Department of Electrical and Computer Engineering

COMPUTER NETWORKS

ENCS3320

PROJECT #1

By:

Sondos Shahin 1200166

Qamar Sharef 1201930

Mohammad Abu Shama 1200270

Istabraq Mohammad 1170256

Instructor:

Dr. Ibrahim Nemer

Introduction

Our project is built on the super client architecture. Based on the concept of socket programming. In this project, we have built a three-part foolproof network that works as follows:

part one :

We explain what ping, tracert, and nslookup are, and then run the following commands:

- Ping a device in the same network, eg from a laptop to a smartphone

2- Bing www.yale.edu

3- Tracert www.yale.edu

4- nslookup www.yale.edu

and take a screenshot of the playback outputs

part two:

Server and client implementation of both TCP and UDP: The client continuously sends numbers from 0 to 1,000,000 to a server listening on port 5566. The server counts the messages received by socket programming used in Python. For each run, we take screenshots of the run and display the number of packets received. We also measured when packets were sent and when packets were received.

part three:

Using socket programming, implement a simple web server in Python. The program should have an HTML webpage that contains the title, names of group members, identifiers, some information about them, etc., and use CSS to make the page look nice.

We took browser screenshots to show the results and tested the project from the browser on itself computer and from a different computer or phone.

Task:

Write program using socket programming a multi-client server TCP program that will do the following:

1) Create a connection between one server and two clients

2) client1 sends data to SERVER.

3) client2 sends data to SERVER.

4) SERVER calculates the Cyclic Redundancy Check (CRC) with KEY (=110) for client1 and client1 data.

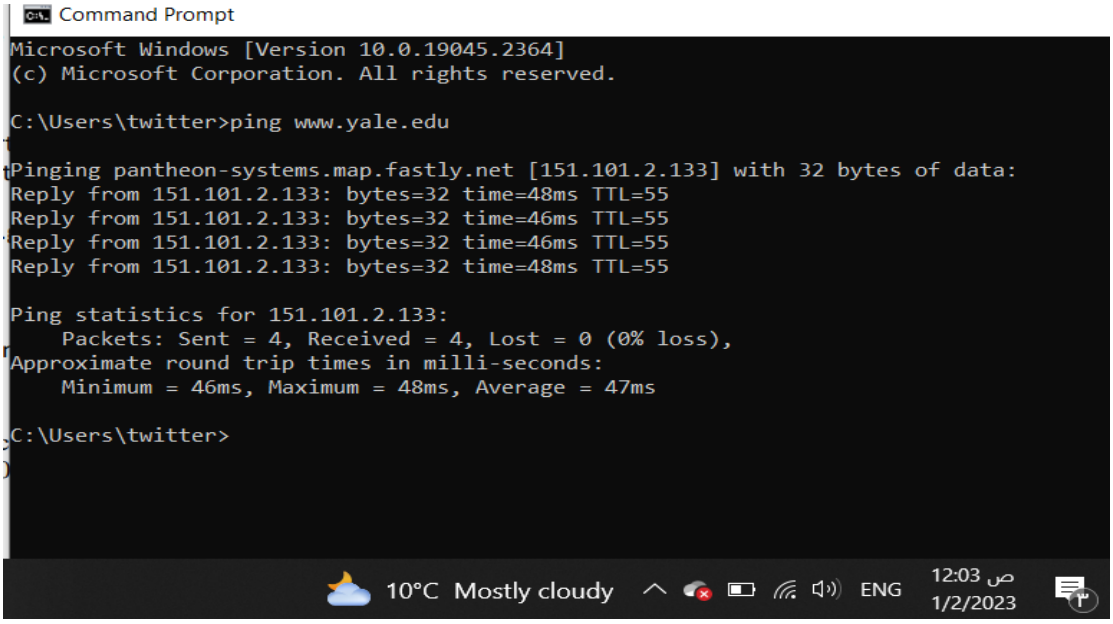
5) SERVER resends the new data along with the CRC bits in binary format to both clients.

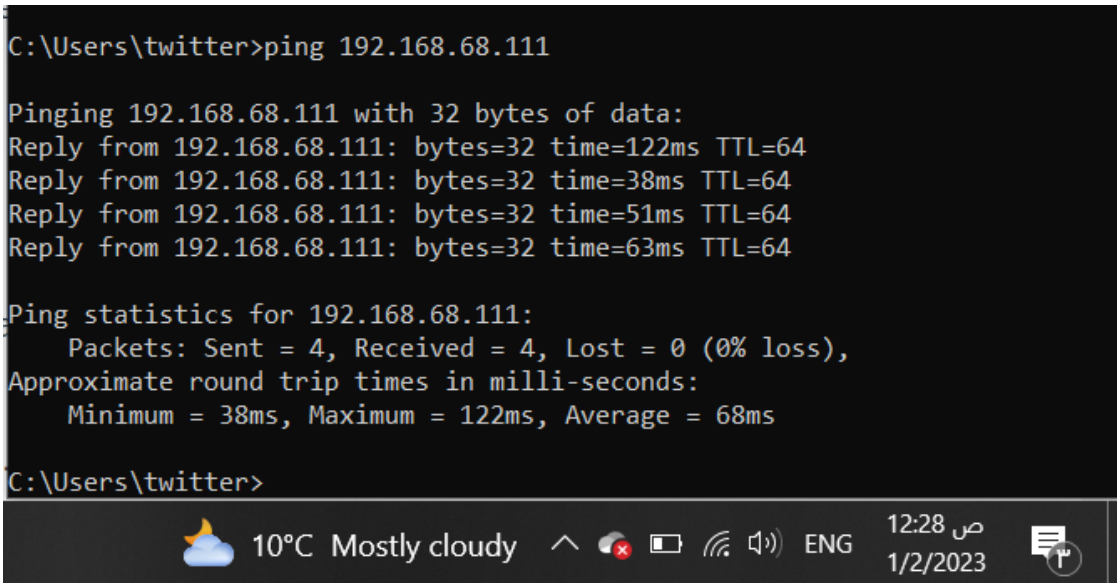
6) Client checks for error with the same key (=110), and displays the appropriate messages for positive and negative ACKs

Part1:

ping is the primary TCP/IP command used to troubleshoot connectivity, reachability, and name resolution. we used “ping” along with the URL you want to ping, In the image below, we’re pinging ping www.yale.edu and getting a normal response. That response shows the URL you’re pinging, the IP address associated with that URL, and the size of the packets being sent on the first line. The next four lines show the replies from each individual packet, including the time (in milliseconds) it took for the response and the time-to-live (TTL) of the packet, which is the amount of time that must pass before the packet is discarded. At the bottom, you’ll see a summary that shows how many packets were sent and received, as well as the minimum, maximum, and average response time.

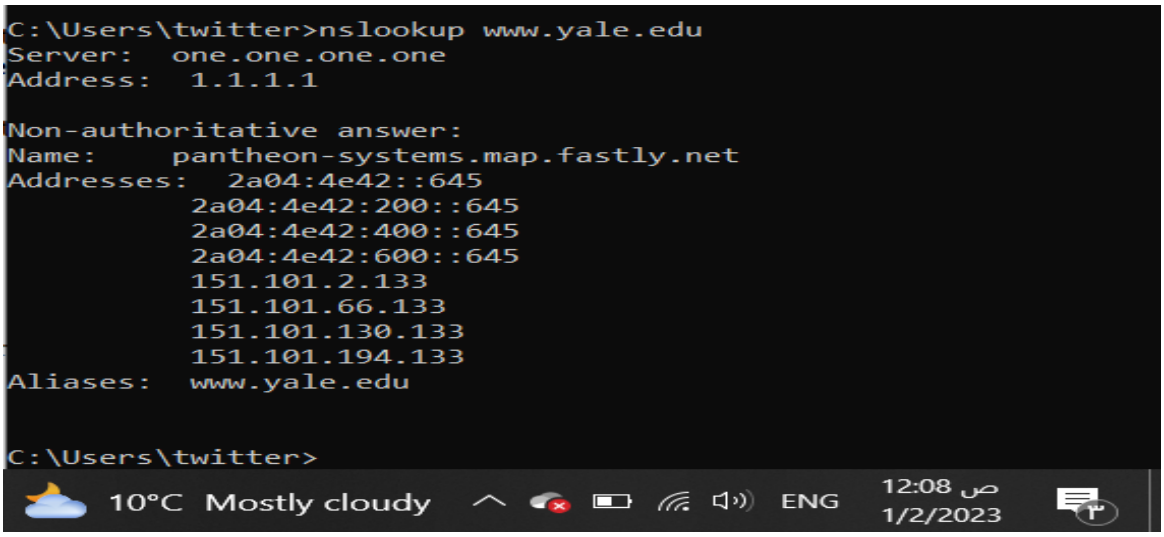
And in the next image(pic.2), we’re pinging the router on our local network using its IP address. We’re also getting a normal response from it.



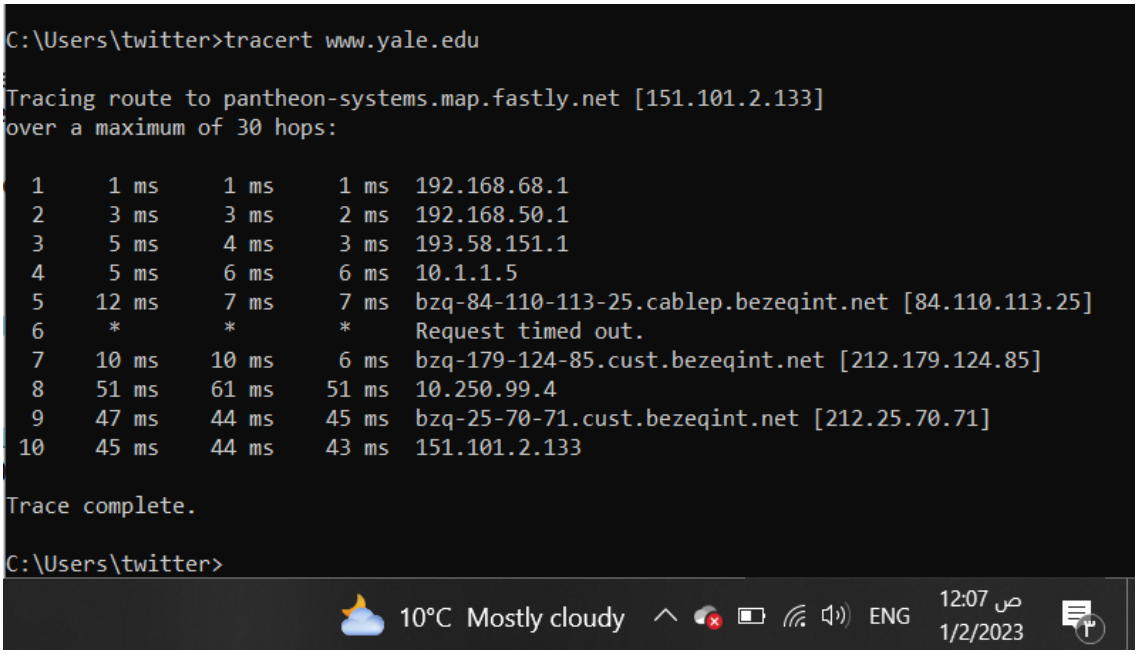


nslookup is a network administration command-line tool available for many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or for any other specific DNS record.

The information displayed will be the local DNS server and its IP address. You can specify a DNS server (IP address), record type, and domain name,and in lat line show you the www.yale.edu address as aliases.



TRACERT (Trace Route), a command-line utility that you can use to trace the path that an Internet Protocol (IP) packet takes to its destination. We can see that tracert again identified 10network devices including our router at 192.168.68.1and all the way through to the target of www.yale.edu, which we can assume uses the public IP address of 151.101.2.133 ,As you can see, tracert didn't resolve any hostnames this time, which significantly sped up the process.



Part2:

UDP server 's show the number of packets received is shown as 1 million (a counter representing). When the program is on the same computer (using localhost)

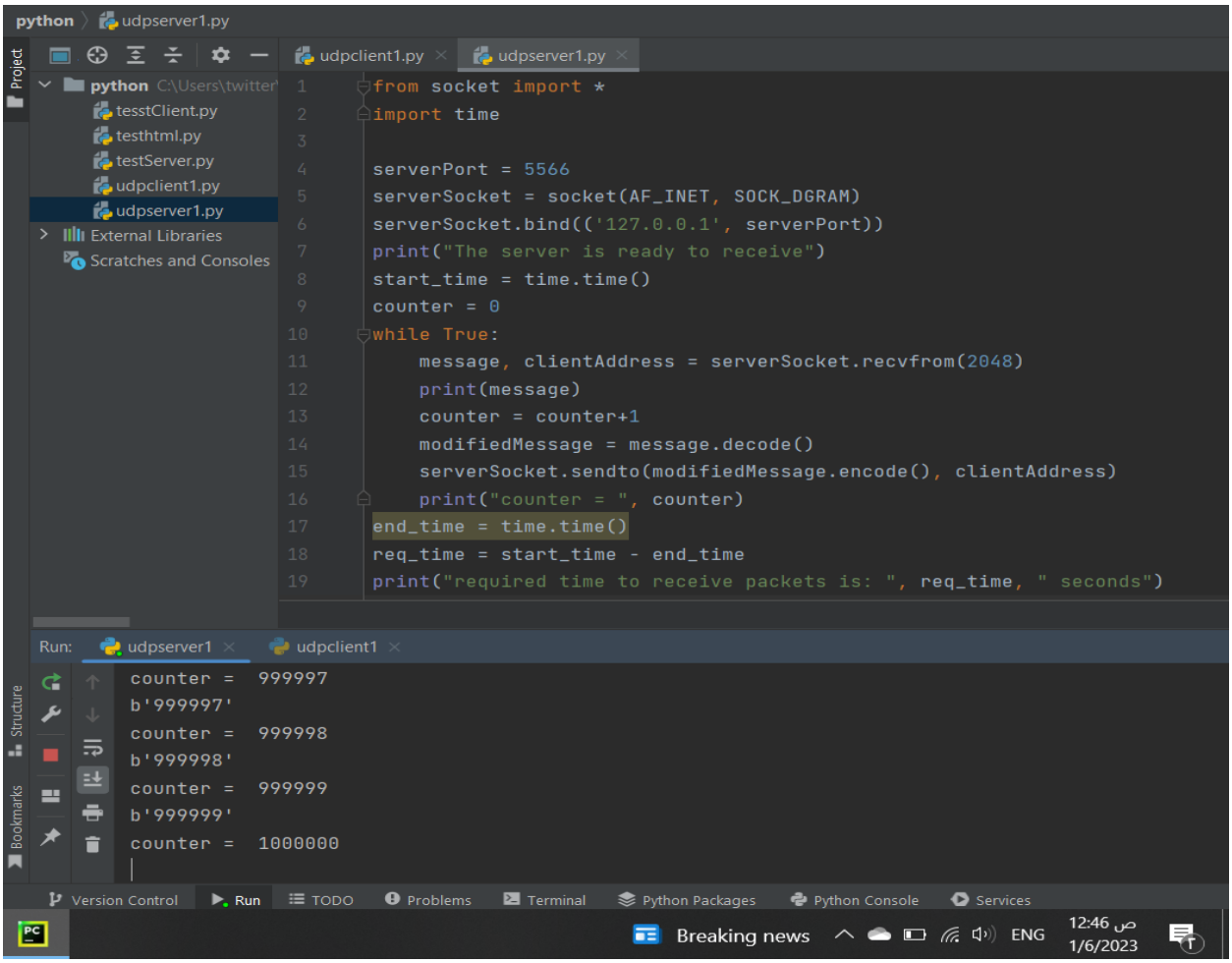


Figure 1/UDP server1

The client UDP continuously sends numbers from 0 to 1,000,000 to the server. When the program is on the same computer (using localhost), the number of packets sender are 1,000,000 ,and the time to send packets its:368.1095399858657 second

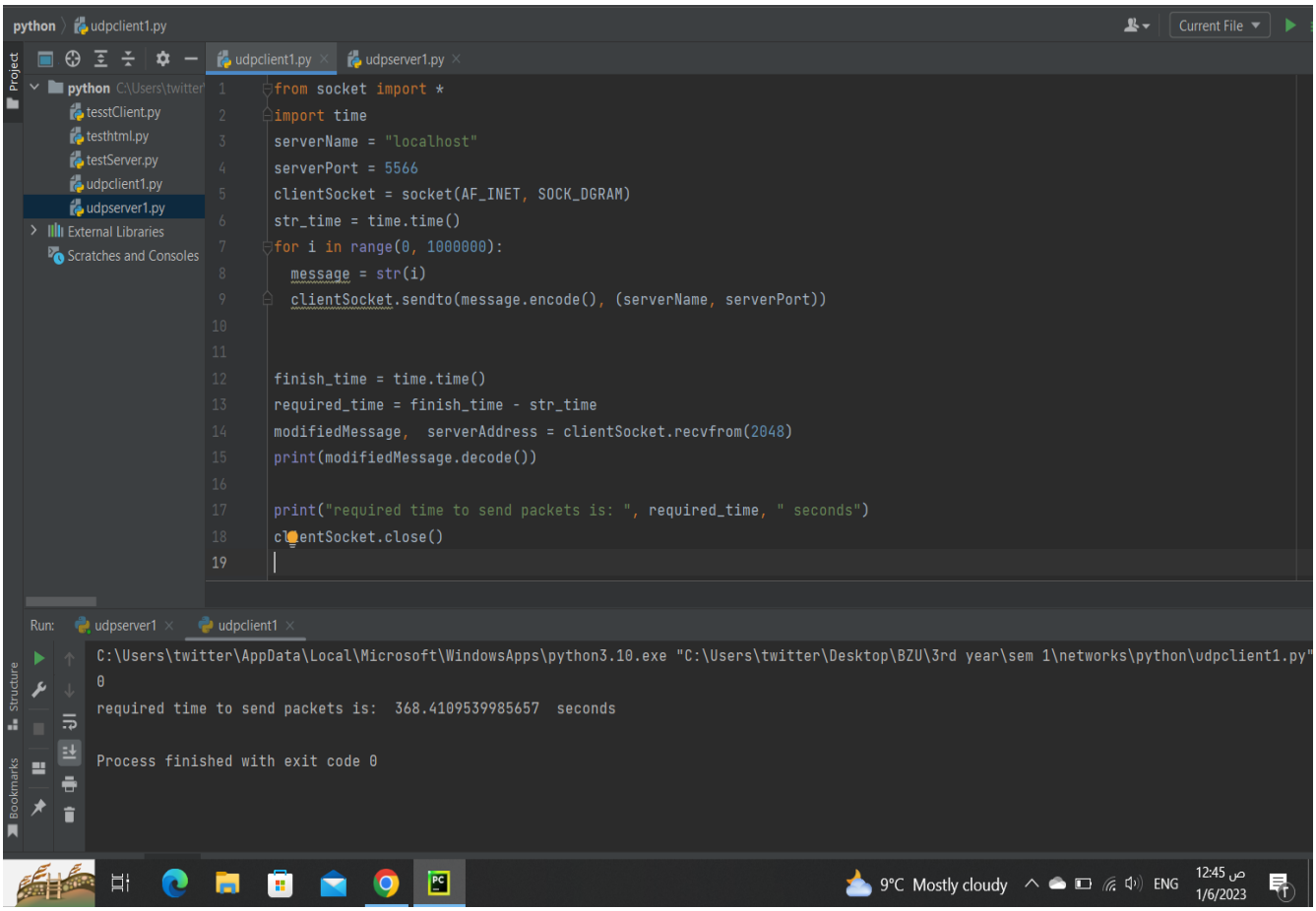


Figure 2/UDP client1

The client TCP continuously sends numbers from 0 to 1,000,000 to the server, when the program is on the same computer (using localhost), number of packets sender are 1,000,000 (from 0 to 999999),and the time to send packets and the time to receive packets from first packet to last packet its:13.095838069915771 second

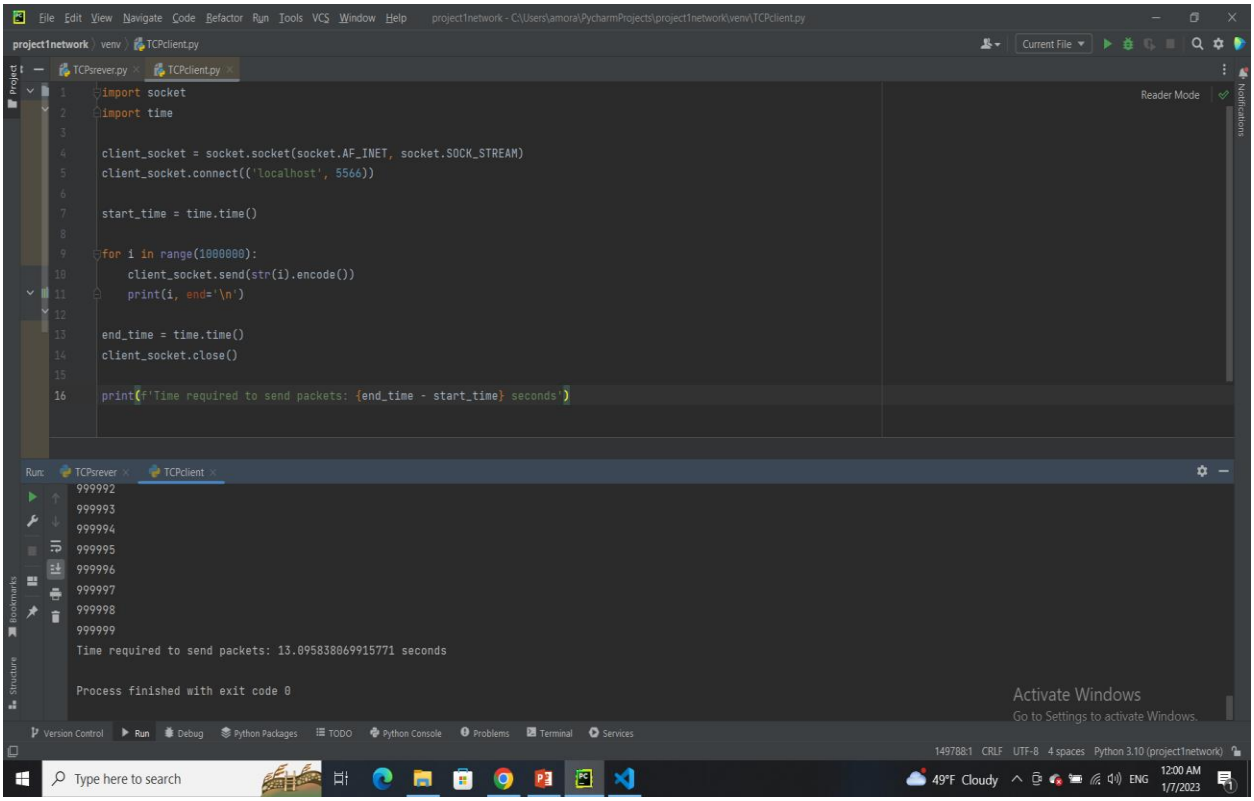


Figure 3/TCP client

TCP server when the program is on the same computer (using localhost), number of packets received are 957338 massages

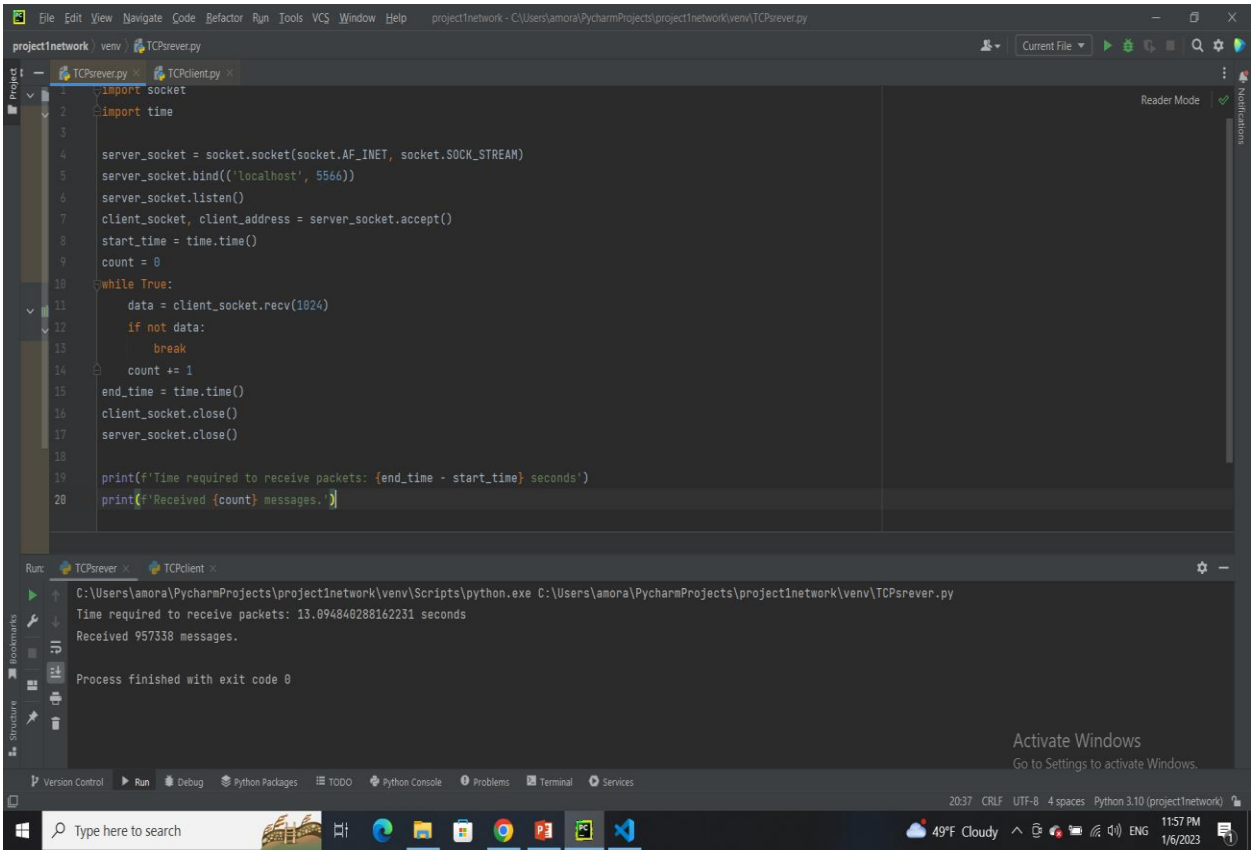


Figure 4/TCP server

The UDP client continuously sends numbers from 0 to 1,000,000 to the server on two different computers connected through cable, time to send packages and time to receive packages from the First Packet to Last: 18.238933123352 seconds

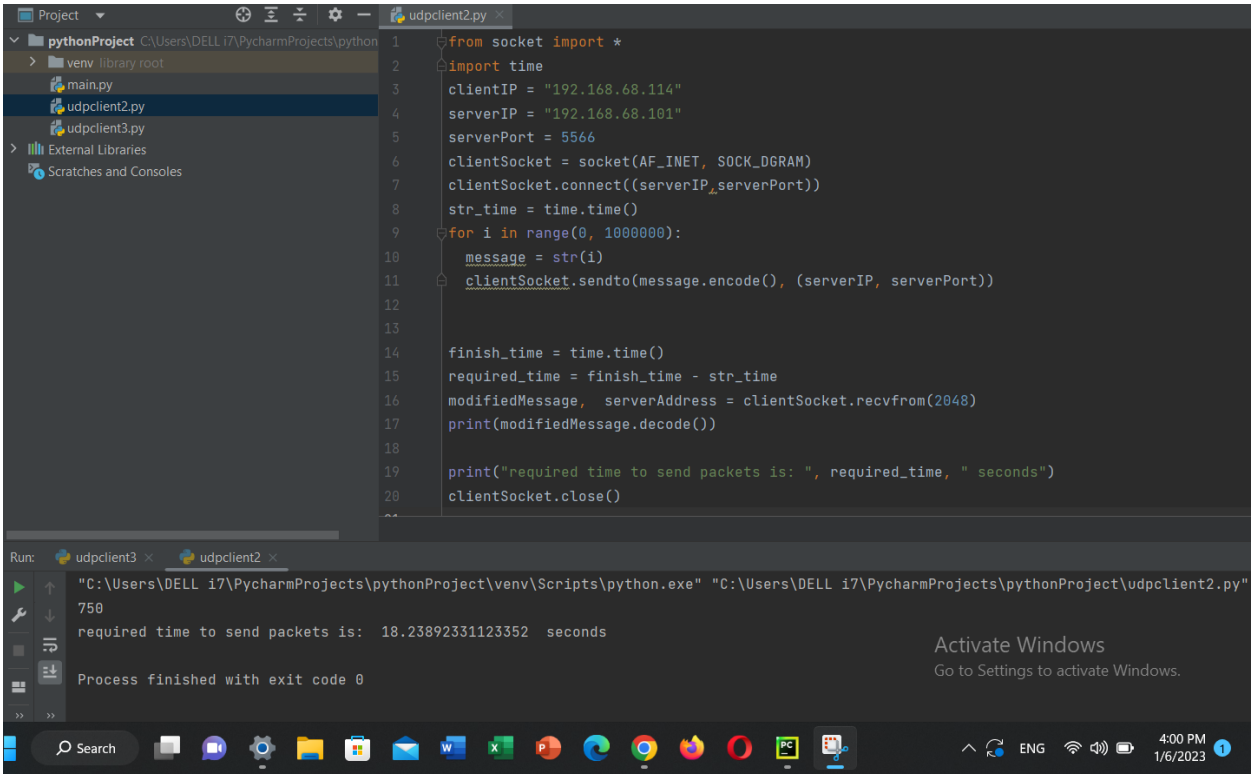


Figure 5/UDP client2

UDP server 's show the number of packets received are shown as 276889 (a counter representing). When the program is on two different computers connected through cable directly or through a switch

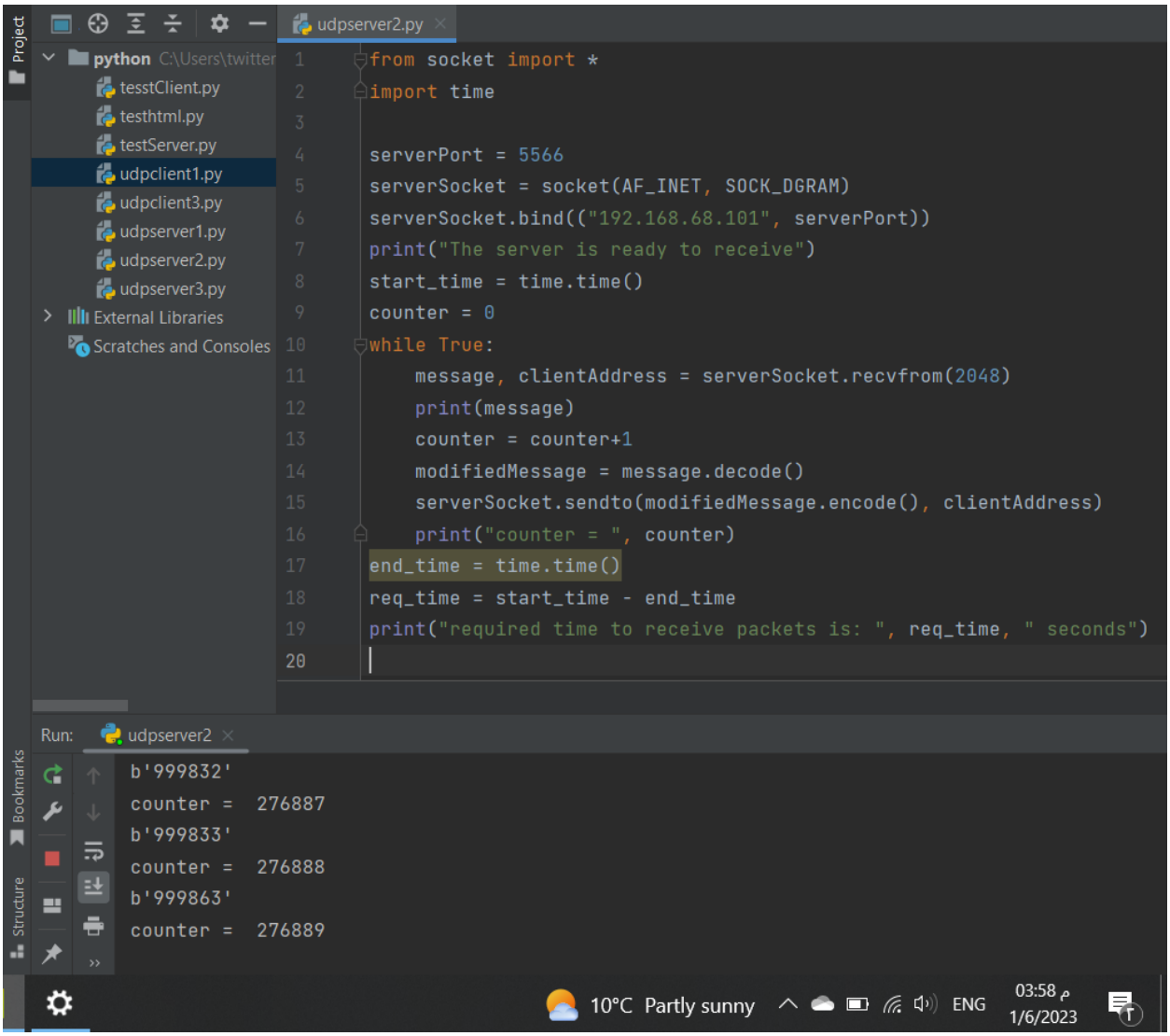


Figure 6/UDP server2

TCP server *When the program is on two different computers connected through Wi-Fi* , number of packets received are 8511 massages ,and the time required to receive packets:13.943995475643 second

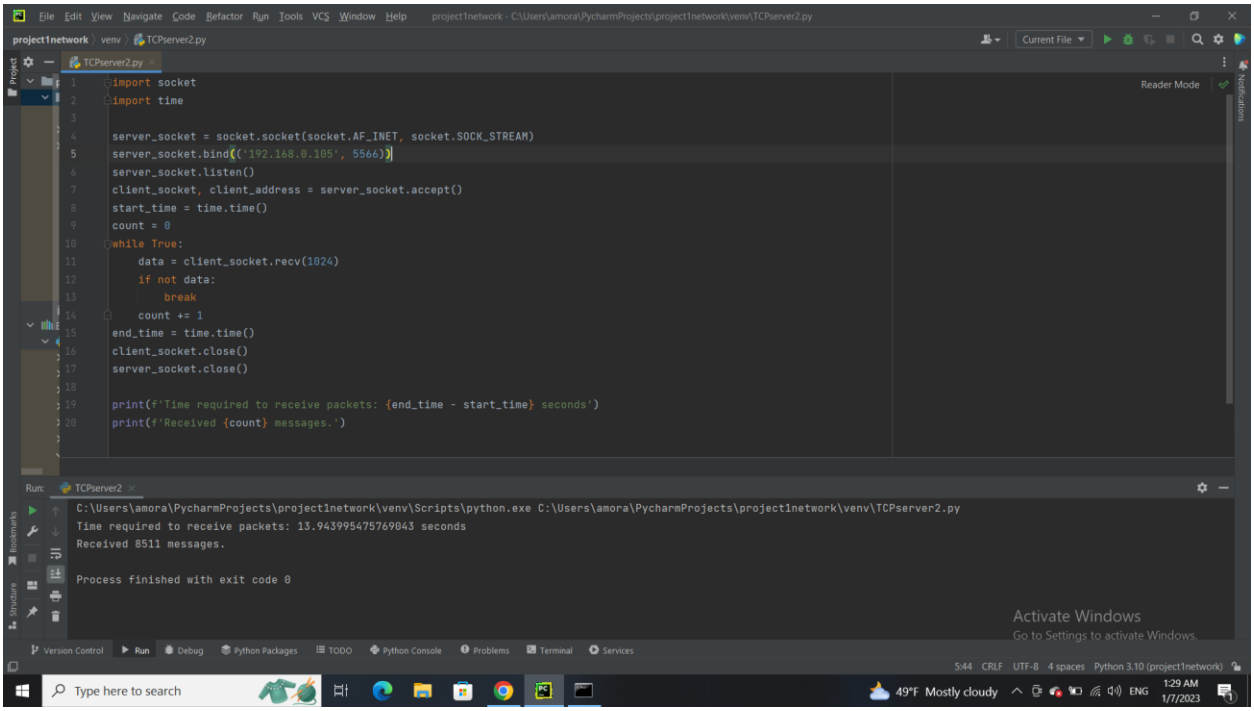


Figure 7/TCP server2

TCP client continuously sends numbers from 0 to 1,000,000 to the server, when the program is *on two different computers connected through Wi-Fi* ,number of packets sender are 1,000,000 (from 0 to 999999),and the time to send packets and the time to receive packets from first packet to last packet its:13. 943288803100586 second.

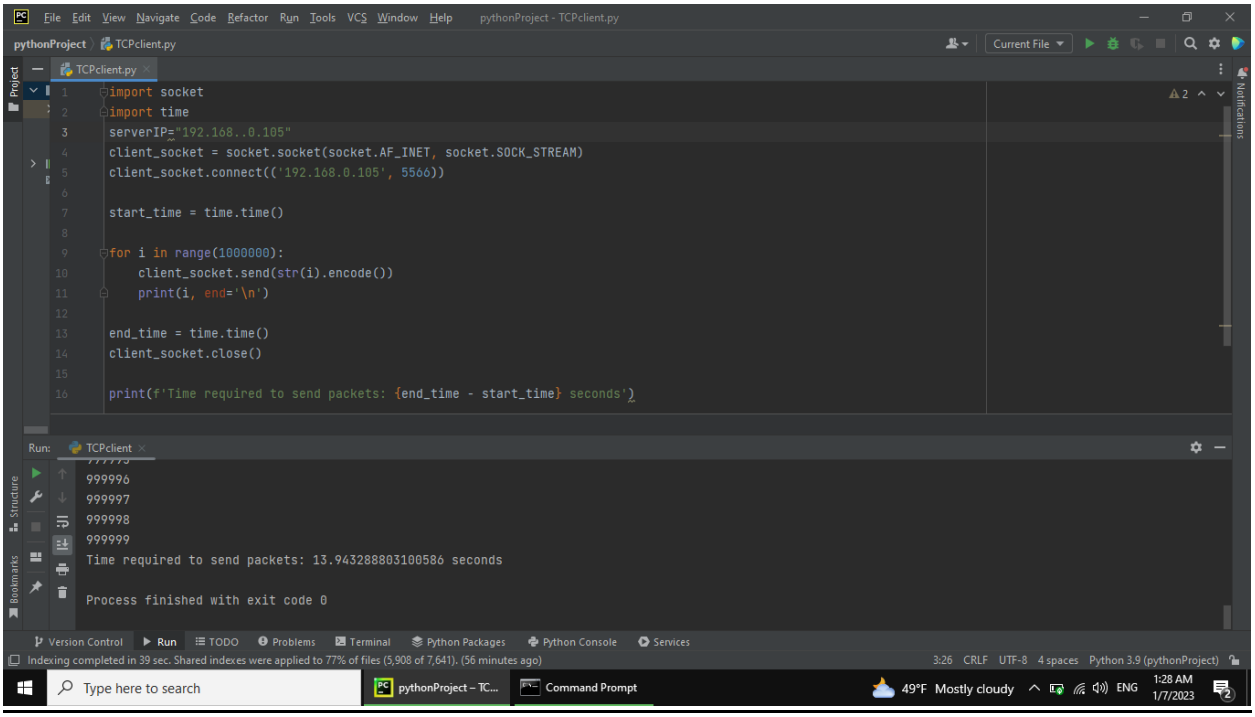


Figure 8/TCP client2

UDP server 's show the number of packets received are shown as 293243 (a counter representing). When the program on 2 different computers connected through wifi.

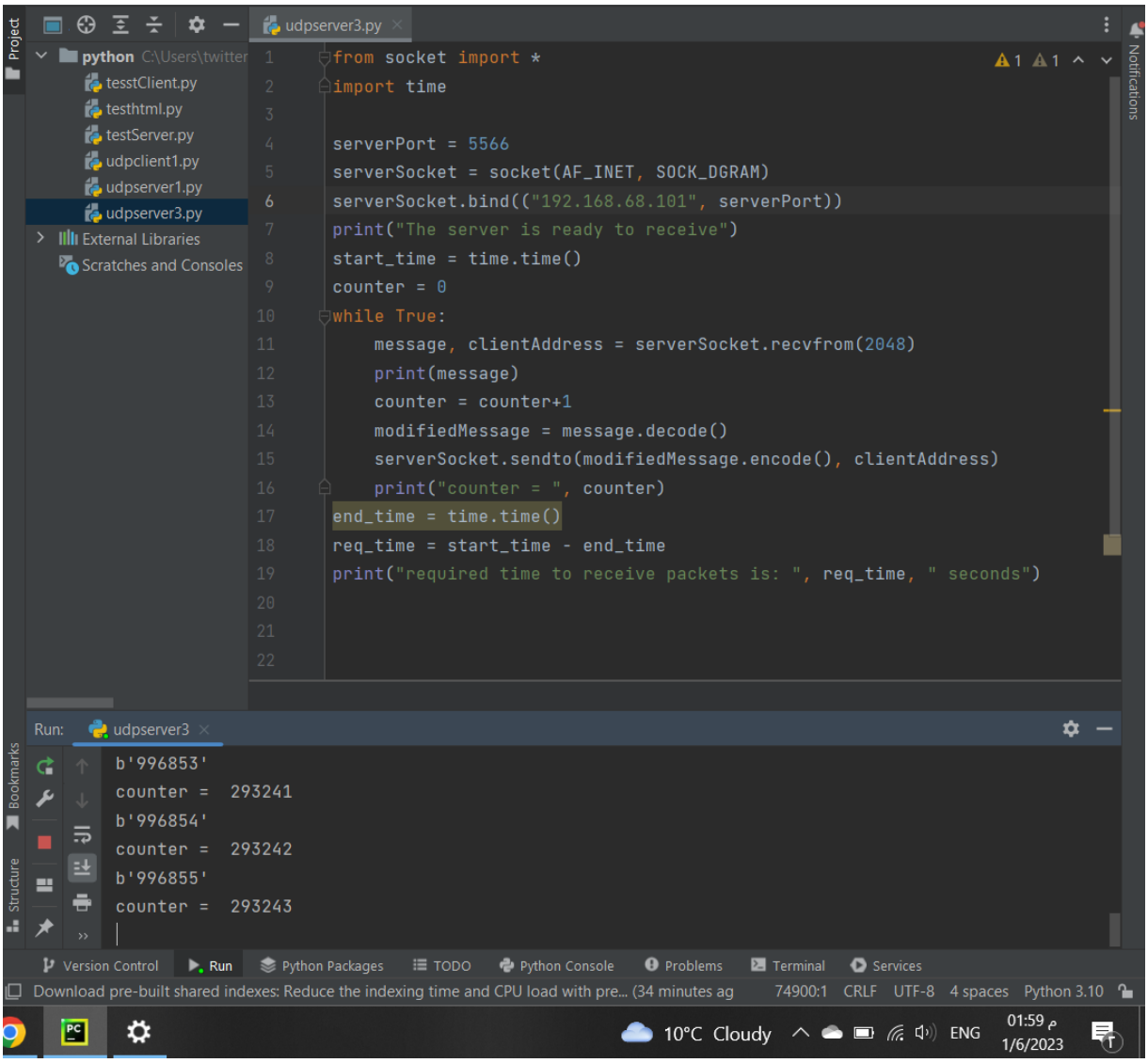


Figure 9/UDP server3

UDP client continuously sends numbers from 0 to 1,000,000 to the server on two different computers connected through Wi-Fi ， time to send packages and time to receive packages from the First Packet to Last: 41.7681710 72006226 seconds.

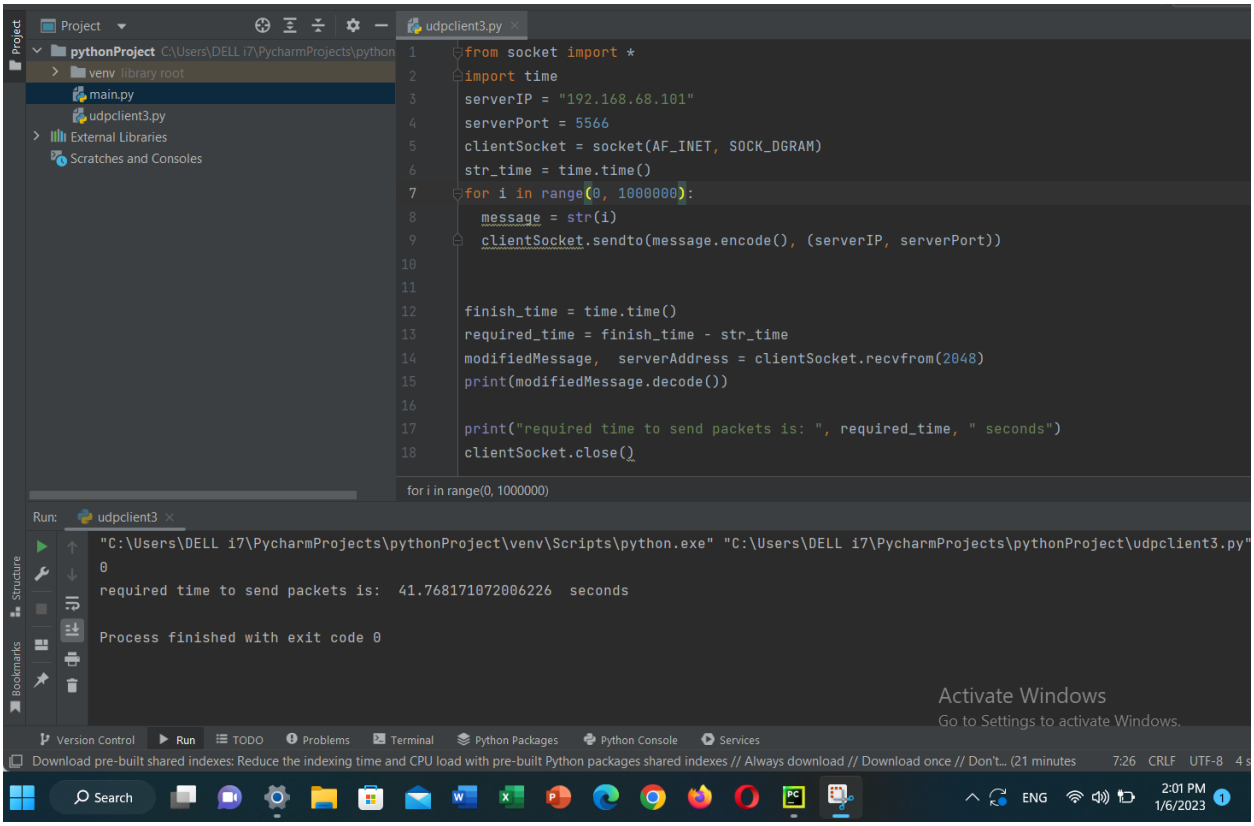


Figure 10/UDP client3

TCP server *When the program is* ***on two different computers connected through cable*** , number of packets received are 8511 massages ,and the time required to receive packets:13.943995475643 second

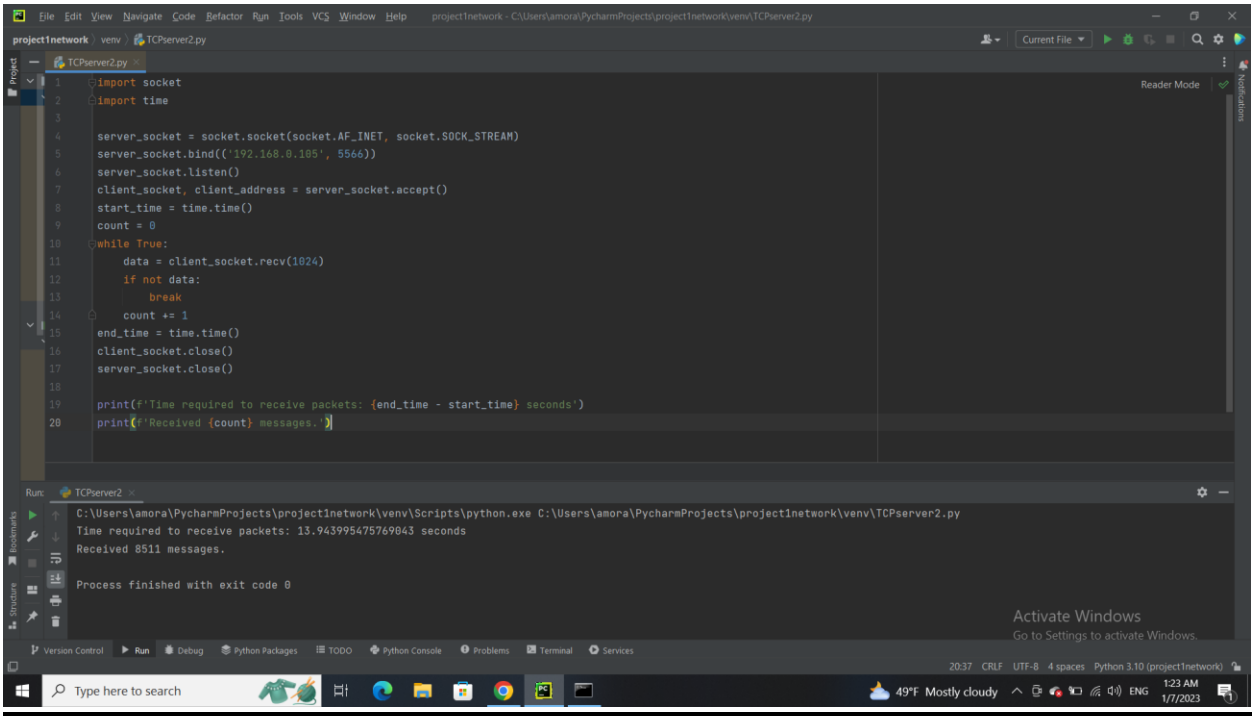


Figure 11/TCP server3

TCP client continuously sends numbers from 0 to 1,000,000 to the server, when the program is *on two different computers connected through cable* ,number of packets sender are 1,000,000 (from 0 to 999999),and the time to send packets and the time to receive packets from first packet to last packet its:13. 943288803100586 second.

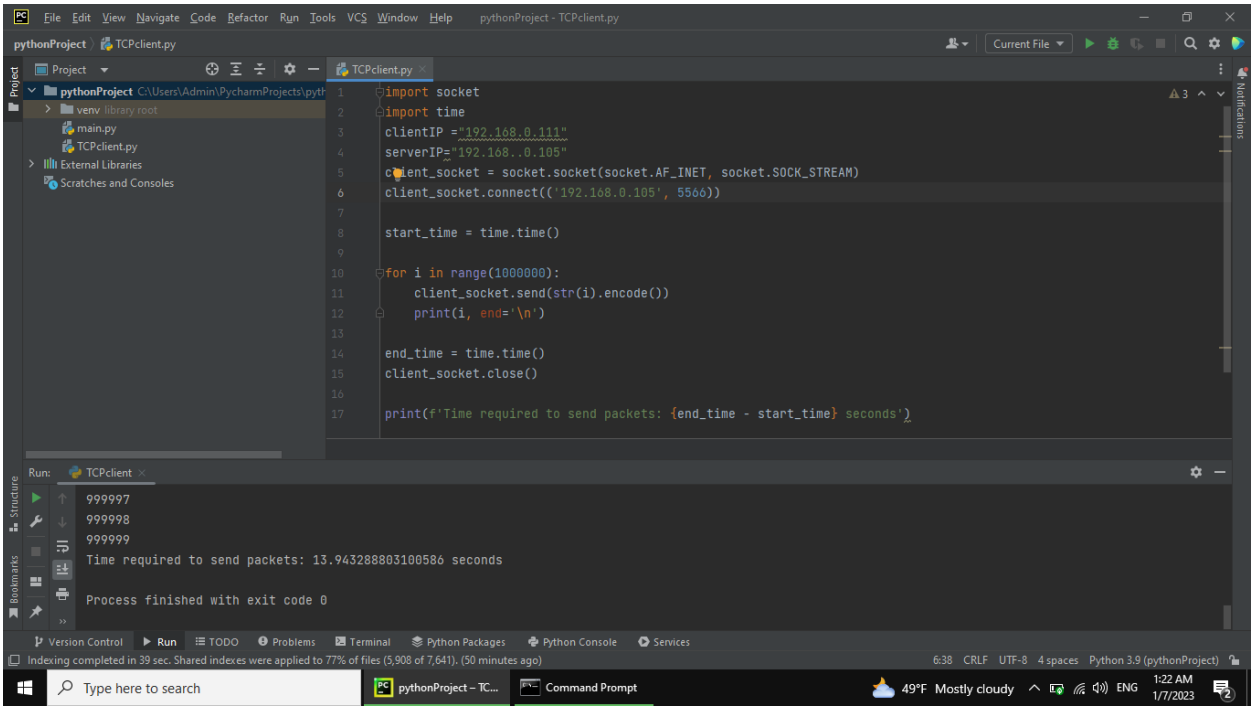
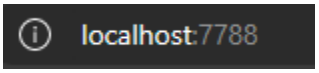


Figure 12/TCP client3

Part3:

Testing website on computer:



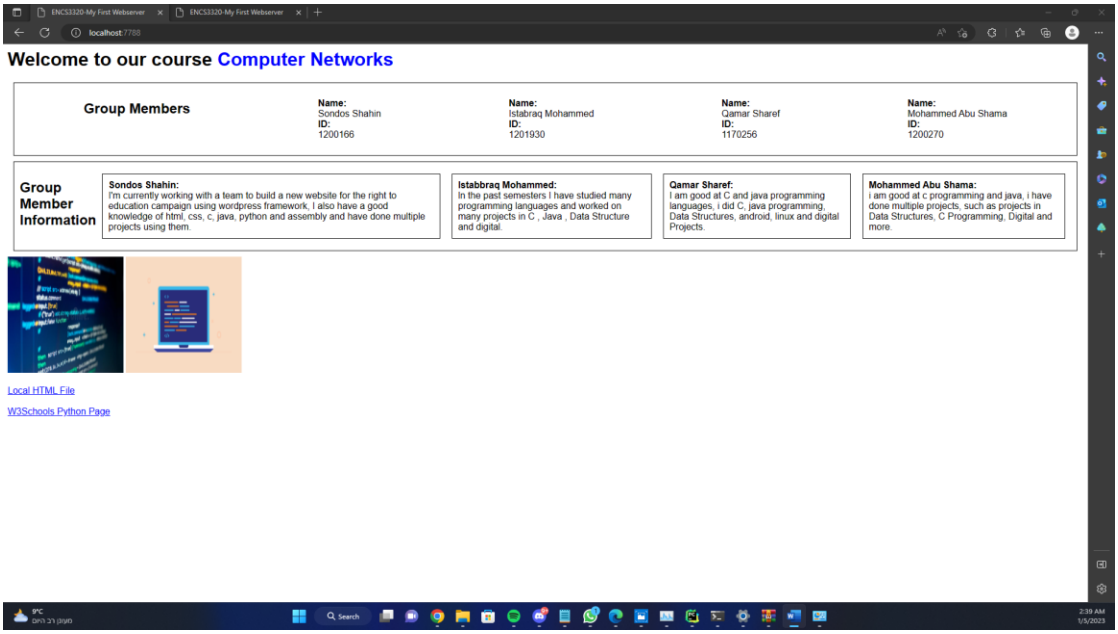
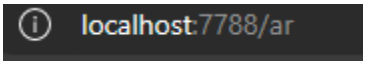


Figure 1/en

For the Arabic page:



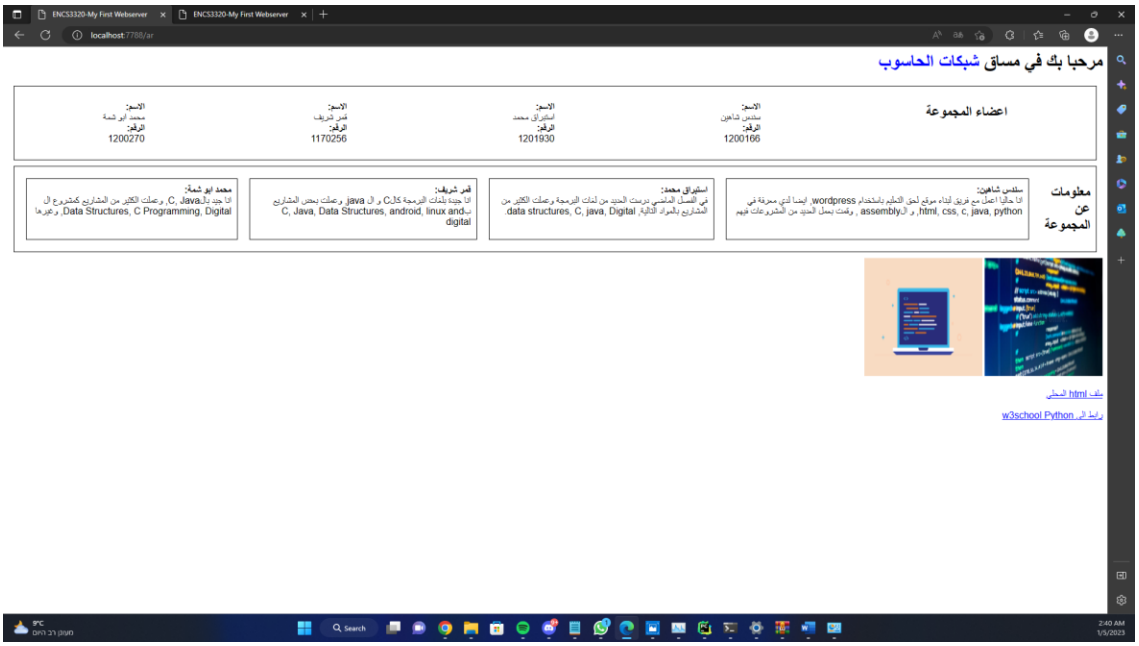
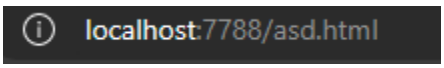


Figure 2/ar

If we request any .html link it display a local_file.html file:



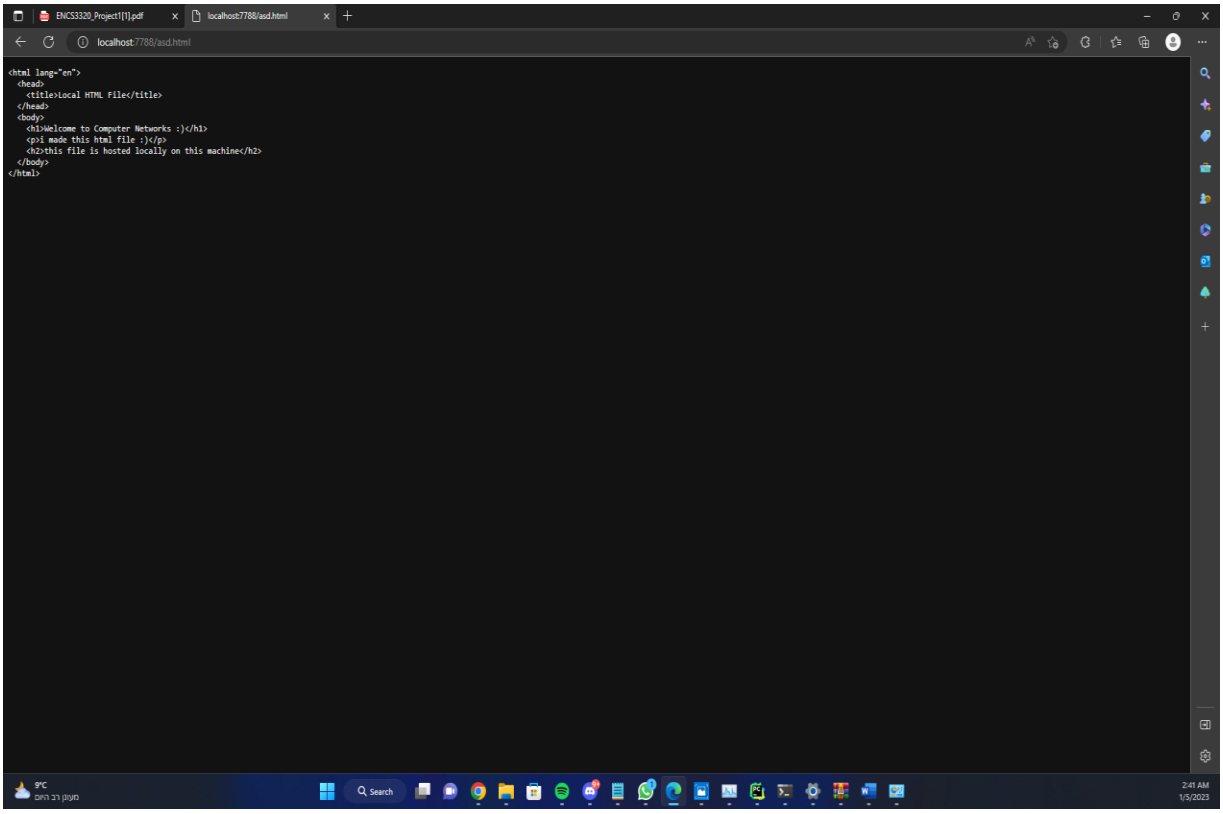


Figure 3/.html

If we request any .css file it display the style.css used to create the main_en and main_ar html files:

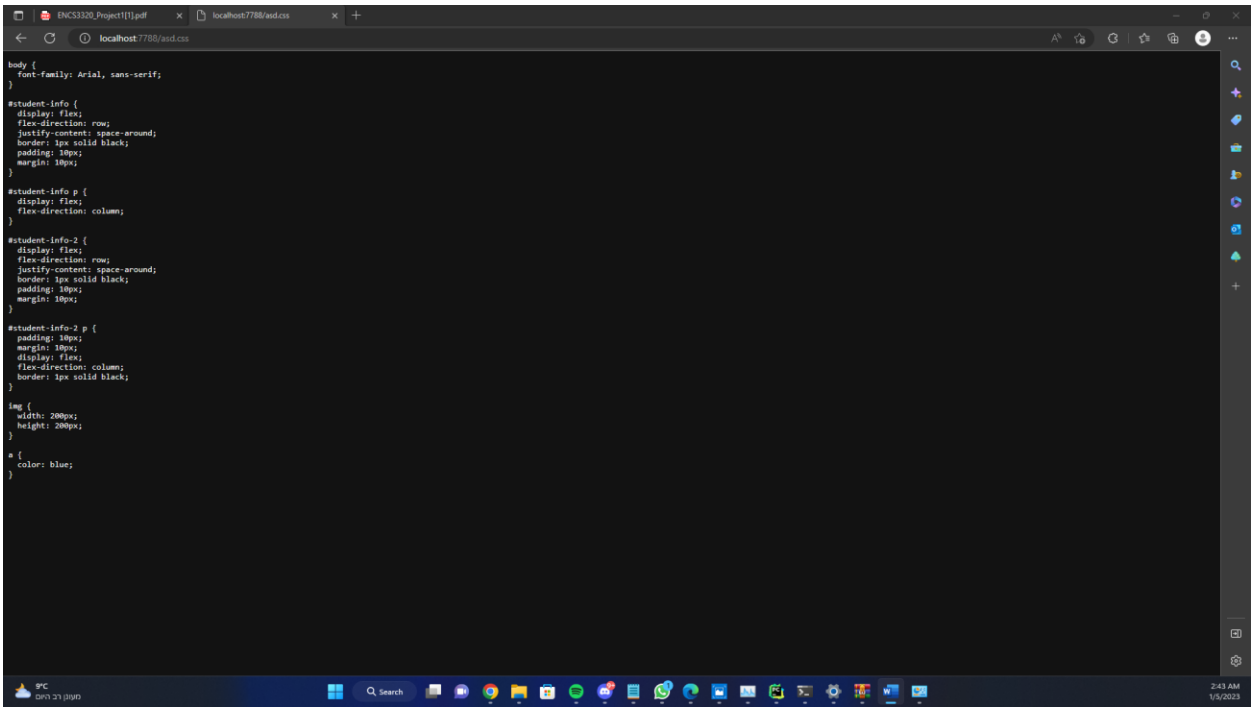
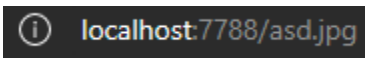
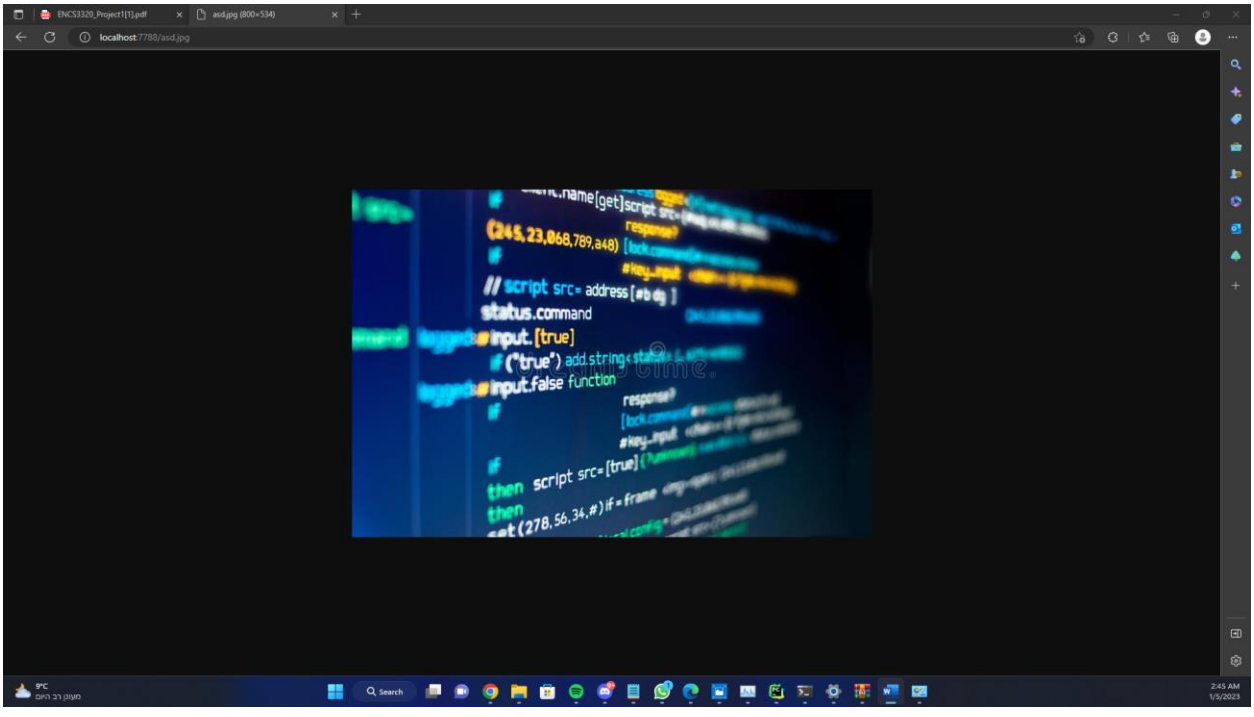


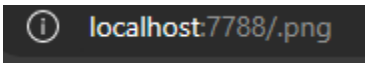
Figure 4/.css

If we request a .jpg file then it displays this image.jpg found online:





If we request a .png file then it displays this image.png found online:



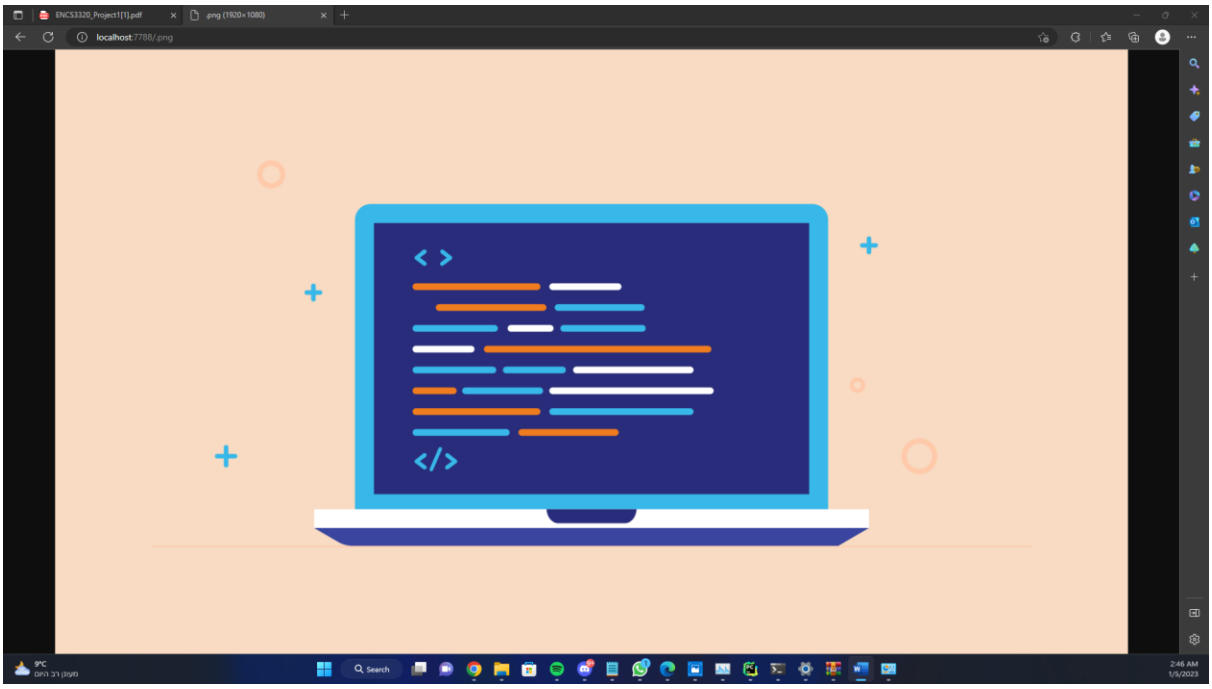
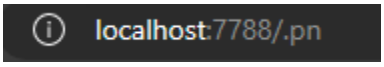


Figure 5/.jpg

If we don’t request any of the previous, the request doesn’t exist so we display the error.html:



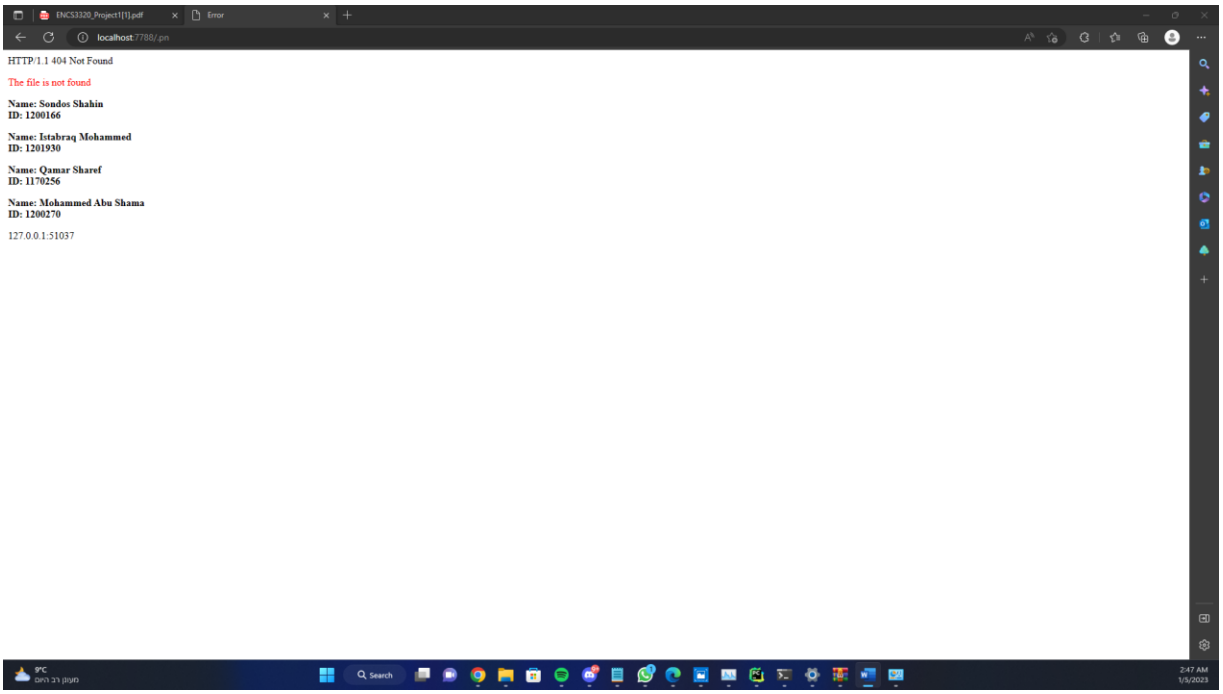


Figure 6/png

Also here are the requests made from phone, to do this we changed the server from local host to our ipv4 local address:

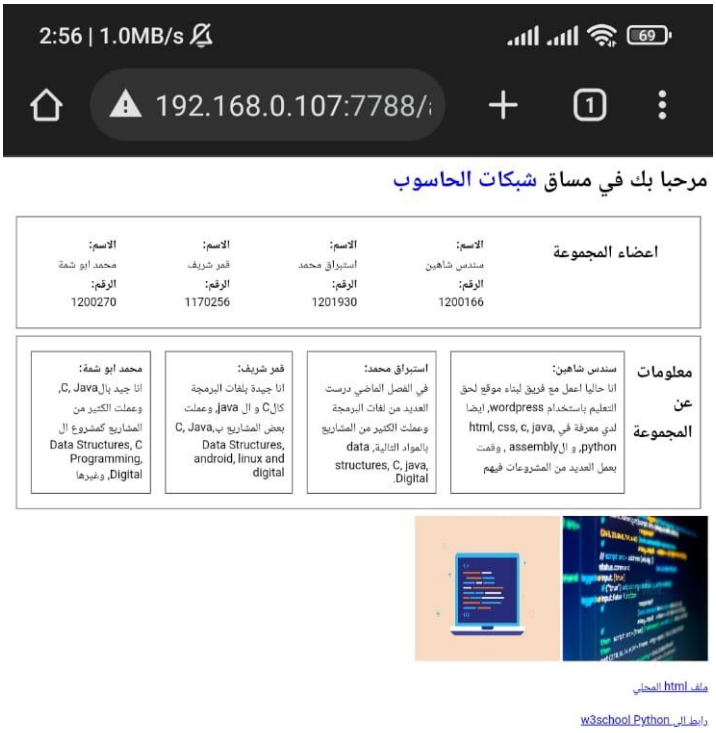


Figure 7phone /ar

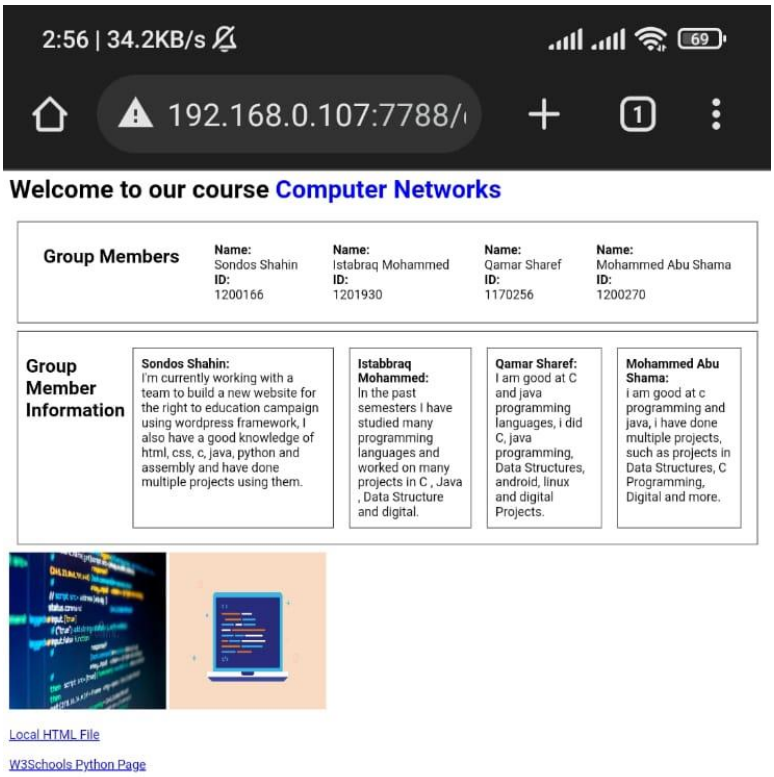


Figure 8phone/en

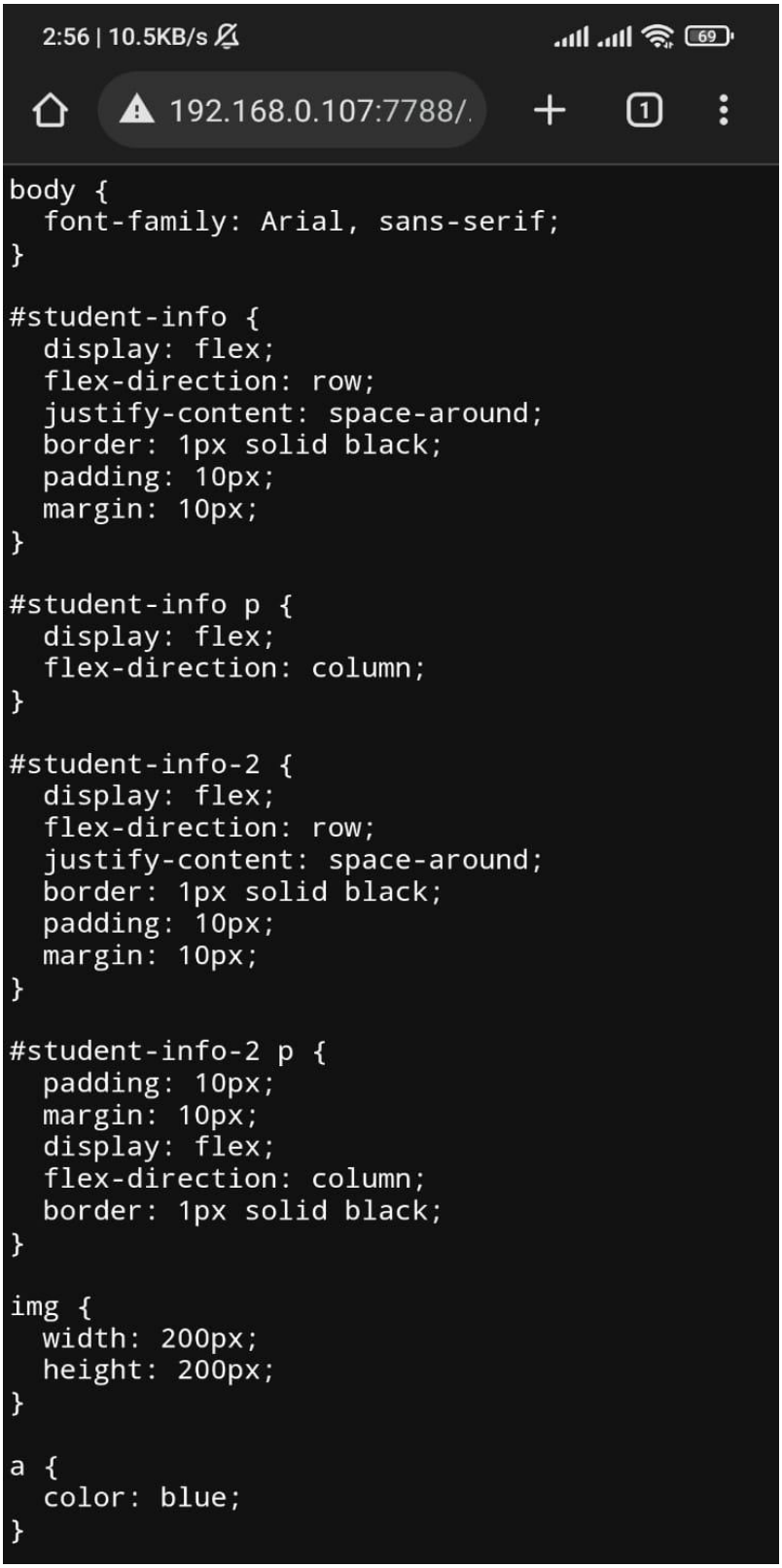

2:56 | 10.5KB/s
192.168.0.107:7788/
body {
font-family: Arial, sans-serif;
}
#student-info {
display: flex;
flex-direction: row;
justify-content: space-around;
border: 1px solid black;
padding: 10px;
margin: 10px;
}
#student-info p {
display: flex;
flex-direction: column;
}
#student-info-2 {
display: flex;
flex-direction: row;
justify-content: space-around;
border: 1px solid black;
padding: 10px;
margin: 10px;
}
#student-info-2 p {
padding: 10px;
margin: 10px;
display: flex;
flex-direction: column;
border: 1px solid black;
}
img {
width: 200px;
height: 200px;
}
a {
color: blue;
}

Figure 9 phonephone/.css

Figure 10phone/.html

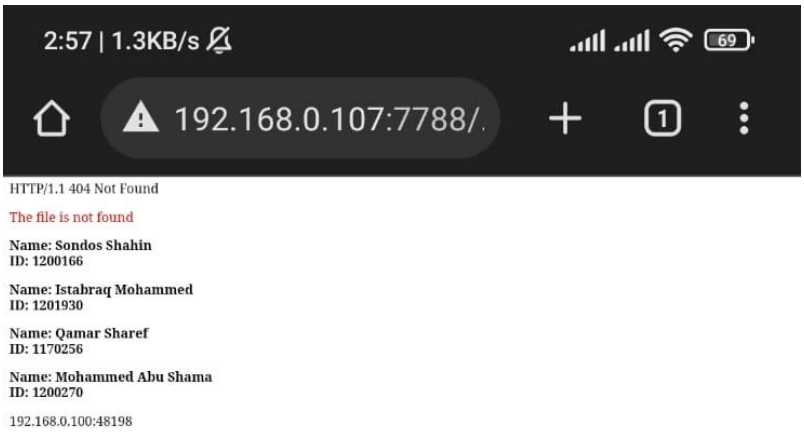


Figure 11phone/notFound

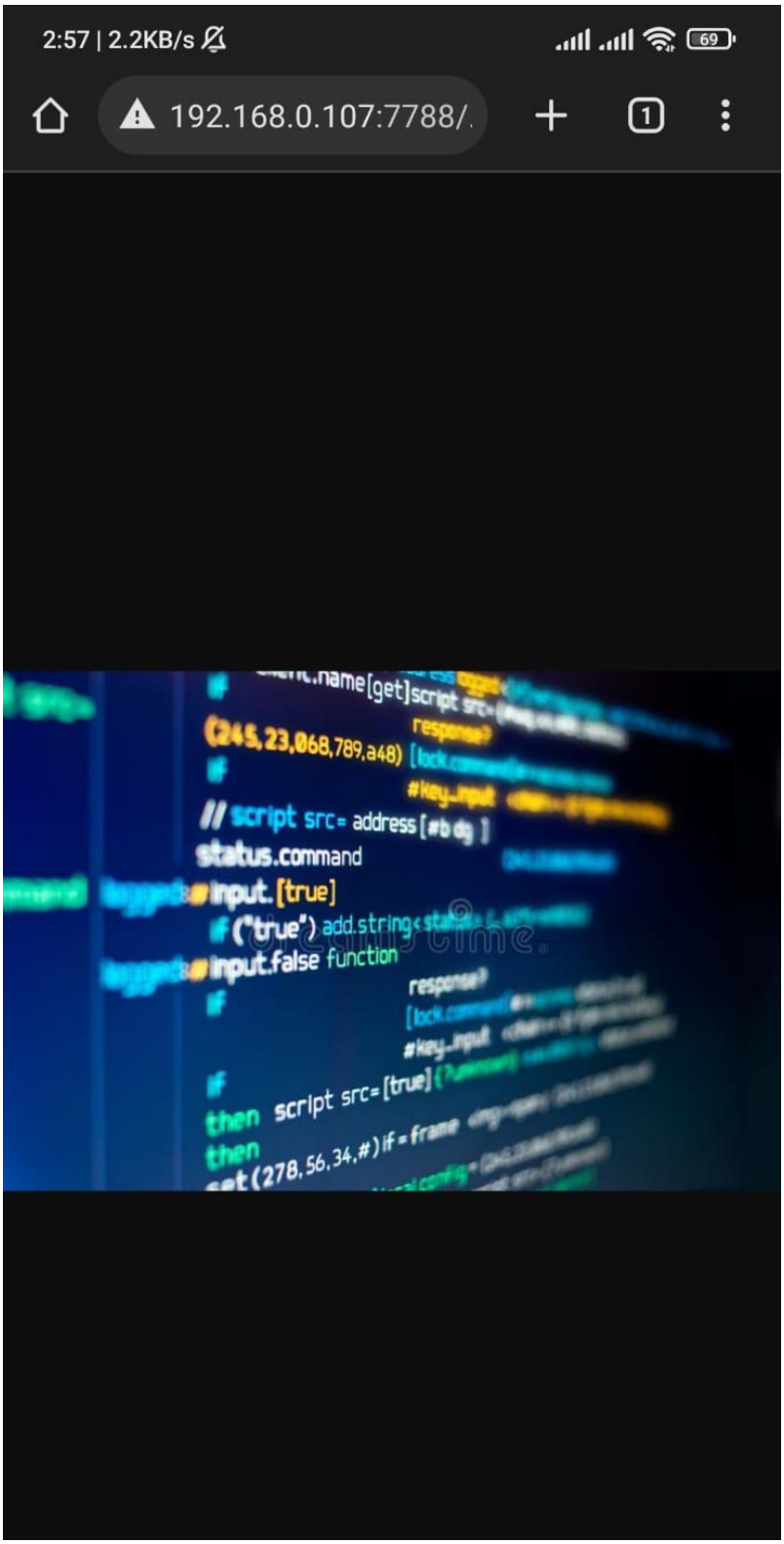


Figure 12phone/.jpg

Task:

First, we run the server:

```
starting up on localhost port 5050
```

Now we run the first client:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
```

Now we open the cmd and run the second client:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
crc in bytes as received: b'\x00\x00\x00l'
crc in decimal: 108
Negative ACK
```

Because the two clients are now connected the server received two messages from the clients:

```
try:
    # send data to server, the message in bytes format
    message = b'hi my name is mohammed'
    print("the message sent: " + str(message) + " in bytes")
    sock.sendall(message)
```

And now the output for client1 is:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
crc in bytes as received: b'\x00\x00\x00l'
crc in decimal: 108
Negative ACK

Process finished with exit code 0
```

After checking the errors and calculating the CRC for both clients.

The final output of the server:

```
starting up on localhost port 5050
connection from ('127.0.0.1', 64374)
connection from ('127.0.0.1', 64375)
client1:
message: b'hi my name is mohammed'
Cyclic Redundancy Check: 108
client2:
message: b'hi my name is mohammed'
Cyclic Redundancy Check: 108

Process finished with exit code 0
```

Another test run by changing the message to hello there:

Client1:

```
connecting to localhost port 5050
the message sent: b'hello there' in bytes
crc in bytes as received: b'\x00\x00\x00B'
crc in decimal: 66
Negative ACK
```

Client2:

```
connecting to localhost port 5050
the message sent: b'hello there' in bytes
crc in bytes as received: b'\x00\x00\x00B'
crc in decimal: 66
Negative ACK
```

Server:

```
starting up on localhost port 5050
connection from ('127.0.0.1', 64383)
connection from ('127.0.0.1', 64385)
client1:
message: b'hello there'
Cyclic Redundancy Check: 66
client2:
message: b'hello there'
Cyclic Redundancy Check: 66
```