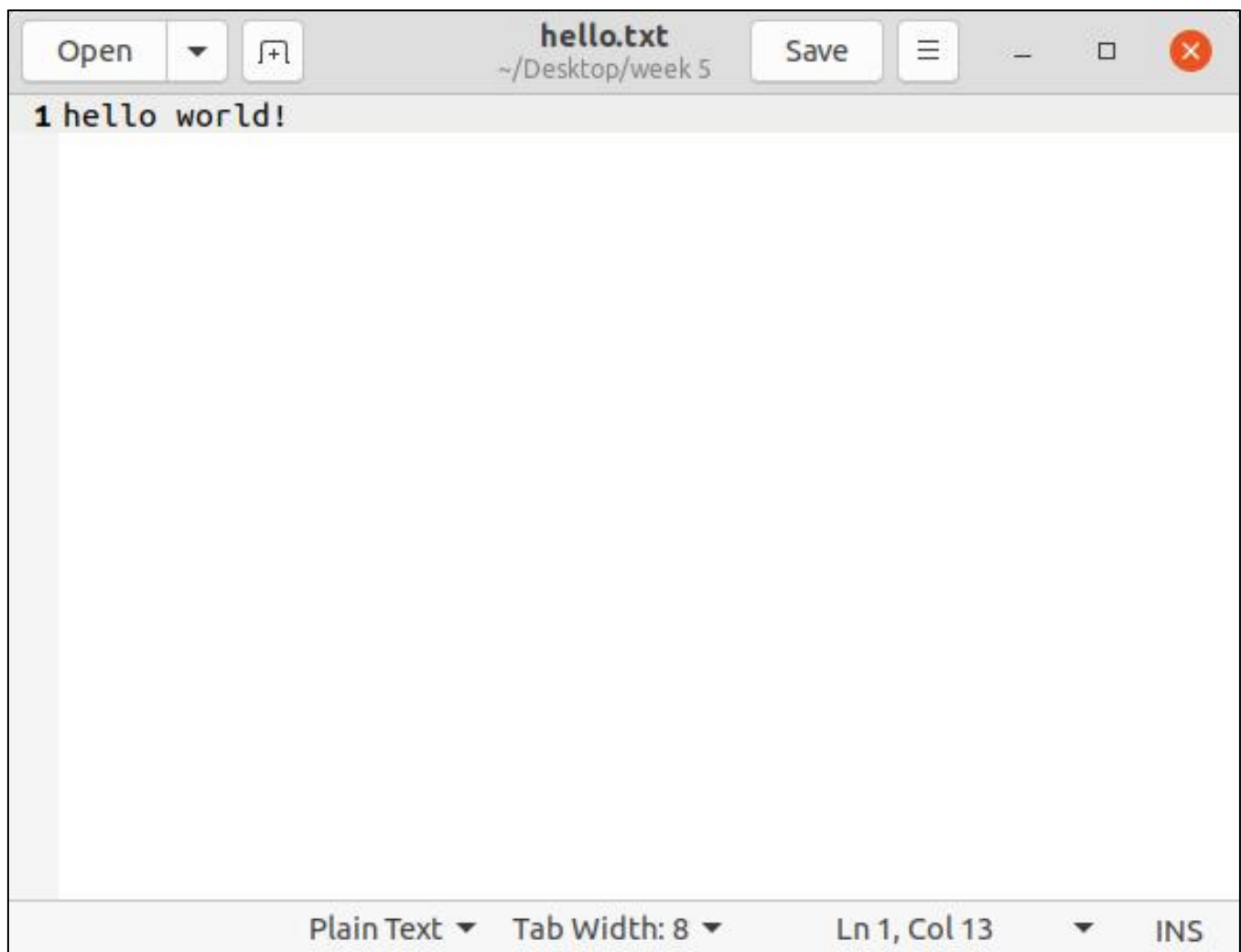
```
suhan@suhan: ~/Desktop/week 5$ python web_server.py
Ready to serve...
Ready to serve...
Ready to serve...
```

C) parse the request to determine the specific file being requested;



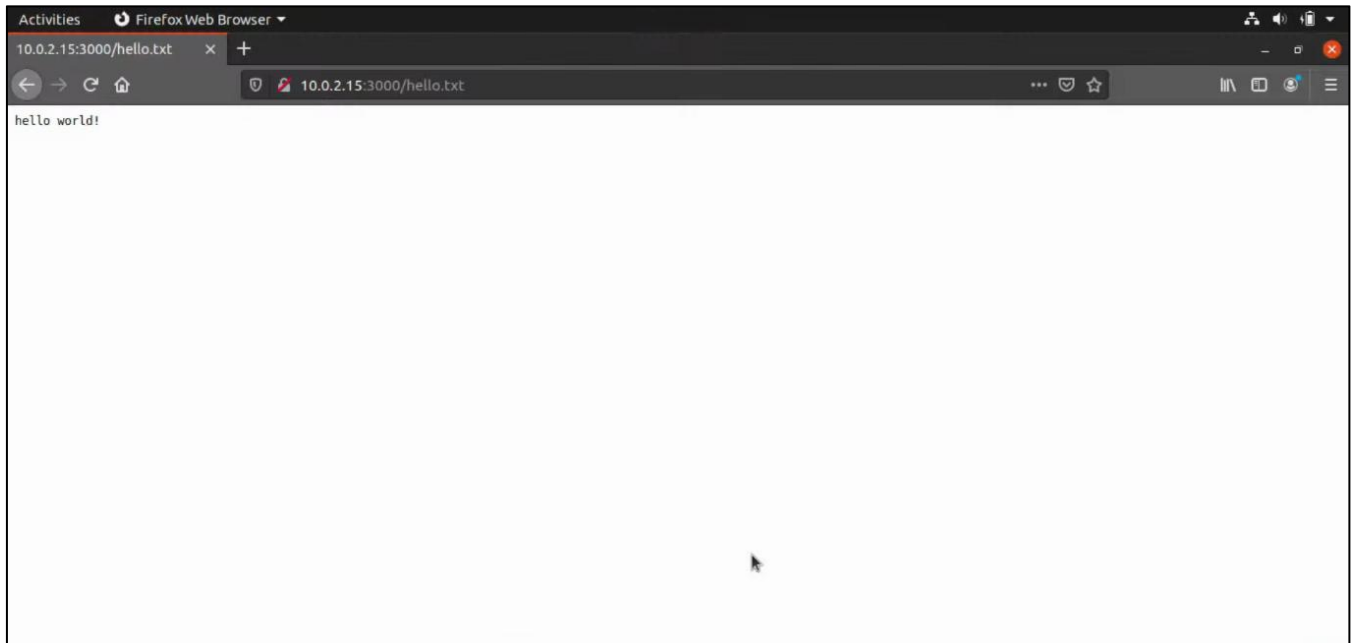
Web Browser requesting the file in server

D) get the requested file from the server's file system;



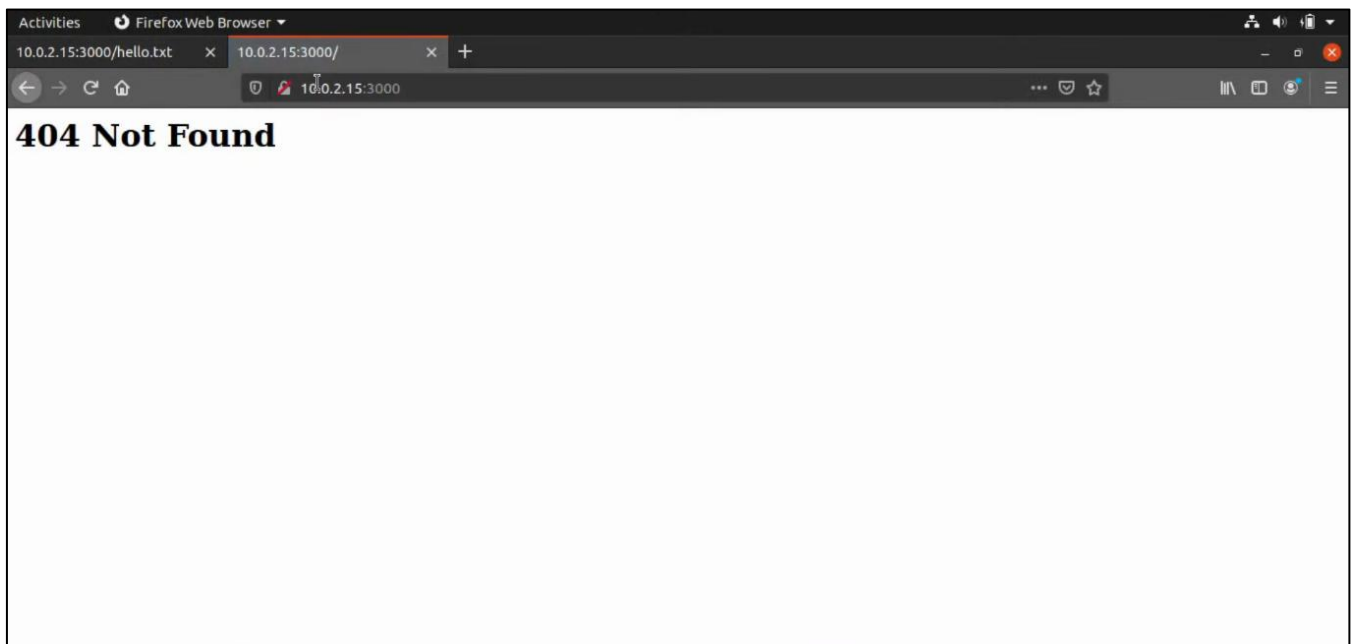
Data File consisting message present in same folder

e) create an HTTP response message consisting of the requested file preceded by header lines;



Message in file being displayed in web browser

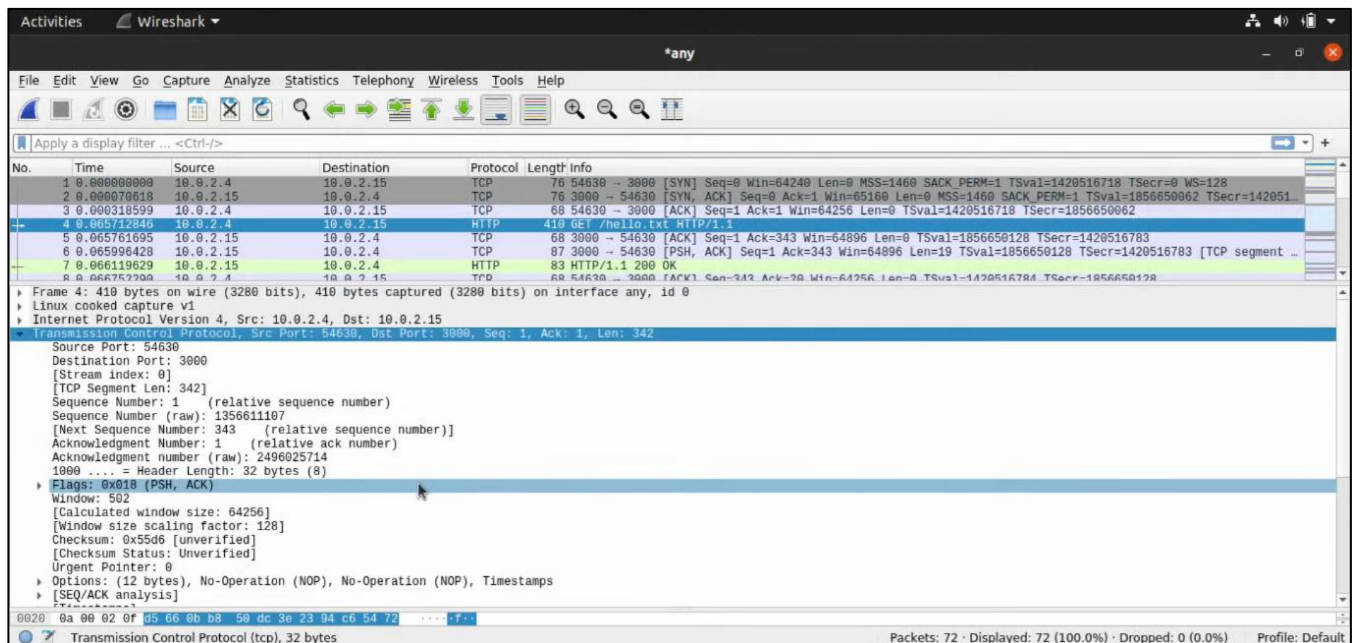
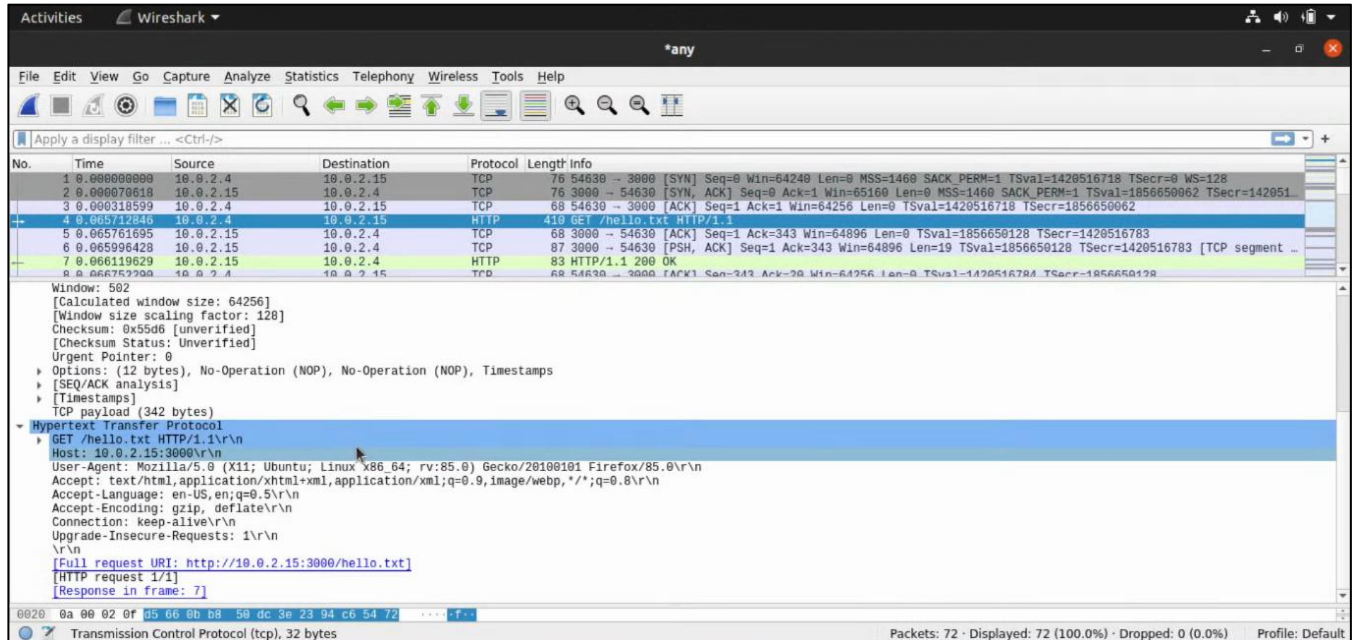
f) send the response over the TCP connection to the requesting browser.
If a browser requests a file that is not present in your server, your server should return a **“404 Not Found”** error message.



Message displayed for invalid

2.3 Wireshark Capture for Web Server

The Wireshark Packet Captures similarly shows the request and response packets received by the web server from the server and client, respectively..



Wireshark Capture for Server Machine

Activities Wireshark

*any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	TCP	76	54630 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1420516718 TSecr=0 WS=128
2	0.000070618	10.0.2.15	10.0.2.4	TCP	76	3000 → 54630 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1856650062 TSecr=142051...
3	0.000318599	10.0.2.4	10.0.2.15	TCP	68	54630 → 3000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1420516718 TSecr=1856650062
4	0.005712846	10.0.2.4	10.0.2.15	HTTP	410	GET /hello.txt HTTP/1.1
5	0.005761695	10.0.2.15	10.0.2.4	TCP	68	3000 → 54630 [ACK] Seq=1 Ack=343 Win=64896 Len=0 TSval=1856650128 TSecr=1420516783
6	0.005996428	10.0.2.15	10.0.2.4	TCP	87	3000 → 54630 [PSH, ACK] Seq=1 Ack=343 Win=64896 Len=19 TSval=1856650128 TSecr=1420516783 [TCP segment ...
7	0.006110629	10.0.2.15	10.0.2.4	HTTP	83	HTTP/1.1 200 OK

Frame 7: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4

Transmission Control Protocol, Src Port: 3000, Dst Port: 54630, Seq: 20, Ack: 343, Len: 15

- Source Port: 3000
- Destination Port: 54630
- [Stream index: 0]
- [TCP Segment Len: 15]
- Sequence Number: 20 (relative sequence number)
- Sequence Number (raw): 2496925733
- [Next Sequence Number: 36 (relative sequence number)]
- Acknowledgment Number: 343 (relative ack number)
- Acknowledgment number (raw): 1356611449
- 1000 = Header Length: 32 bytes (8)
- Flags: 0x019 (FIN, PSH, ACK)
- Window: 507
- [Calculated window size: 64896]
- [Window size scaling factor: 128]
- Checksum: 0x1848 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
- [SEQ/ACK analysis]

Frame 63 bytes / reassembled TCP (24 bytes)

Transmission Control Protocol (tcp), 32 bytes

Packets: 72 · Displayed: 72 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Activities Wireshark

*any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.4	10.0.2.15	TCP	76	54630 → 3000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1420516718 TSecr=0 WS=128
2	0.000070618	10.0.2.15	10.0.2.4	TCP	76	3000 → 54630 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1856650062 TSecr=142051...
3	0.000318599	10.0.2.4	10.0.2.15	TCP	68	54630 → 3000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1420516718 TSecr=1856650062
4	0.005712846	10.0.2.4	10.0.2.15	HTTP	410	GET /hello.txt HTTP/1.1
5	0.005761695	10.0.2.15	10.0.2.4	TCP	68	3000 → 54630 [ACK] Seq=1 Ack=343 Win=64896 Len=0 TSval=1856650128 TSecr=1420516783
6	0.005996428	10.0.2.15	10.0.2.4	TCP	87	3000 → 54630 [PSH, ACK] Seq=1 Ack=343 Win=64896 Len=19 TSval=1856650128 TSecr=1420516783 [TCP segment ...
7	0.006110629	10.0.2.15	10.0.2.4	HTTP	83	HTTP/1.1 200 OK

Frame 7: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.4

Transmission Control Protocol, Src Port: 3000, Dst Port: 54630, Seq: 20, Ack: 343, Len: 15

- Acknowledgment Number: 343 (relative ack number)
- Acknowledgment number (raw): 1356611449
- 1000 = Header Length: 32 bytes (8)
- Flags: 0x019 (FIN, PSH, ACK)
- Window: 507
- [Calculated window size: 64896]
- [Window size scaling factor: 128]
- Checksum: 0x1848 [unverified]
- [Checksum Status: Unverified]
- Urgent Pointer: 0
- Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
- [SEQ/ACK analysis]
- [Timestamps]
- [2 Reassembled TCP Segments (34 bytes): #6(19), #7(15)]
- Hypertext Transfer Protocol
- HTTP/1.1 200 OK\r\n
- \r\n
- [HTTP response 1/1]
- [Time since request: 0.000406783 seconds]
- [Request in frame: 4]
- [Request URI: http://10.0.2.15:3000/hello.txt]
- File Data: 15 bytes
- Data (15 bytes)

Frame 63 bytes / reassembled TCP (24 bytes)

Transmission Control Protocol (tcp), 32 bytes

Packets: 72 · Displayed: 72 (100.0%) · Dropped: 0 (0.0%) Profile: Default

Wireshark Capture for Client Machine