

CN Lab Report – Week 5

NAME	SRN	SECTION
ANKITH J RAI	PES1UG19CS069	B

TASK 1

Client program for datagram service (UDP)

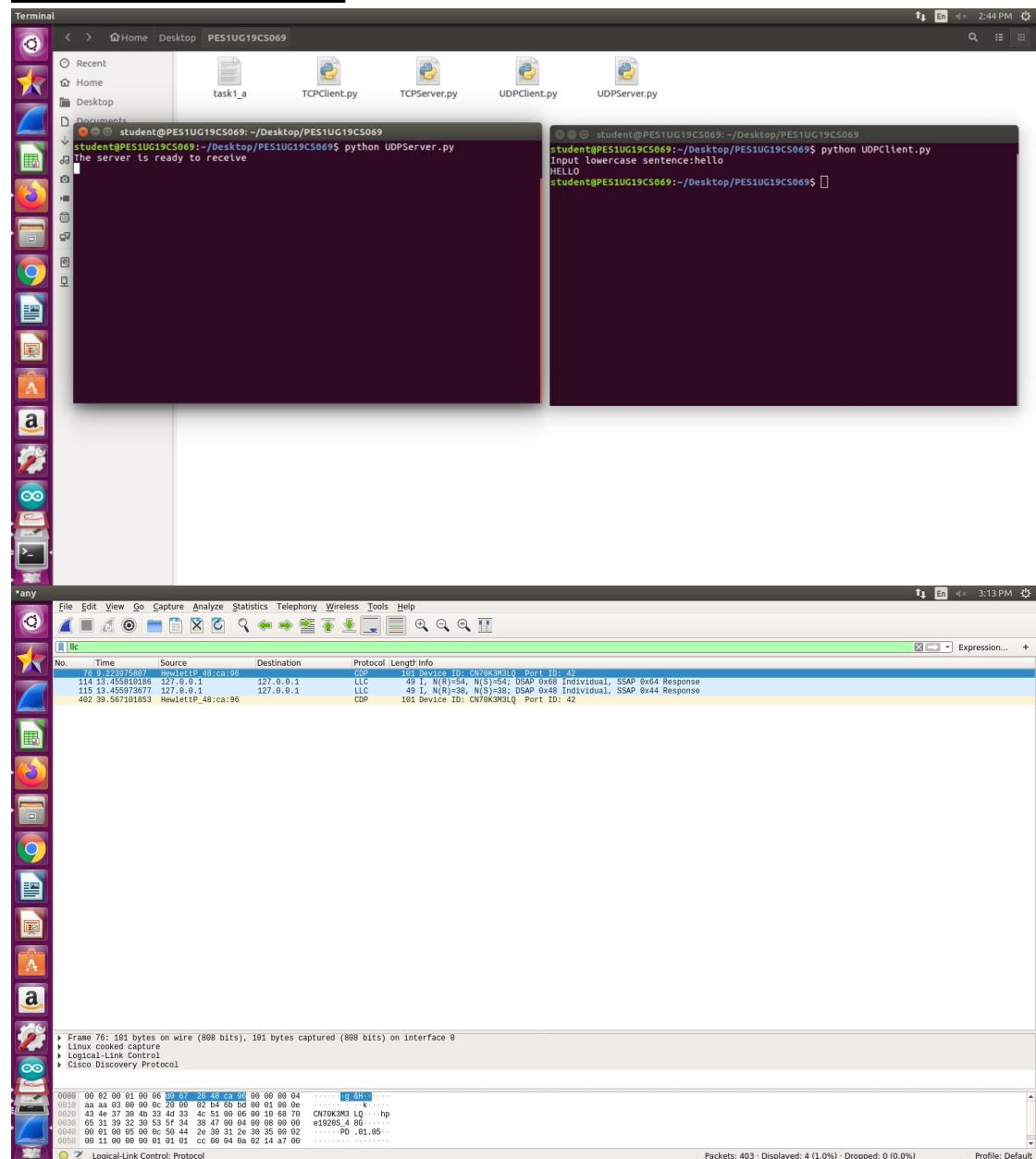
```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
Print(modifiedMessage)
clientSocket.close()
```

Server program for datagram service (UDP)

```
from socket import *
serverPort = 12000
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)
```

UDP - wireshark and terminal

screenshots:



Client program for reliable byte-stream service (TCP)

```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print('From Server:', modifiedSentence)
clientSocket.close()
```

Server program for reliable byte-stream service (TCP)

```
from socket import *
serverPort = 12000
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()
```

TCP - wireshark and terminal screenshots:

```

connectionSocket.close()

student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python TCPclient.py
Input lowercase sentence:Hello
(From Server: "HELLO")
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ 

student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python TCPserver.py
The server is ready to receive

```

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

- create a connection socket when contacted by a client (browser);
- receive the HTTP request from this connection;
- parse the request to determine the specific file being requested;

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	12800 - 35208 [SYN] Seq=0 Win=65536 Len=0 MSS=1460 SACK_PERM=1 TSval=2614662603 Tscr=2614662603 WS=128
2	0.000008889	127.0.0.1	127.0.0.1	TCP	66	35208 - 12800 [SYN, ACK] Seq=1 Win=65536 Len=0 TSval=2614662603 Tscr=2614662603
3	0.000016265	127.0.0.1	127.0.0.1	TCP	71	12800 - 35208 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2614662603 Tscr=2614662603
4	0.000023741	127.0.0.1	127.0.0.1	TCP	66	35208 - 12800 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2614662603 Tscr=2614662603
5	0.227274712	127.0.0.1	127.0.0.1	TCP	71	12800 - 35208 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2614664831 Tscr=2614664830
6	0.227366067	127.0.0.1	127.0.0.1	TCP	71	12800 - 35208 [PSH, ACK] Seq=1 Ack=6 Win=65536 Len=5 TSval=2614664831 Tscr=2614664830
7	0.227378232	127.0.0.1	127.0.0.1	TCP	66	35208 - 12800 [ACK] Seq=6 Ack=6 Win=65536 Len=0 TSval=2614664831 Tscr=2614664831
8	0.227385263	127.0.0.1	127.0.0.1	TCP	66	35208 - 12800 [ACK] Seq=6 Ack=6 Win=65536 Len=0 TSval=2614664831 Tscr=2614664831
9	0.227595535	127.0.0.1	127.0.0.1	TCP	66	35208 - 12800 [ACK] Seq=6 Ack=6 Win=65536 Len=0 TSval=2614664831 Tscr=2614664831
10	0.227576519	127.0.0.1	127.0.0.1	TCP	66	12800 - 35208 [ACK] Seq=7 Ack=7 Win=65536 Len=0 TSval=2614664831 Tscr=2614664831

Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ Transmission Control Protocol, Src Port: 35208, Dst Port: 12800, Seq: 0, Len: 0

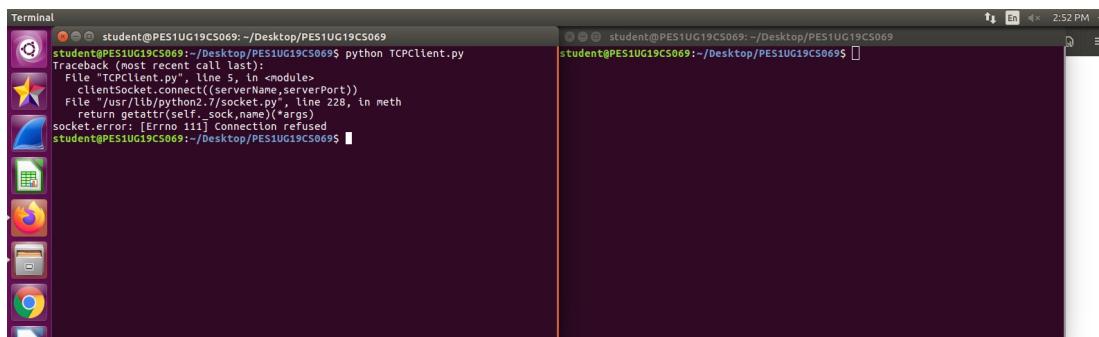
0019 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00 <- 0 0 Z.....E
 0019 00 3c e2 1d 40 09 49 00 5a 9c 7f 09 09 01 07 00 ..0.....
 0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a8 02 ..0.....
 0030 ff d7 fe 30 00 00 02 04 ff d7 94 02 68 8a 9b 08 ..0.....
 0049 95 cb 00 00 00 00 01 03 03 07 ..0.....

Loopback: lo: <live capture in progress> Packets: 10 · Displayed: 10 (100.0%) Profile: Default

PROBLEMS:

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

Ans) Suppose you run the TCPClient before running the TCPServer then the error is obtained is
socket error:connection refused



```
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ 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@PES1UG19CS069:~/Desktop/PES1UG19CS069$
```

Any one of the below tasks is mandatory for Week-5 completion.

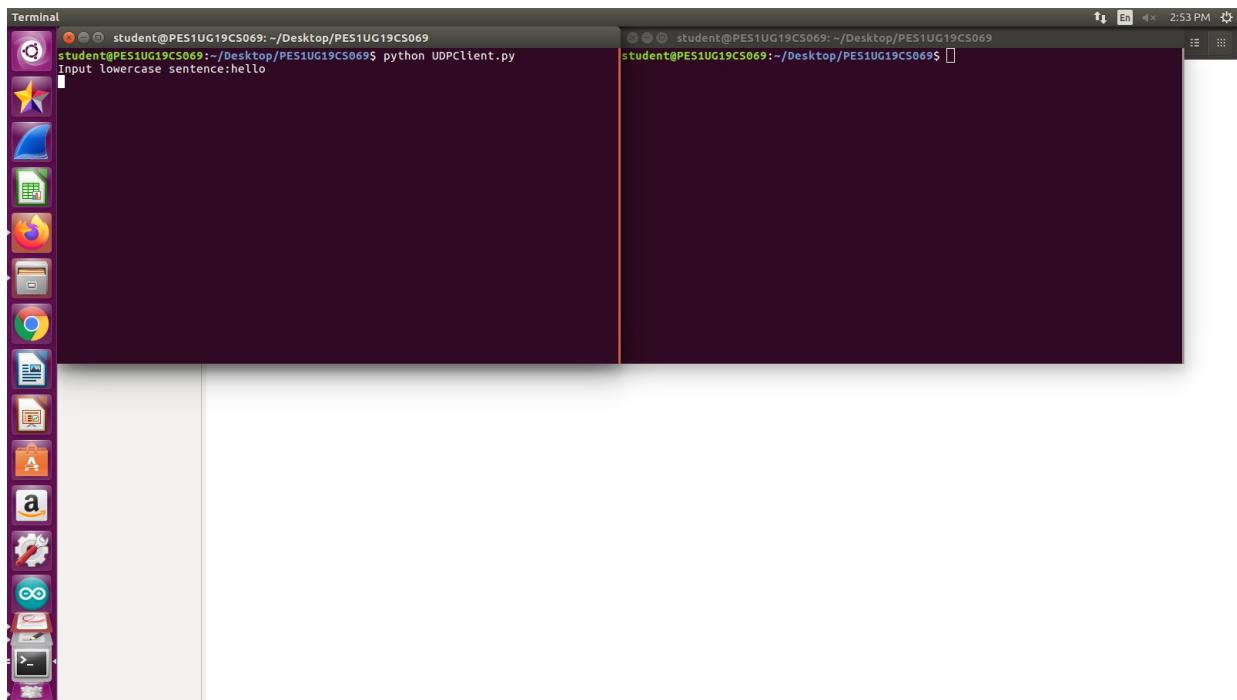
Task 2: Web Server

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

- create a connection socket when contacted by a client (browser);
- receive the HTTP request from this connection;
- parse the request to determine the specific file being requested;

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

Ans) Suppose you run the UDPClient before UDPServer then the error obtained is that the packets are lost hence the UDPClient running terminal does not show any output.



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

Ans)The following screenshots indicate the error obtained

a) For UDP Case

UDPClient

A screenshot of a Gedit text editor window. The title bar says "UDPClient.py (~/Desktop/PES1UG19CS069) - gedit". The code in the editor is as follows:

```
1 from socket import *
2 serverName = '127.0.0.1'
3 serverPort = 8080
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()
```

The status bar at the bottom shows "Python" and "Tab Width: 8".

UDPServer

```
UDPServer.py (-/Desktop/PES1UG19CS069) - gedit
Open  UDPClient.py
Save
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)

Python  Tab Width: 8  Ln 9, Col 9  INS
```

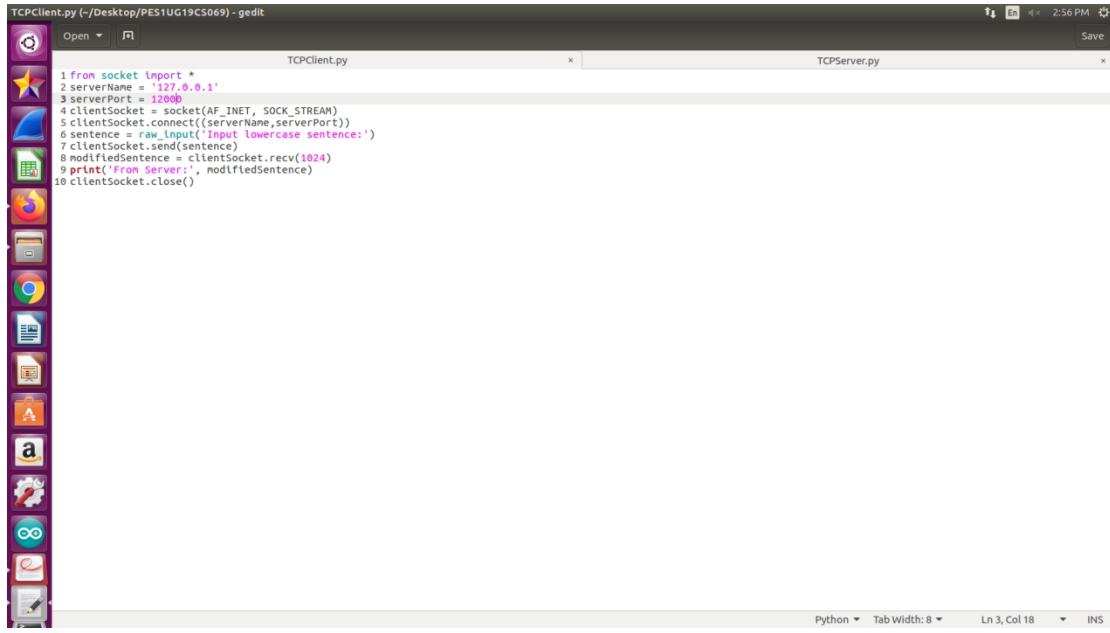
Terminal(screenshot)

```
Terminal
student@PES1UG19CS069:~/Desktop/PES1UG19CS069
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python UDPServer.py
The server is ready to receive
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python UDPClient.py
Input lowercase sentence:hello
```

The error obtained is that the packets are lost hence the UDPClient running terminal does not show any output.

b) For TCP Case

TCPClient

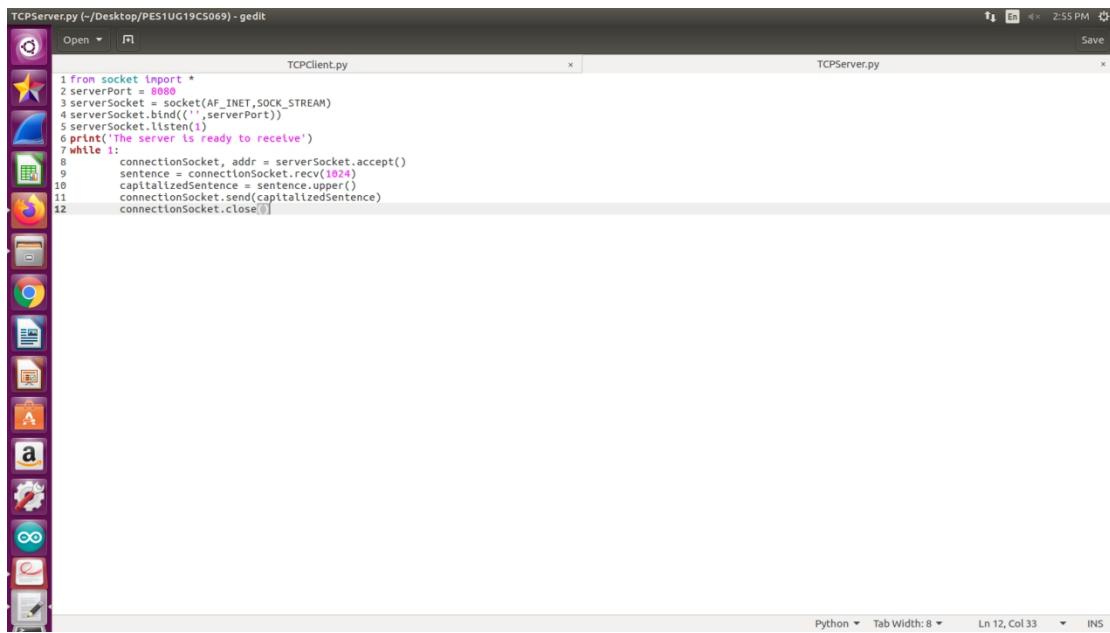


The screenshot shows a Linux desktop interface with a vertical application menu on the left. Two gedit windows are open: one titled "TCPClient.py" containing Python code for a client, and another titled "TCPServer.py" containing Python code for a server. The desktop background is light grey.

```
1 from socket import *
2 serverName = '127.0.0.1'
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 from socket import *
2 serverPort = 8080
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()
```

TCPServer



The screenshot shows a Linux desktop interface with a vertical application menu on the left. Two gedit windows are open: one titled "TCPClient.py" containing Python code for a client, and another titled "TCPServer.py" containing Python code for a server. The desktop background is light grey.

```
1 from socket import *
2 serverName = '127.0.0.1'
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 from socket import *
2 serverPort = 8080
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()
```

Terminal(screenshot)

The screenshot shows a dual-terminal setup on an Ubuntu desktop. The left terminal window displays the output of running `TCPServer.py`, which includes the message "The server is ready to receive". The right terminal window shows the output of running `TCPCClient.py`, which ends with a traceback and the error "socket.error: [Errno 111] Connection refused". A docked application menu on the left contains icons for various applications like Dash, Home, Dash to Dock, and others.

```
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python TCPserver.py
The server is ready to receive
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python TCPCClient.py
Traceback (most recent call last):
  File "TCPCClient.py", line 5, in <module>
    clientSocket.connect((serverName,serverPort))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getatime(self._sock,name)(*args)
socket.error: [Errno 111] Connection refused
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$
```

The error obtained is socket error:connection refused

TASK 2

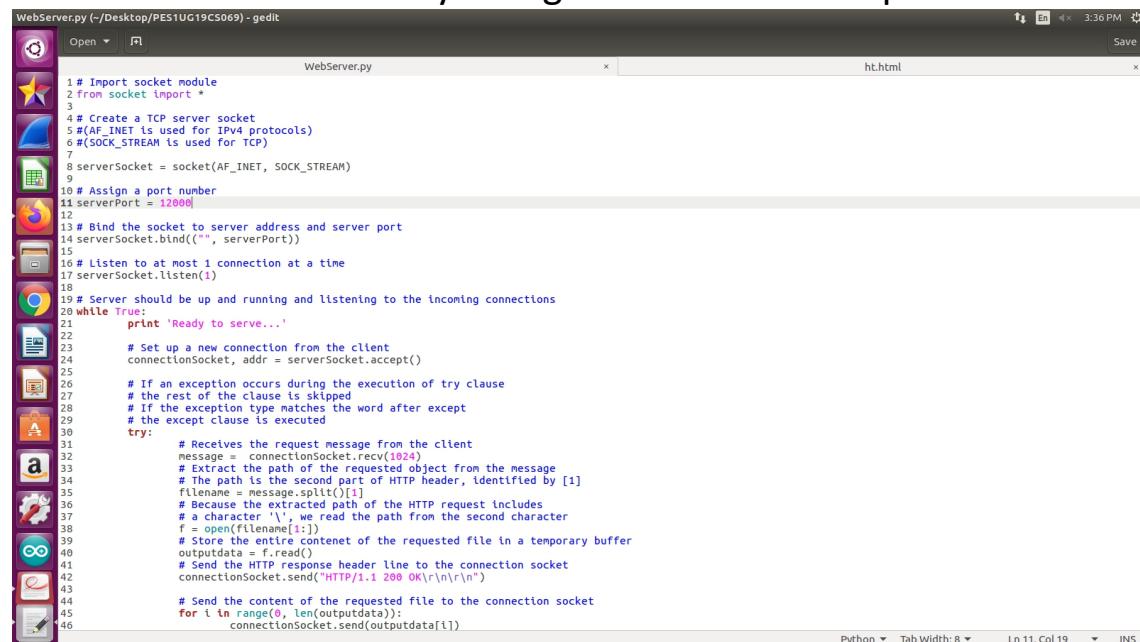
WEB SERVER

Test case 1(browser requesting a file that is present in the server):

Screenshot of code:

The html page I have taken is ht.html.

The ht.html contains only a single line code called pes.

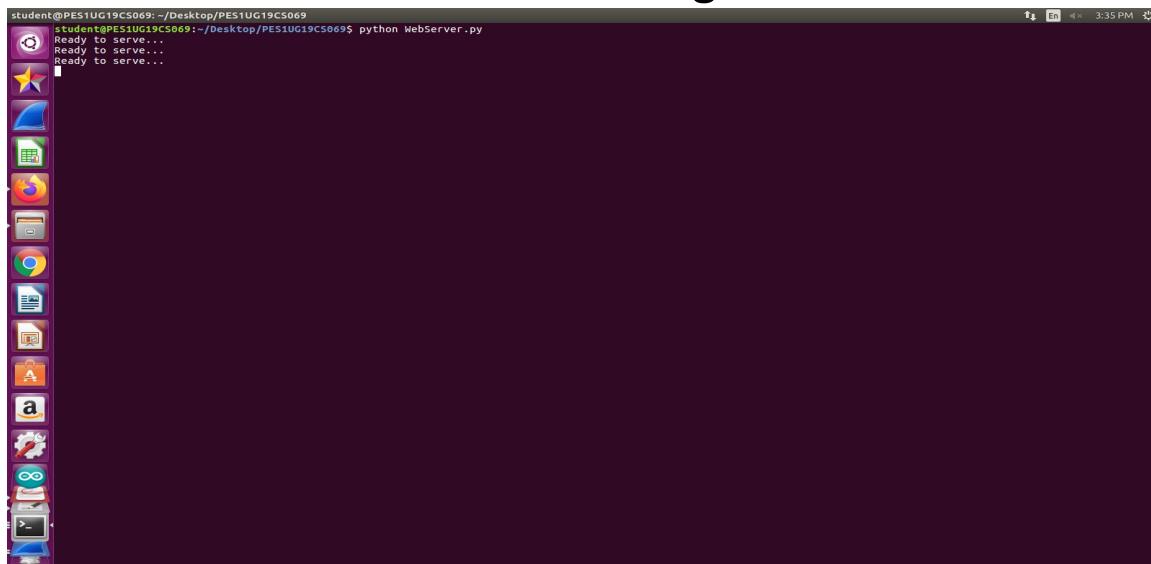


The screenshot shows a code editor with two tabs open. The left tab is named 'WebServer.py' and contains the following Python code:

```
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 = 12000
12
13 # Bind the socket to server address and server port
14 serverSocket.bind(("", 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
31     try:
32         # Receives the request message from the client
33         message = connectionSocket.recv(1024)
34         # Extract the path of the requested object from the message
35         # The path is the second part of HTTP header, identified by [1]
36         filename = message.split()[1]
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         # Strips the ethernet 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])
```

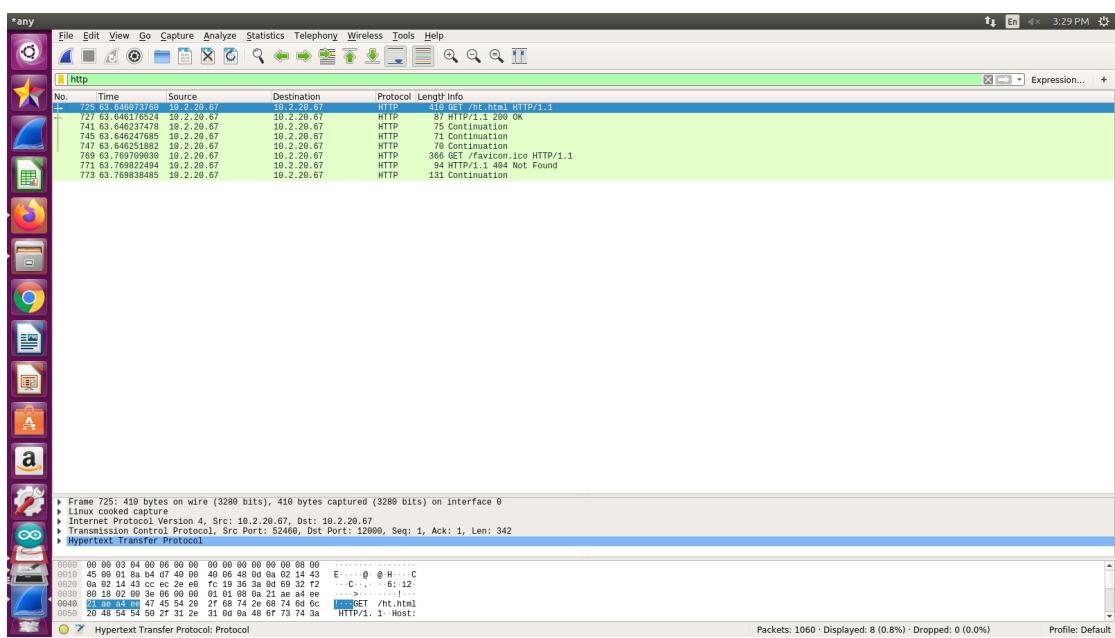
The right tab is named 'ht.html' and is currently empty.

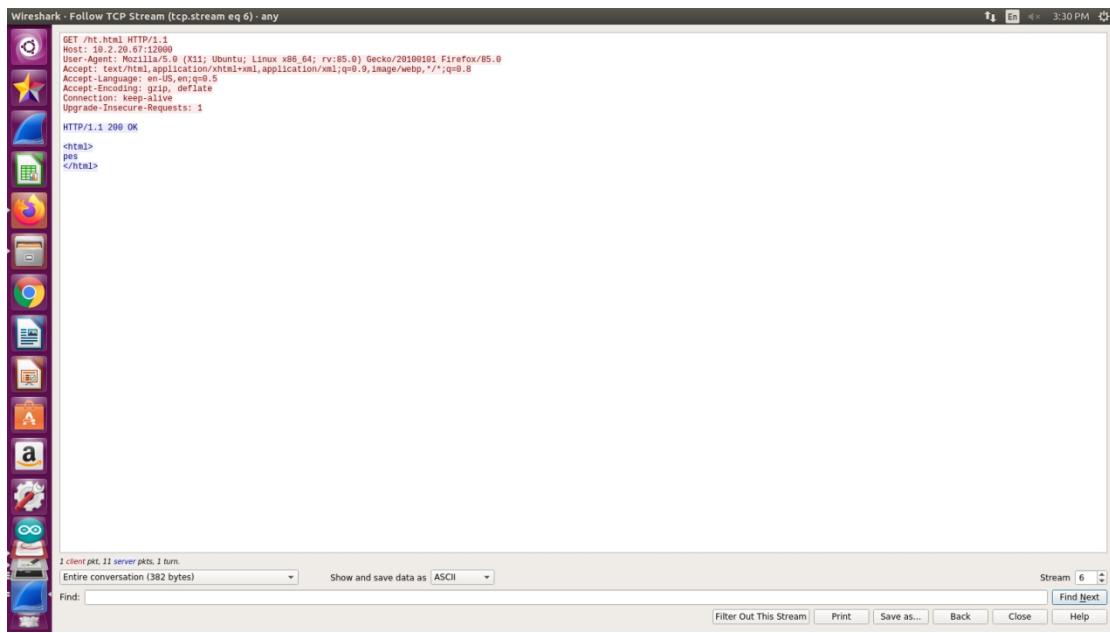
Screenshot of the server running:



```
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python WebServer.py
student@PES1UG19CS069:~/Desktop/PES1UG19CS069$ python WebServer.py
Ready to serve...
Ready to serve...
Ready to serve...
```

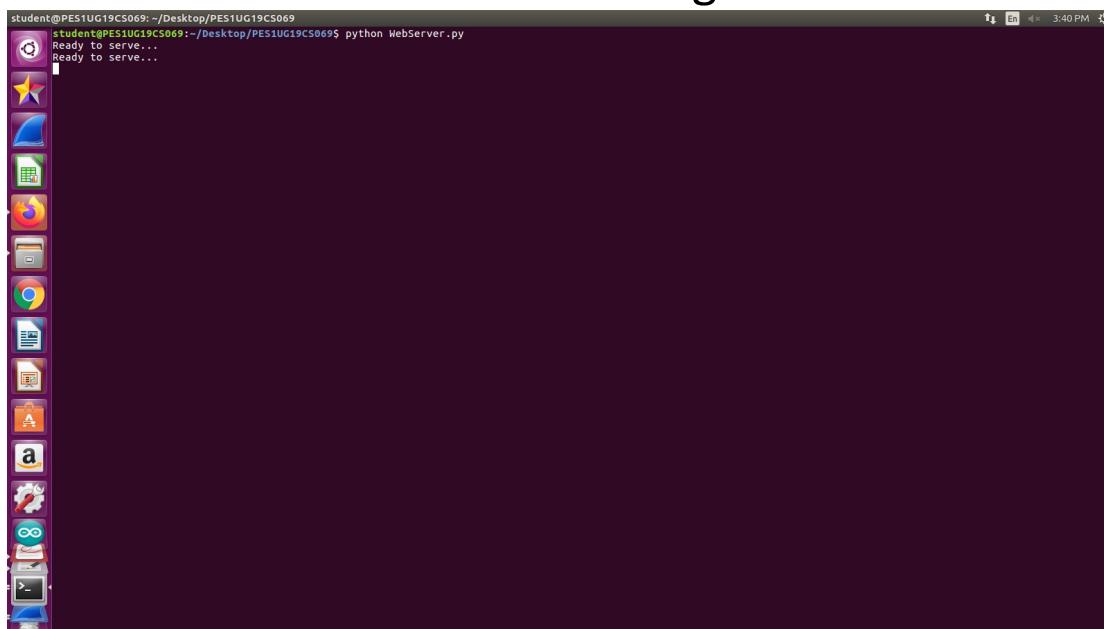
Wireshark screenshots:





Test case 2(browser requesting a file that is not present in the server):

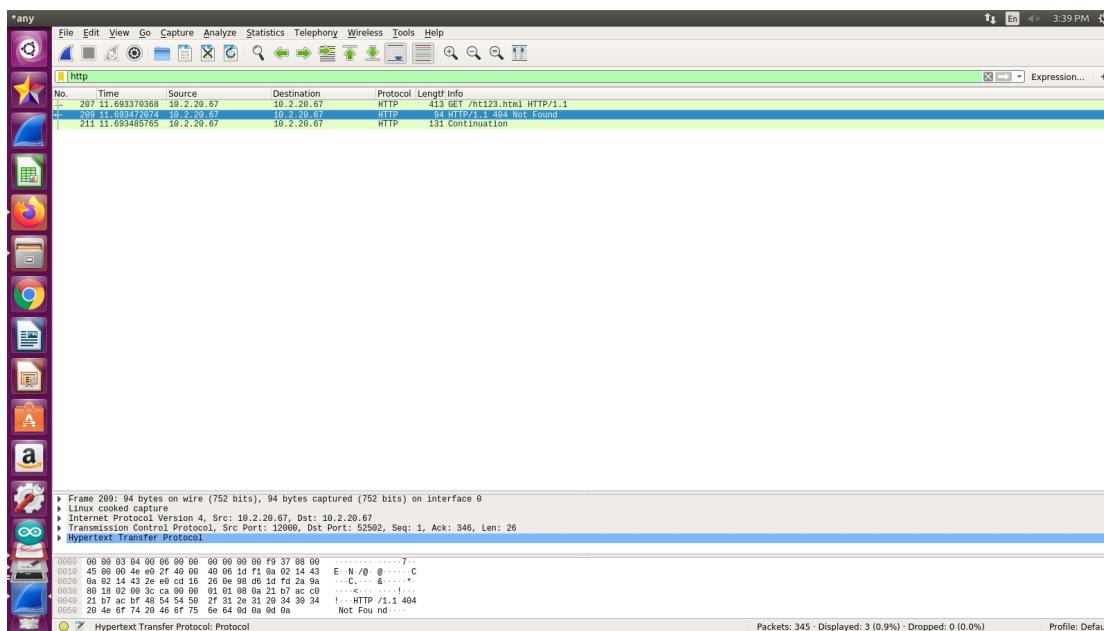
Screenshot of the server running:

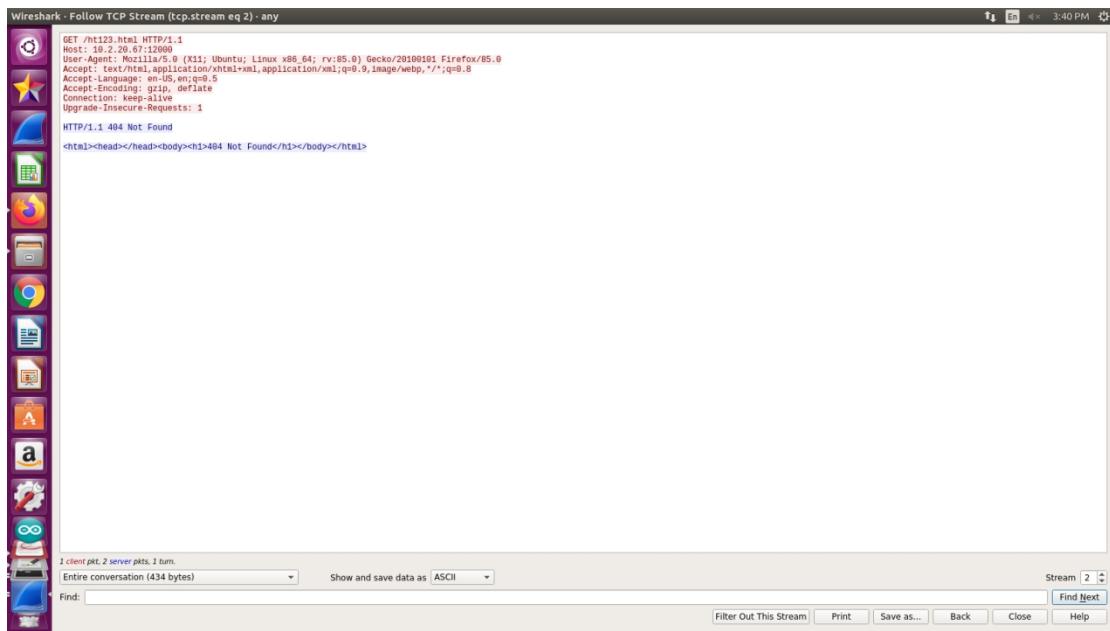


A screenshot of a terminal window on a Linux desktop. The terminal shows the command `python WebServer.py` being run, with the output "Ready to serve..." appearing twice. The terminal has a dark background with light-colored text. On the left, there is a vertical dock containing icons for various applications like a star, a document, a browser, and a terminal.

Here I have used a file called ht123.html which is not present in the server.

Wireshark screenshots:





As I have used a file which is not present in the server hence I am getting an output 404 as seen in the above wireshark screenshots.