



COMPUTER NETWORKS LAB

COURSE CODE:UE19CS255

NAME:PRIYA MOHATA

SRN:PES2UG19CS301

SECTION:E

DATE:21/02/2021

**EXPERIMENT : Simple Client-Server Application using
Network Socket Programming**

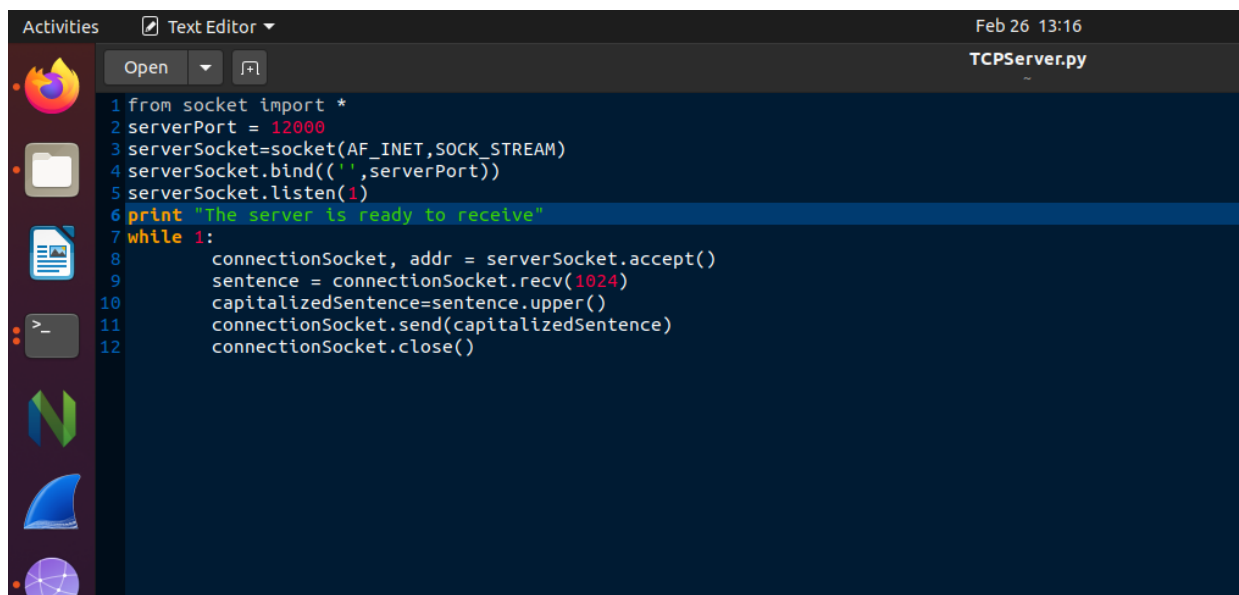
TASK 1: SOCKET PROGRAMMING

1. Create an application that will
 1. Convert lowercase letters to uppercase
 - e.g. [a...z] to [A...Z]
 - code will not change any special characters, e.g. &*!
 2. 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 IN THE SAME SYSTEM

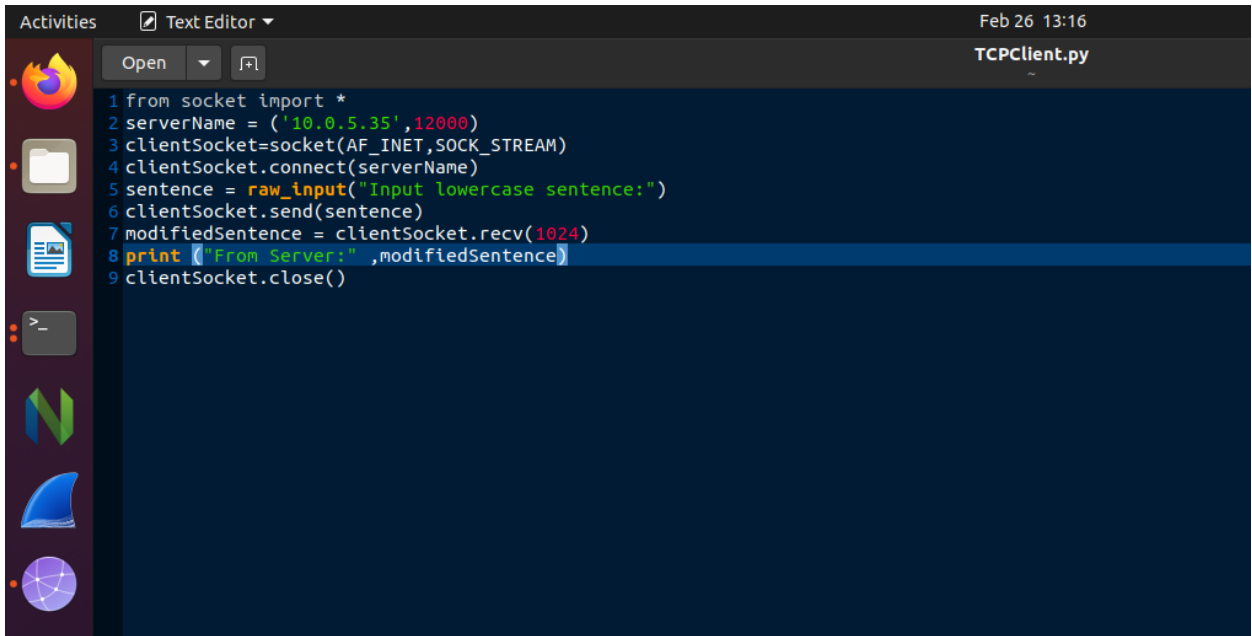
- A TCP connection can be made between two machines with the help of a socket interface using the socket library on Python3.
- To create a TCP socket interface, the type of socket needs to be set as SOCK_STREAM.
- The type of addresses needs to be set as AF_INET which corresponds to IPv4.
- Once the 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

A screenshot of a Linux desktop environment showing a text editor window titled 'Text Editor' with the file 'TCPServer.py' open. The editor displays Python code for a TCP server. The code includes imports, setting a server port to 12000, creating a socket with AF_INET and SOCK_STREAM, binding it to all interfaces on that port, listening for connections, and a loop that accepts connections, receives data, capitalizes it, sends it back, and closes the connection socket. The desktop background is dark, and various application icons are visible on the left sidebar.

```
1 from socket import *
2 serverPort = 12000
3 serverSocket=socket(AF_INET,SOCK_STREAM)
4 serverSocket.bind(('',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()
```

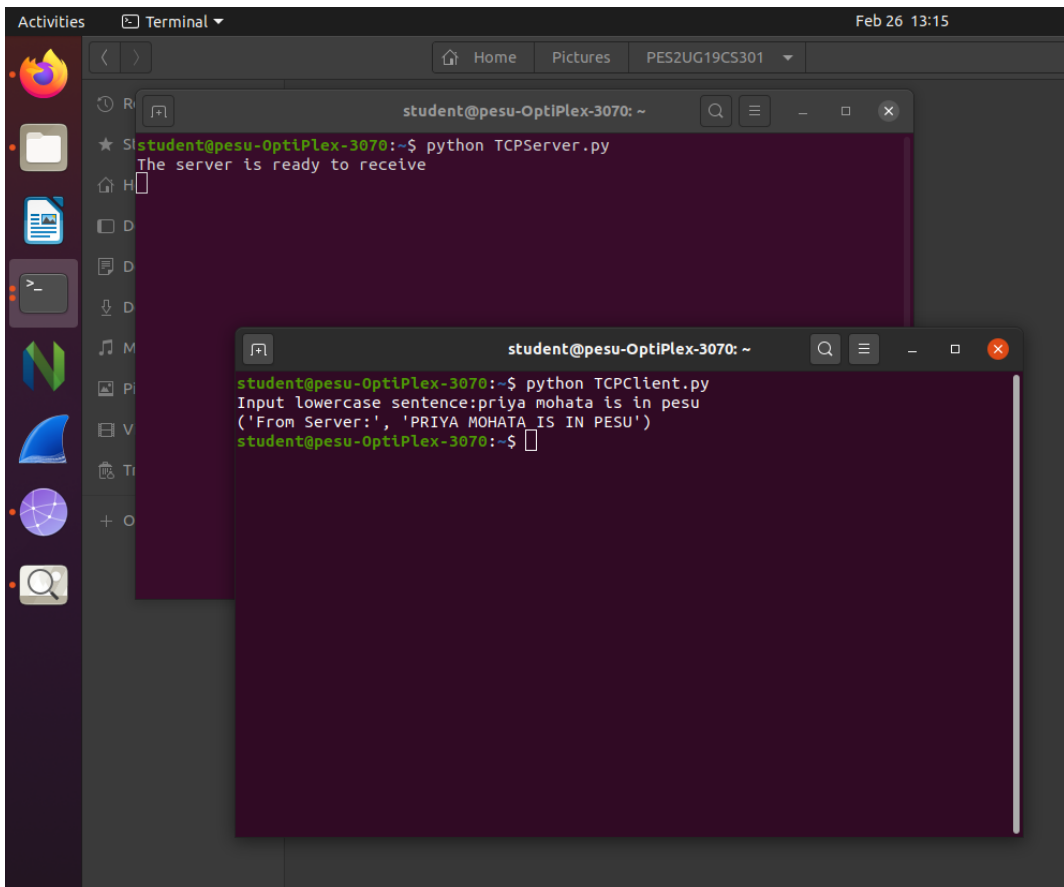
1.1.2 TCP CLIENT

A screenshot of a Linux desktop environment. The top panel shows the 'Activities' button, a 'Text Editor' window title, and the date/time 'Feb 26 13:16'. The text editor window is titled 'TCPClient.py' and contains the following Python code:

```
1 from socket import *
2 serverName = ('10.0.5.35',12000)
3 clientSocket=socket(AF_INET,SOCK_STREAM)
4 clientSocket.connect(serverName)
5 sentence = raw_input("Input lowercase sentence:")
6 clientSocket.send(sentence)
7 modifiedSentence = clientSocket.recv(1024)
8 print ("From Server:" ,modifiedSentence)
9 clientSocket.close()
```

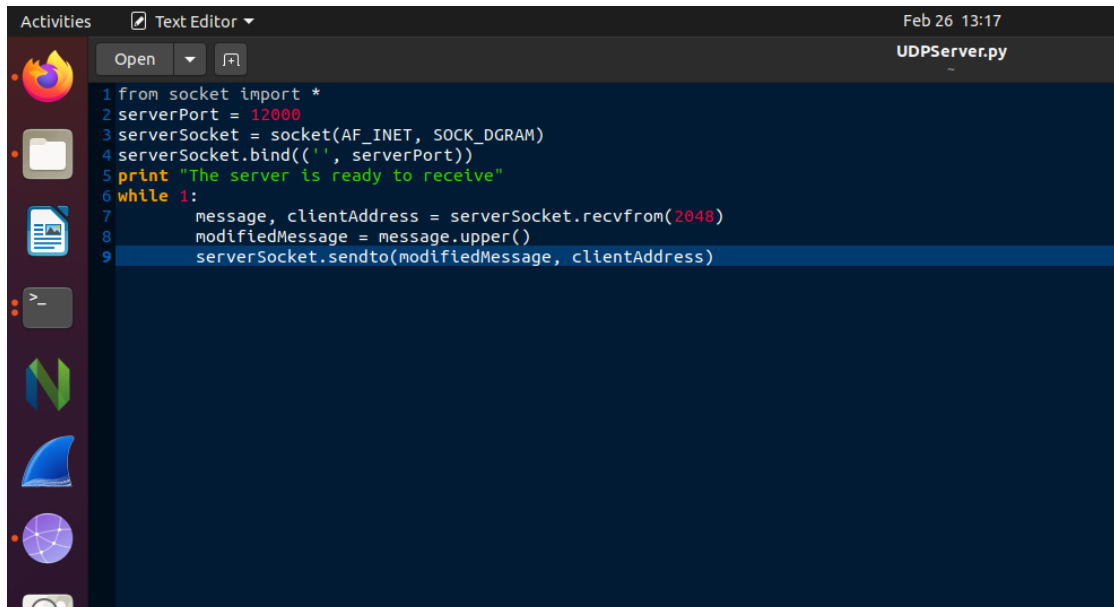
The left sidebar of the desktop shows icons for Firefox, Files, Text Editor, Terminal, and other applications.

1.1.3 TCP CONNECTION BETWEEN TCP SERVER AND TCP CLIENT

A screenshot of a Linux desktop environment showing two terminal windows. The top terminal window is titled 'student@pesu-OptiPlex-3070: ~' and shows the output of running 'python TCPServer.py', which is 'The server is ready to receive'. The bottom terminal window is also titled 'student@pesu-OptiPlex-3070: ~' and shows the output of running 'python TCPClient.py'. The client prompts for an input, 'Input lowercase sentence:priya mohata is in pesu', and then displays the received message, '('From Server:', 'PRIYA MOHATA IS IN PESU')'. The desktop background is dark, and the left sidebar shows icons for various applications.

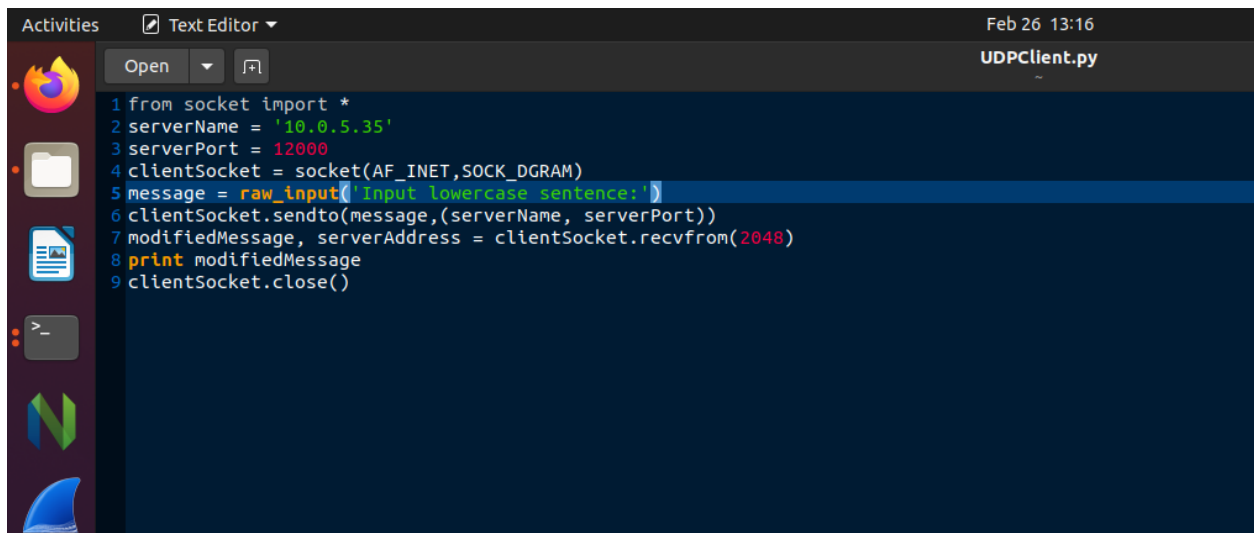
1.2 UDP CONNECTION IN THE SAME SYSTEM

1.2.1 UDP SERVER

A screenshot of a Linux desktop environment showing a text editor window titled 'UDPServer.py'. The editor contains Python code for a UDP server. The code imports the socket module, sets a server port of 12000, binds the socket to all interfaces on that port, and enters a loop to receive and echo back messages in uppercase. The desktop background is dark, and various application icons are visible on the left sidebar.

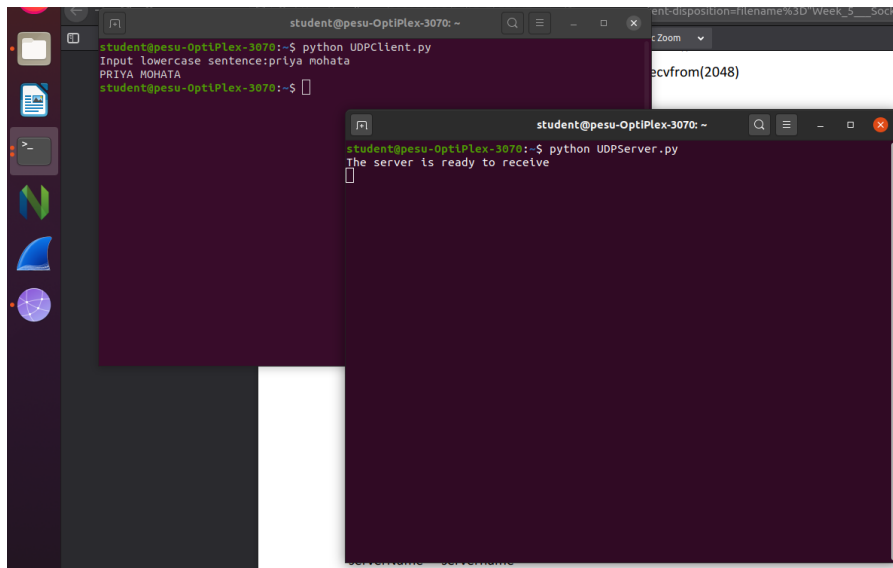
```
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('', serverPort))
5 print "The server is ready to receive"
6 while 1:
7     message, clientAddress = serverSocket.recvfrom(2048)
8     modifiedMessage = message.upper()
9     serverSocket.sendto(modifiedMessage, clientAddress)
```

1.2.2 UDP CLIENT

A screenshot of a Linux desktop environment showing a text editor window titled 'UDPClient.py'. The editor contains Python code for a UDP client. The code sets a server name of '10.0.5.35' and a server port of 12000, creates a client socket, sends a message (prompting user input), receives the modified message back, and prints it. The desktop background is dark, and various application icons are visible on the left sidebar.

```
1 from socket import *
2 serverName = '10.0.5.35'
3 serverPort = 12000
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()
```

1.2.3 UDP CONNECTION BETWEEN UDP SERVER AND UDP CLIENT



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window has a title bar that reads "student@pesu-OptiPlex-3070: ~". The terminal output is as follows:

```
student@pesu-OptiPlex-3070:~$ python UDPClient.py
Input lowercase sentence:priya mohata
PRIYA MOHATA
student@pesu-OptiPlex-3070:~$
```

Below this, another terminal window is shown, also titled "student@pesu-OptiPlex-3070: ~". Its output is:

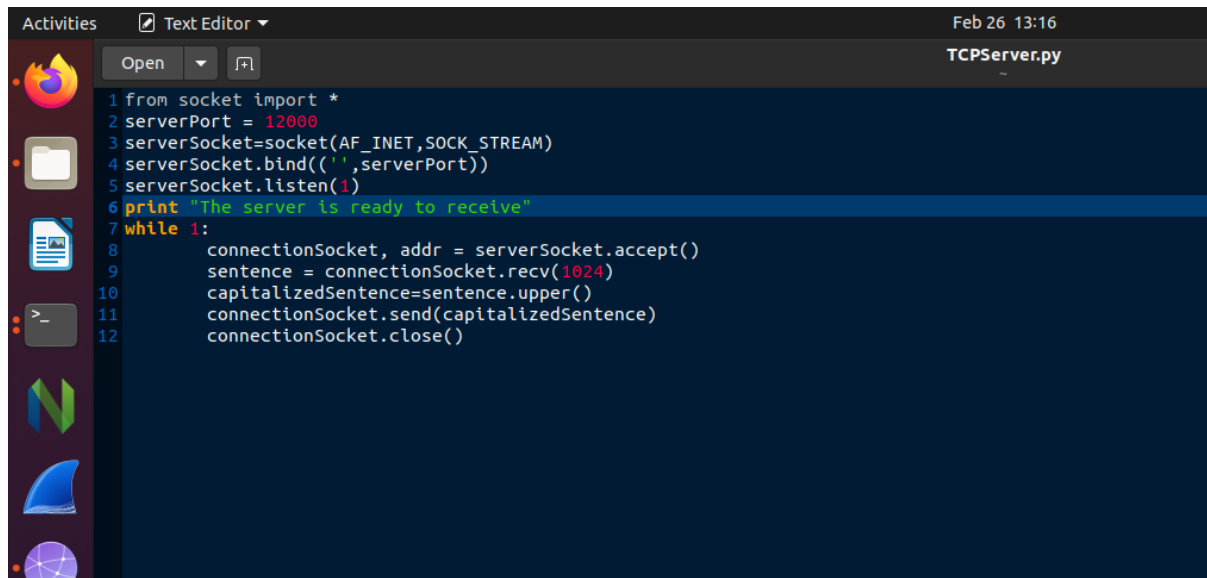
```
student@pesu-OptiPlex-3070:~$ python UDPServer.py
The server is ready to receive

```

The desktop background is dark, and the terminal windows have a dark theme. The terminal text is in a monospaced font, with the prompt character being a green dollar sign.

1.3 TCP CONNECTION in the two systems

1.3.1 TCP SERVER



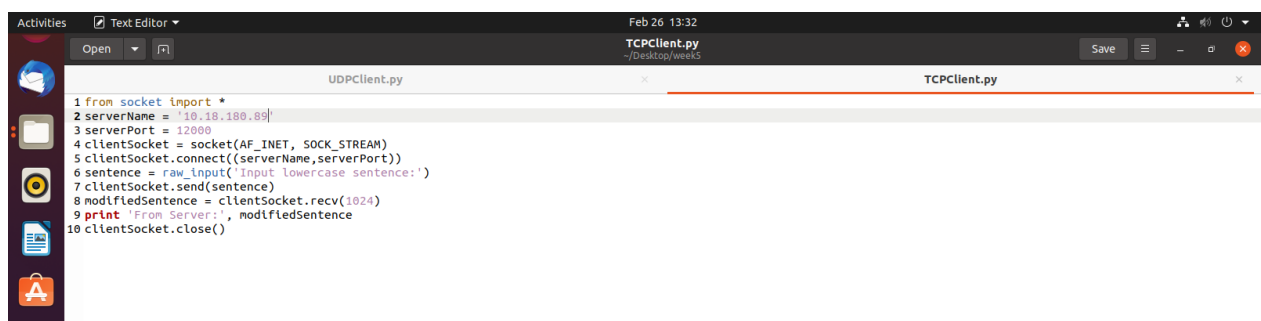
The screenshot shows a text editor window titled 'Text Editor' with a file named 'TCPServer.py'. The code is as follows:

```

1 from socket import *
2 serverPort = 12000
3 serverSocket=socket(AF_INET,SOCK_STREAM)
4 serverSocket.bind(('',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()

```

1.3.2 TCP CLIENT



The screenshot shows a text editor window titled 'Text Editor' with a file named 'TCPClient.py'. The code is as follows:

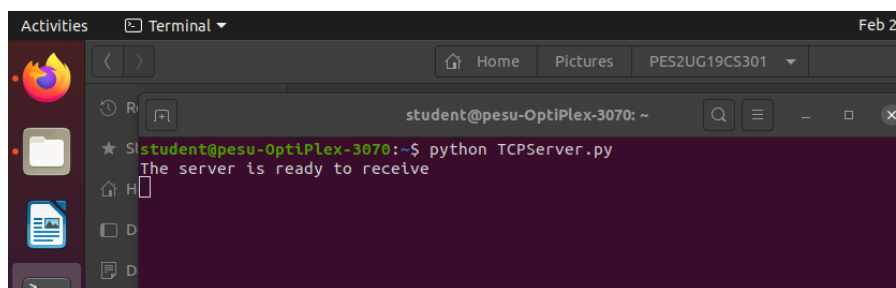
```

1 from socket import *
2 serverName = '10.18.180.89'
3 serverPort = 12000
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()

```

1.3.3 TCP CONNECTION

SERVER



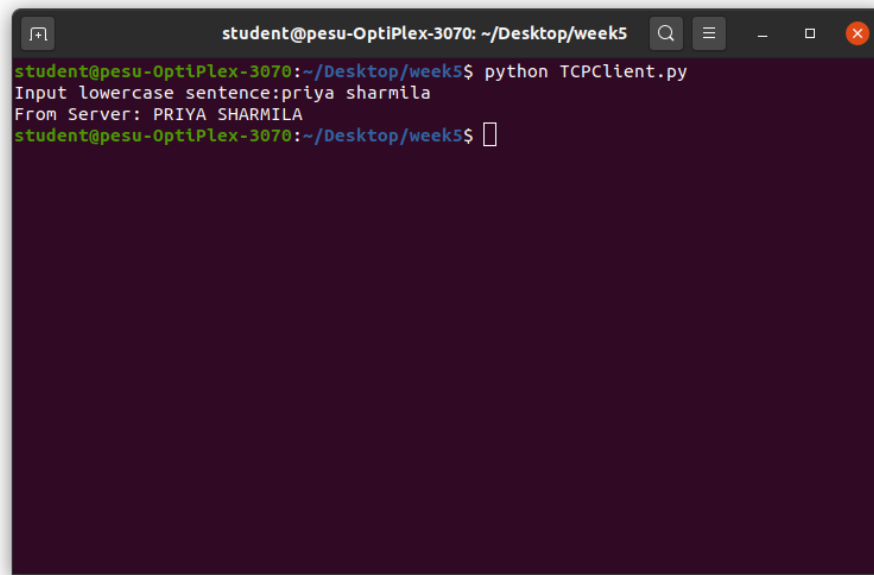
The screenshot shows a terminal window titled 'Terminal' with the following output:

```

student@pesu-OptiPlex-3070: ~
$ python TCPServer.py
The server is ready to receive

```

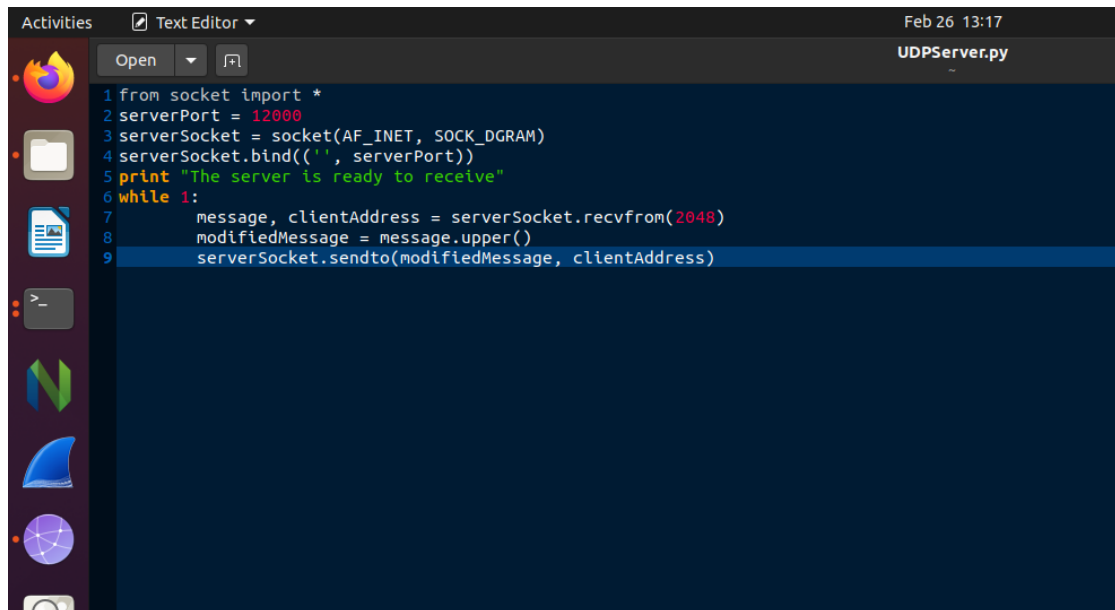
CLIENT

A terminal window titled 'student@pesu-OptiPlex-3070: ~/Desktop/week5' with standard window controls. The terminal shows a command 'python TCPClient.py' being executed. The output consists of two lines: 'Input lowercase sentence:priya sharmila' and 'From Server: PRIYA SHARMILA'. The prompt returns to the shell.

```
student@pesu-OptiPlex-3070: ~/Desktop/week5
student@pesu-OptiPlex-3070:~/Desktop/week5$ python TCPClient.py
Input lowercase sentence:priya sharmila
From Server: PRIYA SHARMILA
student@pesu-OptiPlex-3070:~/Desktop/week5$
```

1.4 UDP CONNECTION in two systems

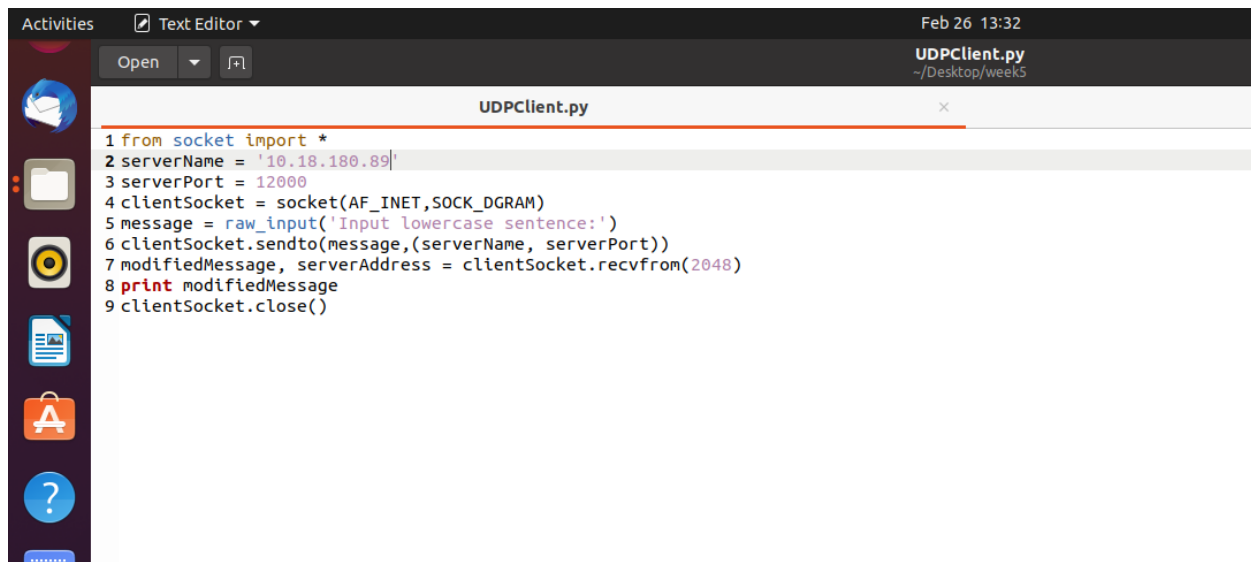
1.4.1 UDP SERVER



A screenshot of a Linux desktop environment showing a text editor window titled 'UDPServer.py'. The editor contains the following Python code:

```
1 from socket import *
2 serverPort = 12000
3 serverSocket = socket(AF_INET, SOCK_DGRAM)
4 serverSocket.bind(('', 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)
```

1.4.2 UDP CLIENT

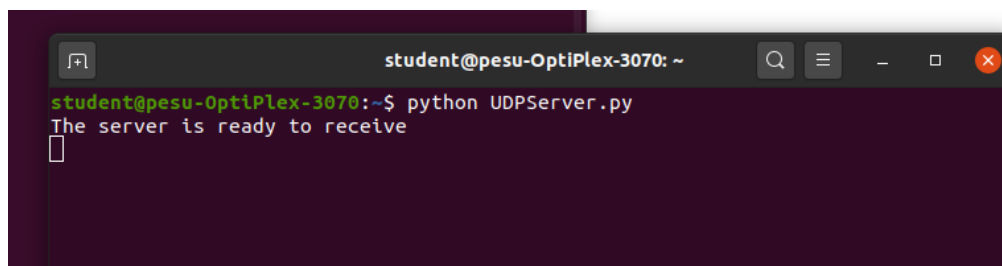


A screenshot of a Linux desktop environment showing a text editor window titled 'UDPClient.py'. The editor contains the following Python code:

```
1 from socket import *
2 serverName = '10.18.180.89'
3 serverPort = 12000
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()
```

1.4.3 UDP CONNECTION

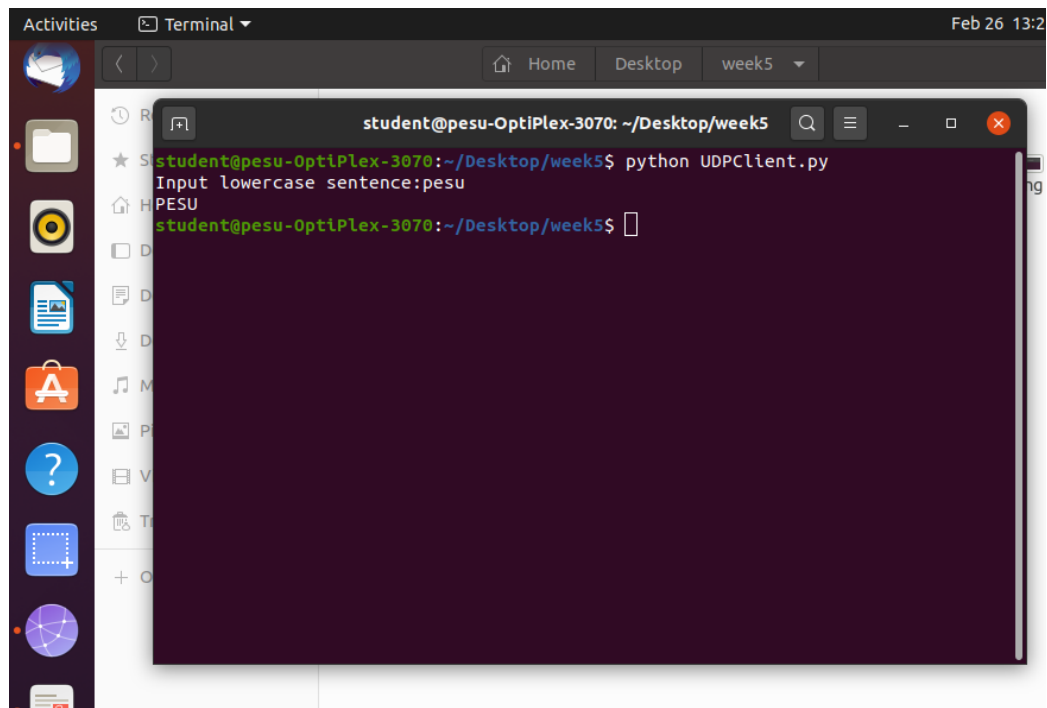
SERVER



A screenshot of a terminal window titled 'student@pesu-OptiPlex-3070: ~'. The terminal shows the command 'python UDPServer.py' being executed, and the output 'The server is ready to receive'.

```
student@pesu-OptiPlex-3070: ~$ python UDPServer.py
The server is ready to receive
```


CLIENT



The screenshot shows a Linux desktop environment with a terminal window open. The terminal title bar reads "student@pesu-OptiPlex-3070: ~/Desktop/week5". The terminal content shows the execution of a Python script named "UDPClient.py". The prompt is "student@pesu-OptiPlex-3070:~/Desktop/week5\$". The command "python UDPClient.py" is entered. The program then prompts "Input lowercase sentence:pesu". The user enters "PESU". The prompt returns to "student@pesu-OptiPlex-3070:~/Desktop/week5\$".

```
student@pesu-OptiPlex-3070:~/Desktop/week5$ python UDPClient.py
Input lowercase sentence:pesu
PESU
student@pesu-OptiPlex-3070:~/Desktop/week5$
```

QUESTIONS RELATED TO TASK 1 :

QUESTION 1 : Suppose you run TCPClient before you run TCPServer. What happens? Why?

ANSWER 1: 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.

```
student@pesu-OptiPlex-3070:~/Desktop/week5$ python TCPClient.py
Traceback (most recent call last):
  File "TCPClient.py", line 5, in <module>
    clientSocket.connect((serverName,serverPort))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 111] Connection refused
student@pesu-OptiPlex-3070:~/Desktop/week5$
```

QUESTION 2: Suppose you run UDPClient before you run UDPServer. What happens? Why?

ANSWER 2: 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.

```
student@pesu-OptiPlex-3070:~/Desktop/week5$ python UDPClient.py
Input lowercase sentence:priya sharmila
PRIYA SHARMILA
student@pesu-OptiPlex-3070:~/Desktop/week5$
```

QUESTION 3: What happens if you use different port numbers for the client and server sides?

ANSWER 3: 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.

TASK 2:WEB SERVER

Web server will

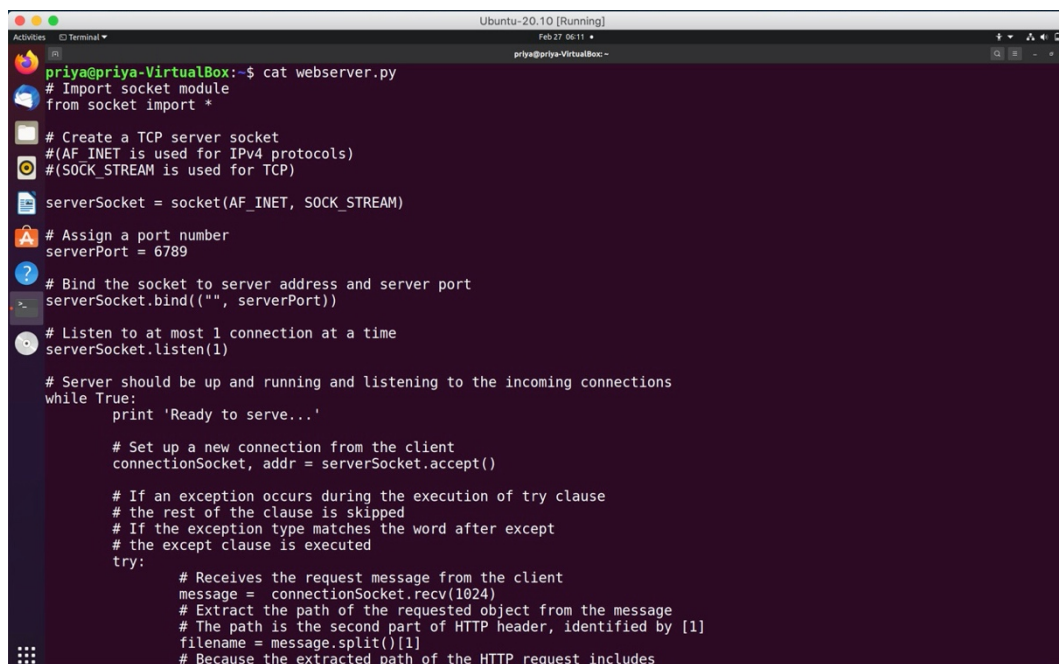
- a) create a connection socket when contacted by a client (browser);
- b) receive the HTTP request from this connection;
- c) parse the request to determine the specific file being requested;
- d) get the requested file from the server's file system;
- e) create an HTTP response message consisting of the requested file preceded by header

lines; and

- 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.

WEBSERVER.py



```

priya@priya-VirtualBox:~$ cat webservice.py
# Import socket module
from socket import *

# Create a TCP server socket
#(AF_INET is used for IPv4 protocols)
#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

# Assign a port number
serverPort = 6789

# Bind the socket to server address and server port
serverSocket.bind(('', serverPort))

# Listen to at most 1 connection at a time
serverSocket.listen(1)

# Server should be up and running and listening to the incoming connections
while True:
    print 'Ready to serve...'

    # Set up a new connection from the client
    connectionSocket, addr = serverSocket.accept()

    # If an exception occurs during the execution of try clause
    # the rest of the clause is skipped
    # If the exception type matches the word after except
    # the except clause is executed
    try:
        # Receives the request message from the client
        message = connectionSocket.recv(1024)
        # Extract the path of the requested object from the message
        # The path is the second part of HTTP header, identified by [1]
        filename = message.split()[1]
        # Because the extracted path of the HTTP request includes

```

```

# If an exception occurs during the execution of try clause
# the rest of the clause is skipped
# If the exception type matches the word after except
# the except clause is executed
try:
    # Receives the request message from the client
    message = connectionSocket.recv(1024)
    # Extract the path of the requested object from the message
    # The path is the second part of HTTP header, identified by [1]
    filename = message.split()[1]
    # Because the extracted path of the HTTP request includes
    # a character '\', we read the path from the second character
    f = open(filename[1:])
    # Store the entire content of the requested file in a temporary buffer
    outputdata = f.read()
    # Send the HTTP response header line to the connection socket
    connectionSocket.send("HTTP/1.1 200 OK\r\n\r\n")

    # Send the content of the requested file to the connection socket
    for i in range(0, len(outputdata)):
        connectionSocket.send(outputdata[i])
        connectionSocket.send("\r\n")

    # Close the client connection socket
    connectionSocket.close()

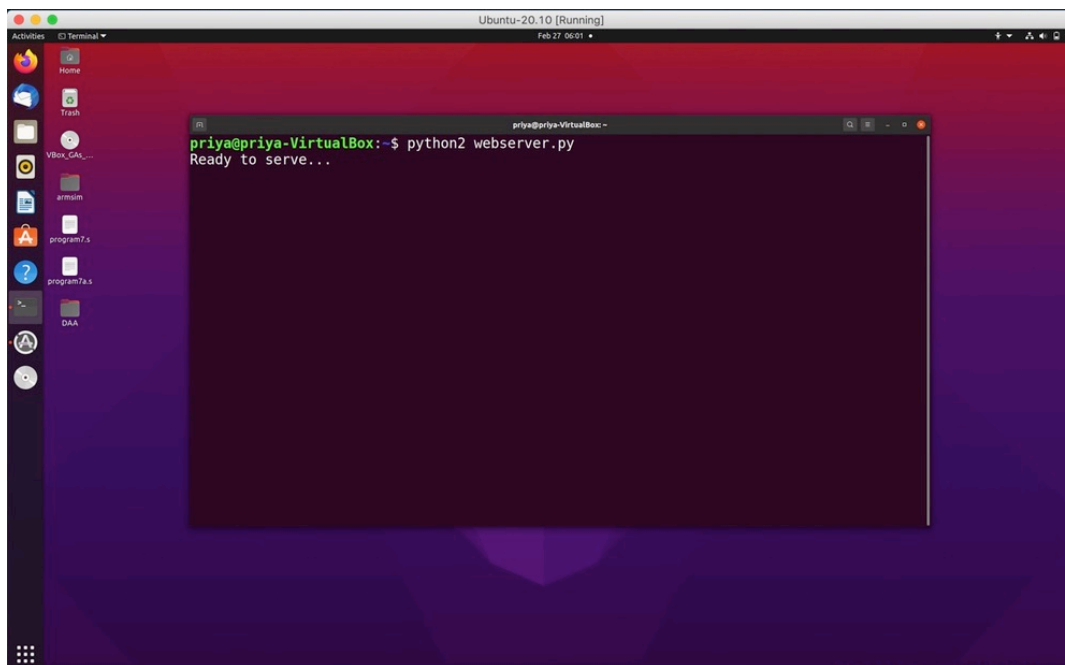
except IOError:
    # Send HTTP response message for file not found
    connectionSocket.send("HTTP/1.1 404 Not Found\r\n\r\n")
    connectionSocket.send("<html><head></head><body><h1>404 Not Found</h1></body></html>\r\n")
    # Close the client connection socket
    connectionSocket.close()

serverSocket.close()

priya@priya-VirtualBox:~$

```

Running the webserver



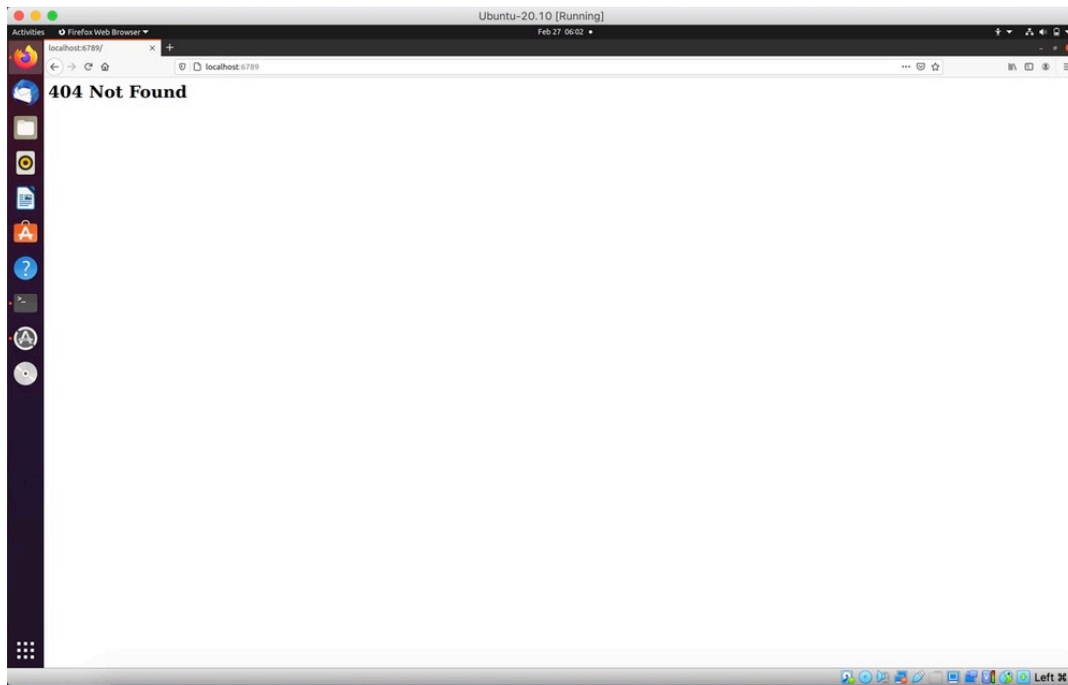
```

priya@priya-VirtualBox:~$ python2 webserver.py
Ready to serve...

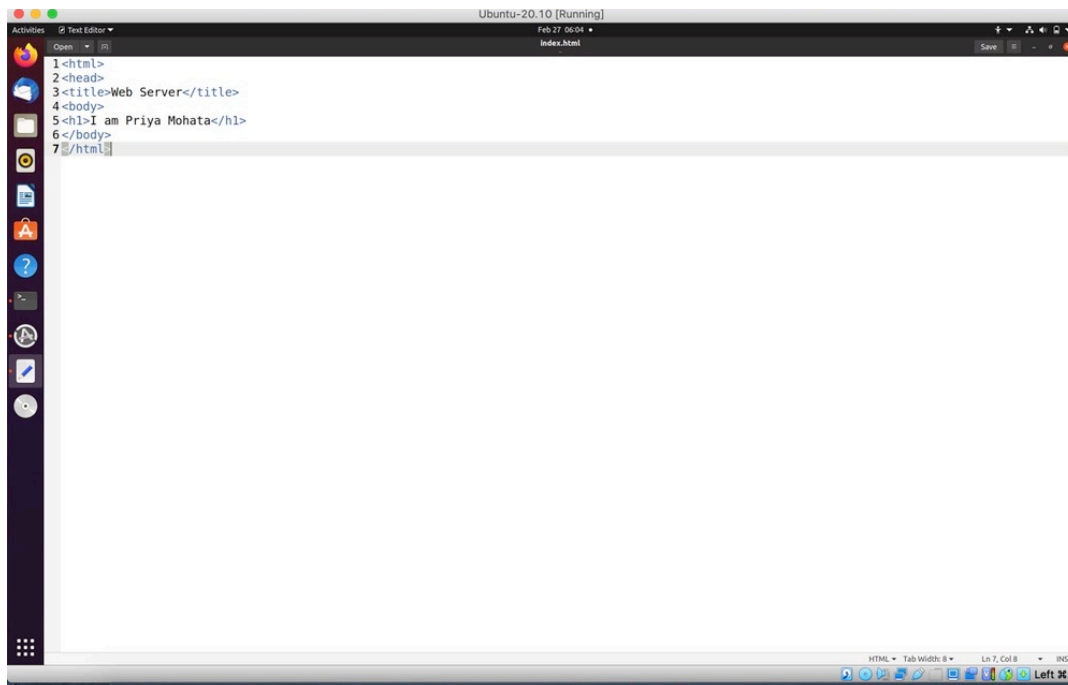
```

Accessing a file not present on the server

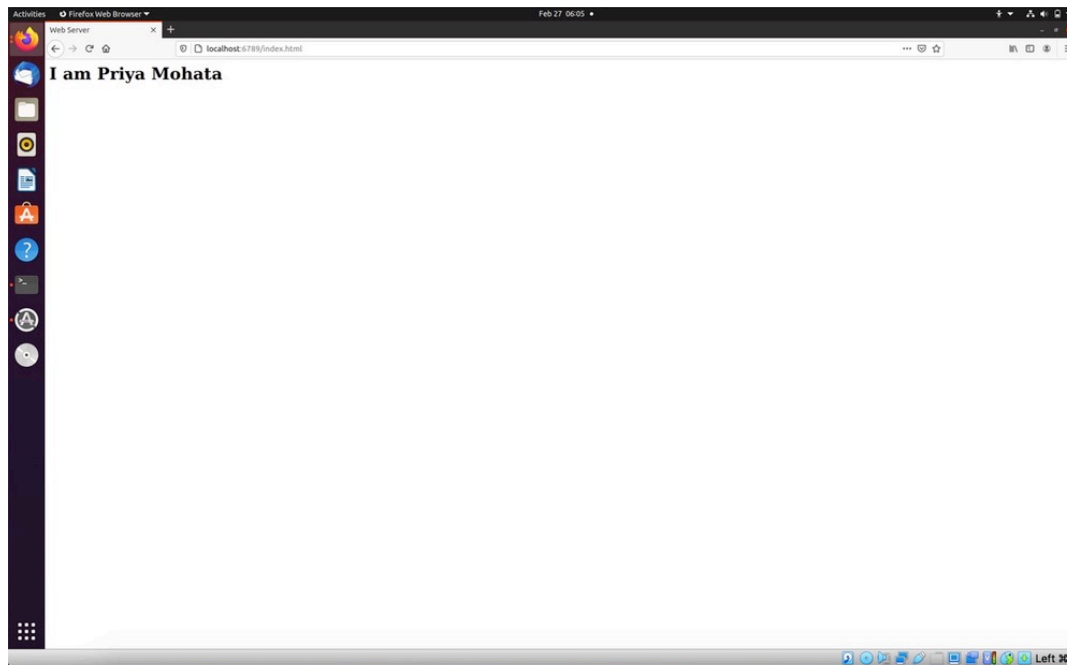
404 Not Found error is obtained



Creating index.html that will now be accessed



Accessing localhost:6789/index.html



-----*Thank you*-----