Computer Networks Laboratory – Week5

Name: OP JOY JEFFERSON

Section: E

SRN: PES2UG19CS270

DATE: 27/02/2021

AIM: To develop a simple Client-Server application using TCP and UDP.

1. Sockets

Tasks:

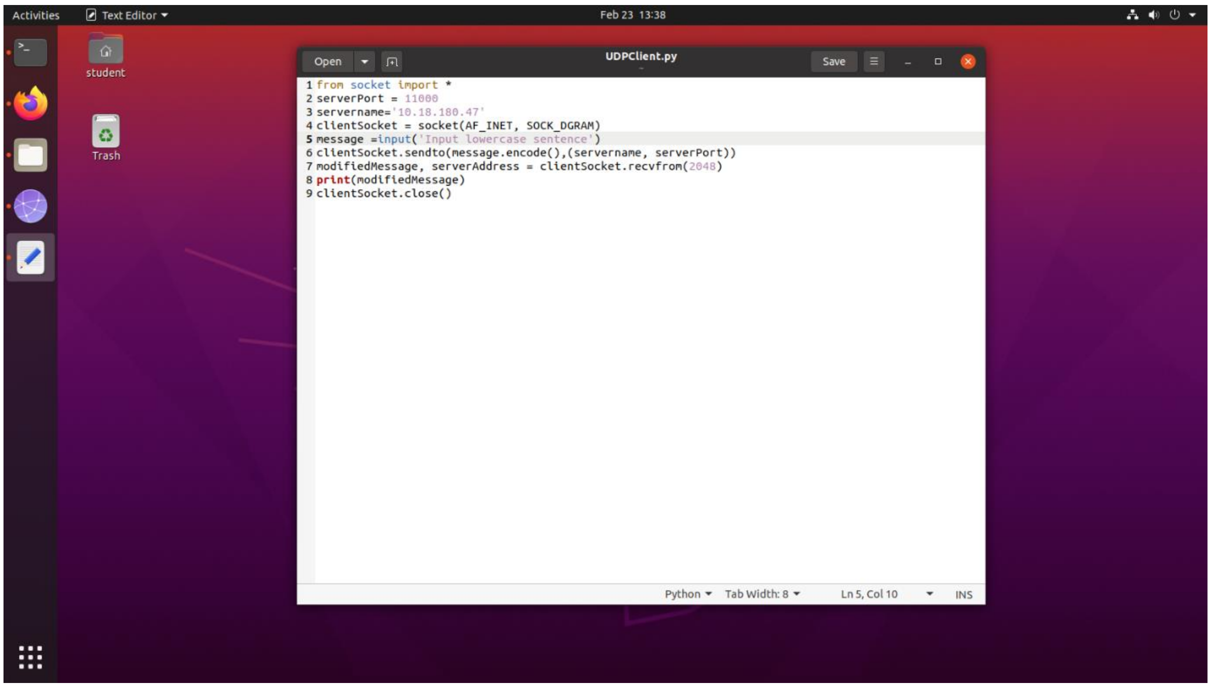
- Create an application that will
  a).Convert lowercase letters to uppercase
    •e.g. [a...z] to [A...Z]
    •code will not change any special characters, e.g. &*!
  b).If the character is in uppercase, the program must not alter
- Create Socket API both for client and server.
- Must take the server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

1.1 Socket programming with UDP

- A simple client-server application using UDP can be setup with python3 and python's socket module.

1.1.1 Implementing client

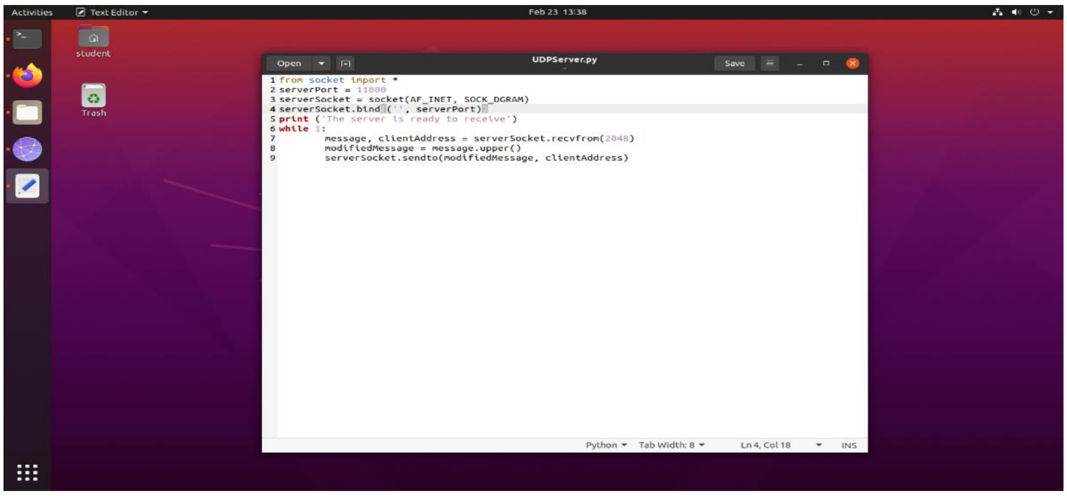
- To begin with we will need to import the socket python module.
- Once we've got this, we need to declare the IPv4 address that we will be trying to send our UDP messages to as well as the port number. This port number is arbitrary but ensure that you aren't using a socket that has already been taken.
- Now that we've declared these few variables it's time to create the socket through which we will be sending our UDP message to the server.
- Finally, once we've constructed our new socket it's time to write the code that will send our UDP message.



```python
from socket import *
serverPort = 11000
servername='10.18.180.47'
clientSocket = socket(AF_INET, SOCK_DGRAM)
message =input('Input lowercase sentence')
clientSocket.sendto(message.encode(),(servername, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage)
clientSocket.close()
```

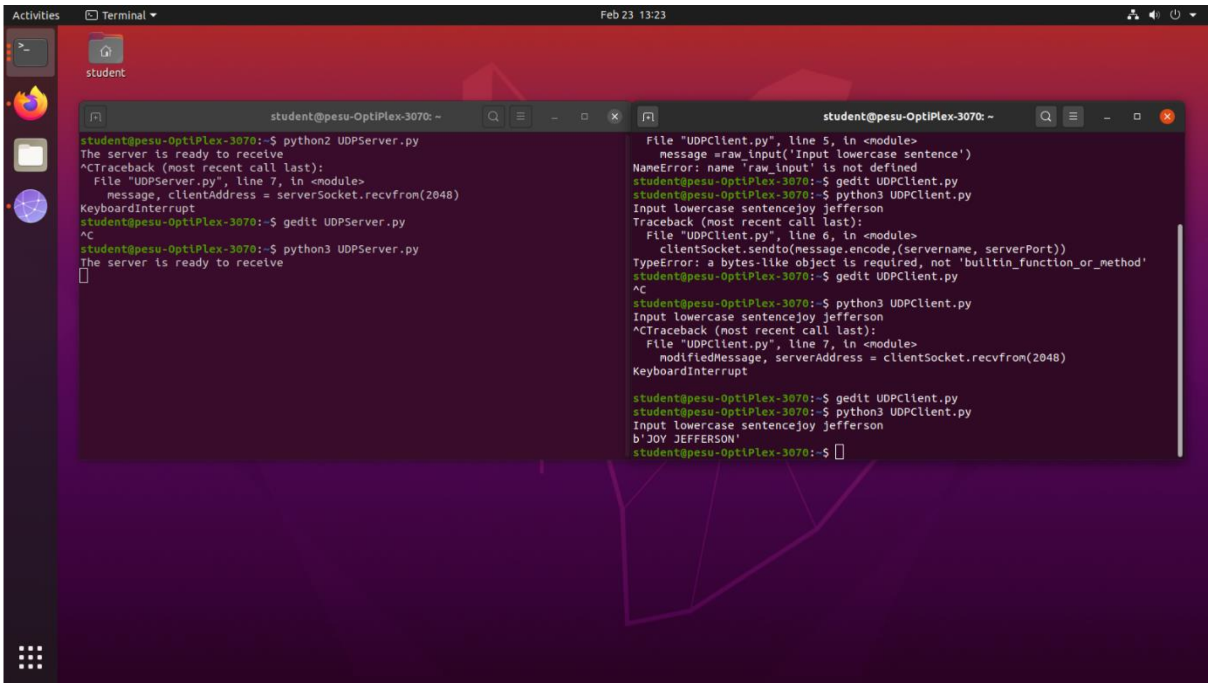
1.1.2 Implementing server:

- Now that we've coded our client, we then need to move on to creating our server program which will be continuously listening on our defined IPv4 address and port number for any UDP messages. It is essential that this server has to be run prior to the execution of the client python script or the client script will fail.
- Once we've imported the socket module and declared our ip address and port number we can create another socket which will look exactly like the socket we constructed in our client program.
- Finally, once we've created our server socket, we need to write the code that will keep our script continuously listening to this socket until its termination. This takes form as a simple while loop, like so



```python
from socket import *
serverPort = 11000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ('The server is ready to receive')
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

1.1.3 After establishing the connection



```
student@pesu-OptiPlex-3070:~$ python2 UDPServer.py
The server is ready to receive
^CTraceback (most recent call last):
  File "UDPServer.py", line 7, in <module>
    message, clientAddress = serverSocket.recvfrom(2048)
KeyboardInterrupt
student@pesu-OptiPlex-3070:~$ gedit UDPServer.py
^C
student@pesu-OptiPlex-3070:~$ python3 UDPServer.py
The server is ready to receive
```

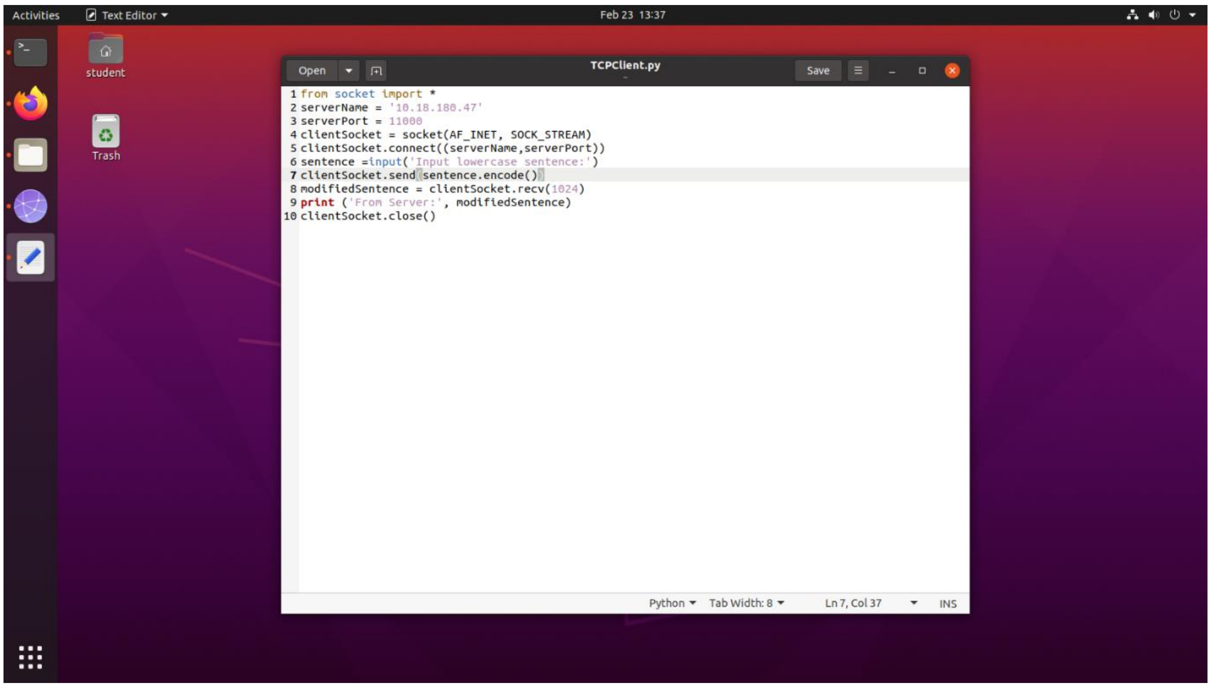
```
  File "UDPClient.py", line 5, in <module>
    message =raw_input('Input lowercase sentence')
NameError: name 'raw_input' is not defined
student@pesu-OptiPlex-3070:~$ gedit UDPClient.py
student@pesu-OptiPlex-3070:~$ python3 UDPClient.py
Input lowercase sentencejoy jefferson
Traceback (most recent call last):
  File "UDPClient.py", line 6, in <module>
    clientSocket.sendto(message.encode,(servername, serverPort))
TypeError: a bytes-like object is required, not 'builtin_function_or_method'
student@pesu-OptiPlex-3070:~$ gedit UDPClient.py
^C
student@pesu-OptiPlex-3070:~$ python3 UDPClient.py
Input lowercase sentencejoy jefferson
^CTraceback (most recent call last):
  File "UDPClient.py", line 7, in <module>
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
KeyboardInterrupt

student@pesu-OptiPlex-3070:~$ gedit UDPClient.py
student@pesu-OptiPlex-3070:~$ python3 UDPClient.py
Input lowercase sentencejoy jefferson
b'JOY JEFFERSON'
student@pesu-OptiPlex-3070:~$
```

1.2 Socket programming with TCP

- A simple client-server application using TCP can be setup with python3 and python's socket module.

1.2.1 Implementing client

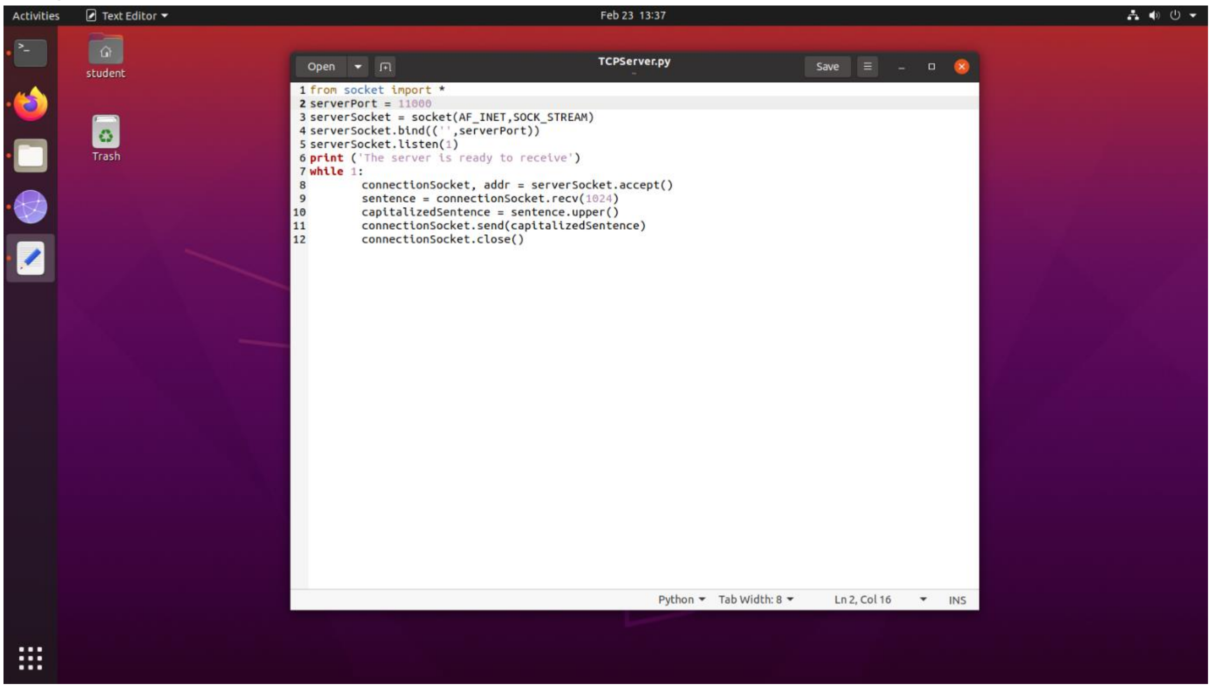
- To begin with we will need to import the socket python module.
- The client program sets up its socket differently from the way a server does. Instead of binding to a port and listening, it uses connect() to attach the socket directly to the remote address
- After the connection is established, data can be sent through the socket with send() and received with recv(), just as in the server.
- When the entire message is sent and a copy received, the socket is closed to free up the port.



```python
from socket import *
serverName = '10.18.180.47'
serverPort = 11000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence =input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence)
clientSocket.close()
```

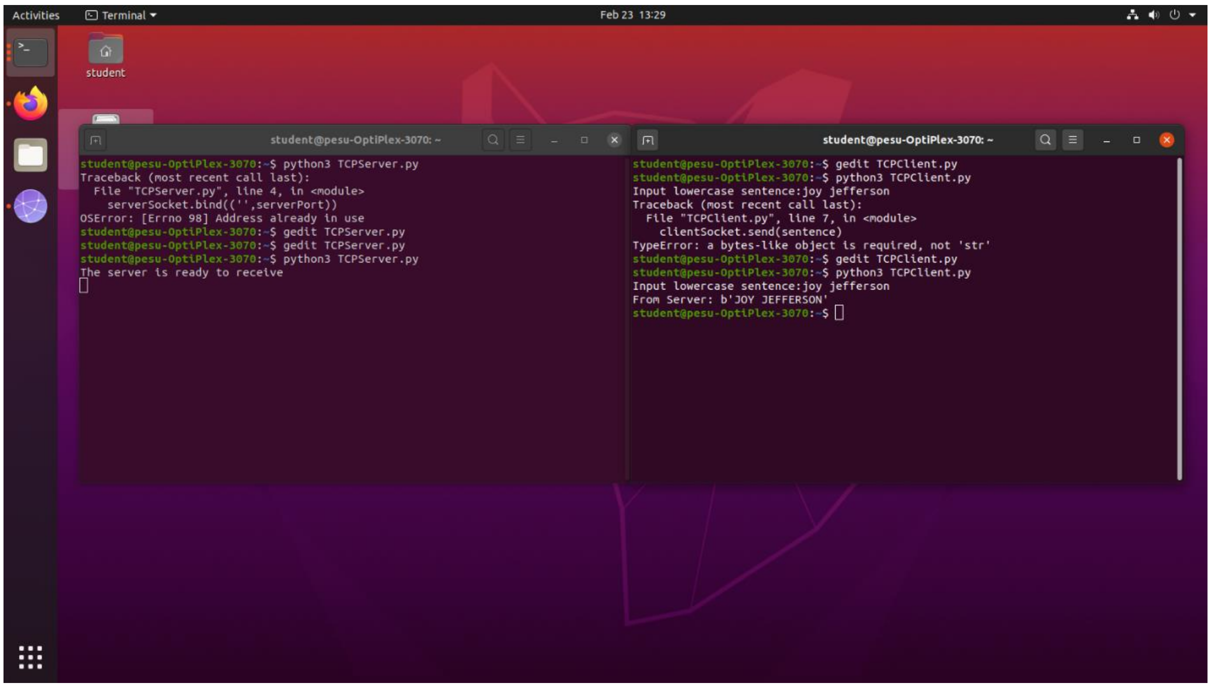
1.2.2 Implementing server

- Then bind () is used to associate the socket with the server address. In this case, the address is localhost, referring to the current server, and the port number is 11000.
- Calling listen () puts the socket into server mode, and accept () waits for an incoming connection.
- When communication with a client is finished, the connection needs to be cleaned up using close () .



```python
from socket import *
serverPort = 11000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print ('The server is ready to receive')
while 1:
        connectionSocket, addr = serverSocket.accept()
        sentence = connectionSocket.recv(1024)
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
        connectionSocket.close()
```

1.2.3 After establishing the connection



```
student@pesu-OptiPlex-3070:~$ python3 TCPServer.py
Traceback (most recent call last):
  File "TCPServer.py", line 4, in <module>
    serverSocket.bind(('',serverPort))
OSError: [Errno 98] Address already in use
student@pesu-OptiPlex-3070:~$ gedit TCPServer.py
student@pesu-OptiPlex-3070:~$ gedit TCPServer.py
student@pesu-OptiPlex-3070:~$ python3 TCPServer.py
The server is ready to receive
```

```
student@pesu-OptiPlex-3070:~$ gedit TCPClient.py
student@pesu-OptiPlex-3070:~$ python3 TCPClient.py
Input lowercase sentence:joy jefferson
Traceback (most recent call last):
  File "TCPClient.py", line 7, in <module>
    clientSocket.send(sentence)
TypeError: a bytes-like object is required, not 'str'
student@pesu-OptiPlex-3070:~$ gedit TCPClient.py
student@pesu-OptiPlex-3070:~$ python3 TCPClient.py
Input lowercase sentence:joy jefferson
From Server: b'JOY JEFFERSON'
student@pesu-OptiPlex-3070:~$
```

Problems:

1.Suppose you run TCPClient before you run TCPServer. What happens? Why?
A1).When TCPClient is run before TCPServer ,it throws **ConnectionRefusedError**.If you run TCPClient first, then the client will attempt to make a TCP connection with a non-existent server process. A TCP connection will not be made.

2.Suppose you run UDPClient before you run UDPServer. What happens? Why?
A2).UDPClient doesn't establish a TCP connection with the server. Thus, everything should work fine if you first run UDPClient, then run UDPServer. Therefore, it only leads to loss of packets if client is executed first.
3.What happens if you use different port numbers for the client and server sides?
A3).If you use different port numbers, then the client will attempt to establish a TCP connection with the wrong process or a non-existent process and therefore it will throw a **ConnectionRefusedError** again.

For UDP since no prior connection is required, no such error will be thrown but all packets transferred meanwhile will be lost.

Task2:

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

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;

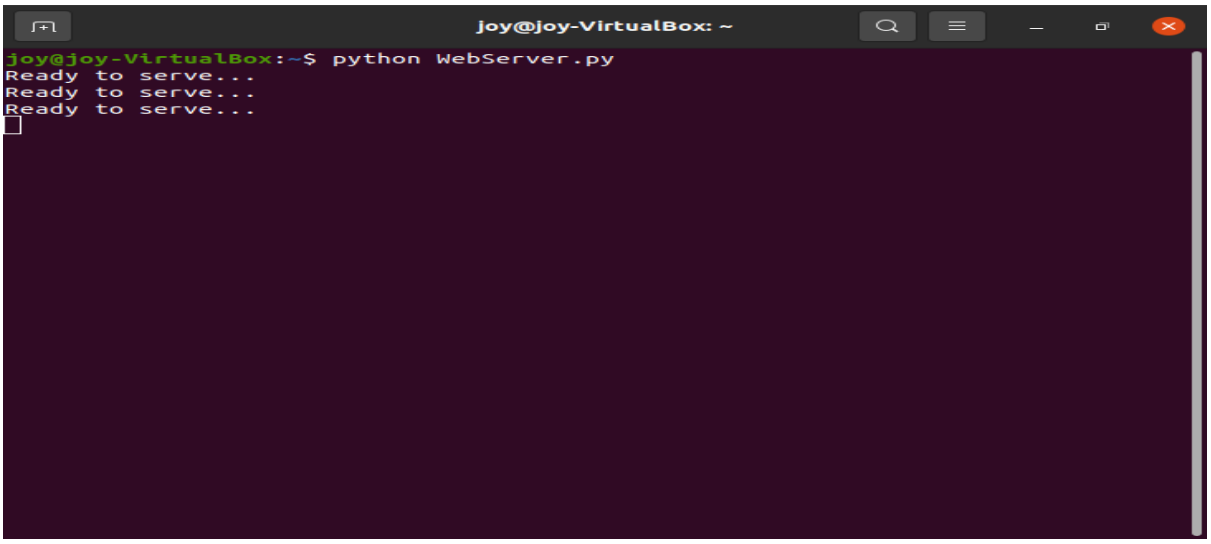
d) get the requested file from the server's file system;

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.

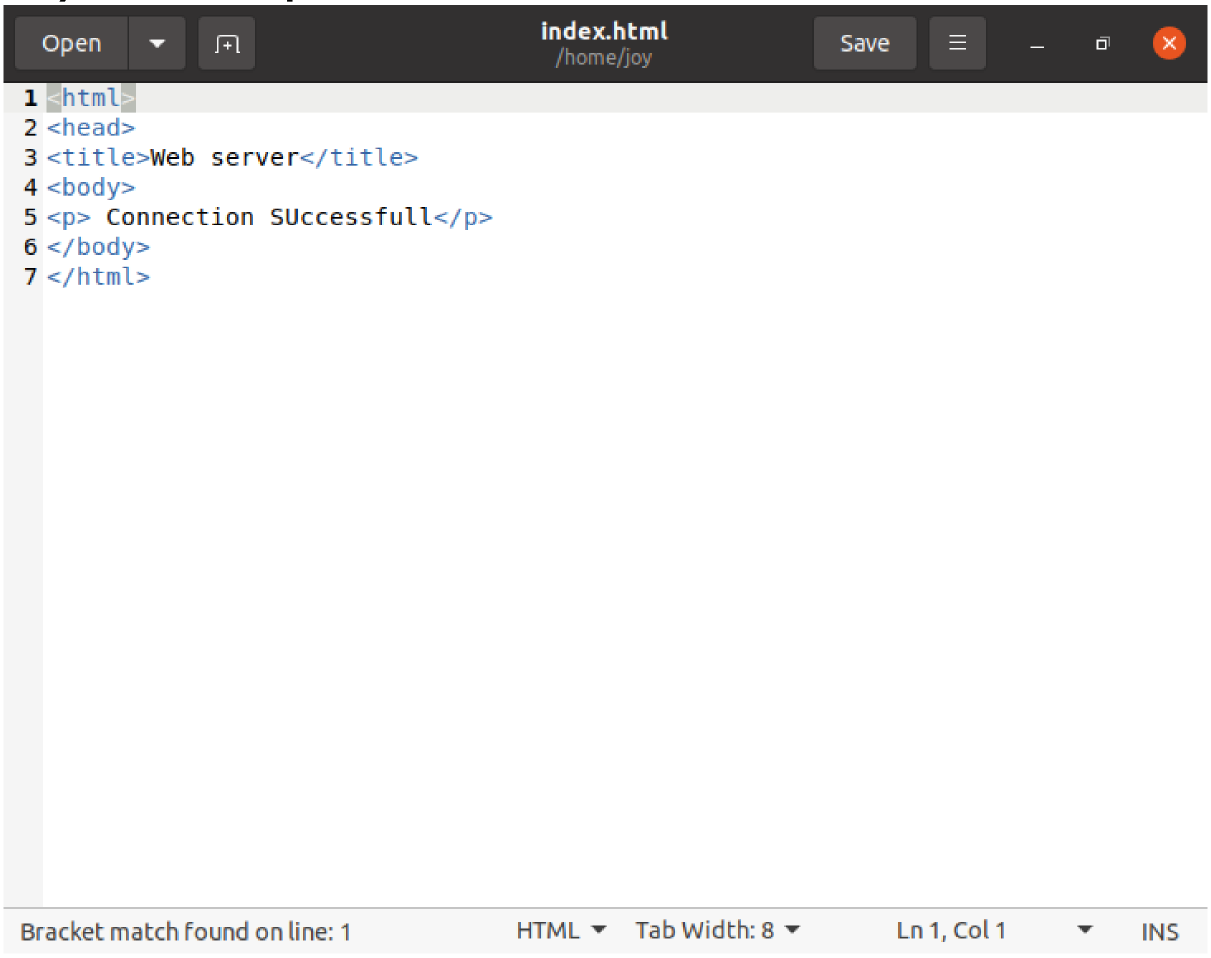
2.1 The server is setup by running,

$python WebServer.py



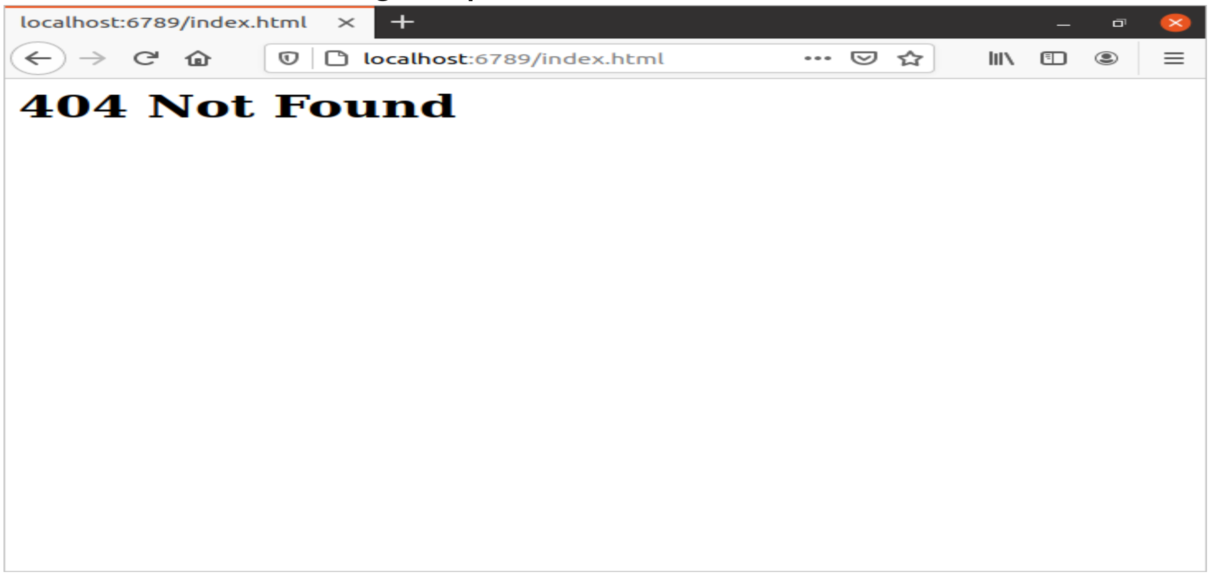
```
joy@joy-VirtualBox:~$ python WebServer.py
Ready to serve...
Ready to serve...
Ready to serve...
```

2.2).ServerScript



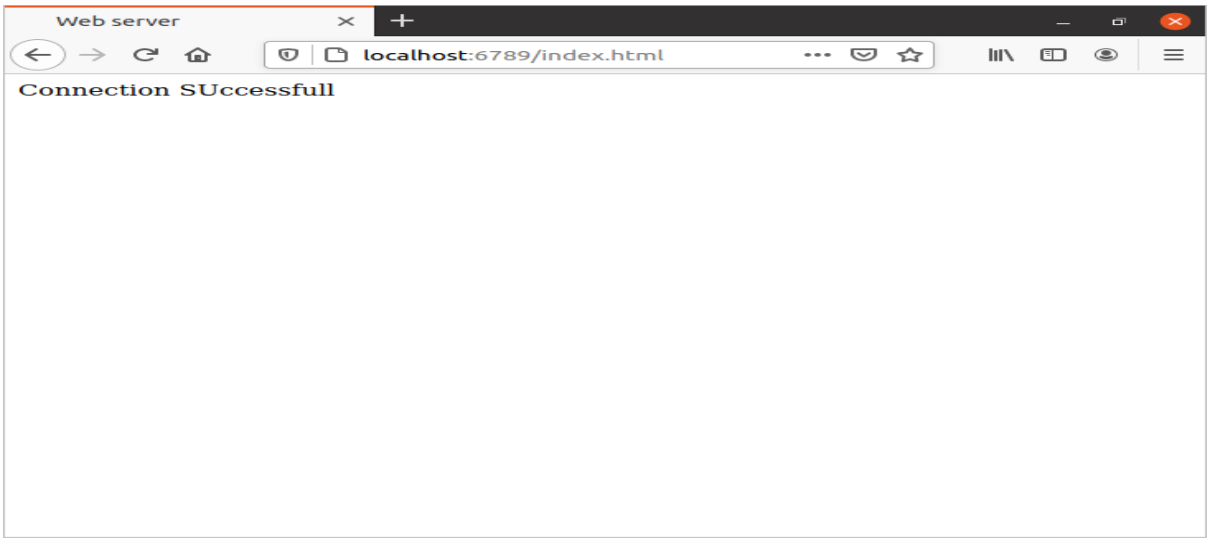
```html
<html>
<head>
<title>Web server</title>
<body>
<p> Connection SUccessfull</p>
</body>
</html>
```

2.3). When the browser tries to access a file which isn't created,the following response is obtained.



404 Not Found

2.4). After setting up the index.html file, and then requesting again, we get the following response.



```
Connection SUccessfull
```