Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENEE3320, COMPUTER NETWORTKS

Course project 1

Prepared by: **ID:**

Lama Naser **1200190**

Jehad Hamayel **1200348**

Sec:1

Instructor: Dr.Abd Alkareem Awad .

Abstract

This project consists of three parts. The first part is to run some commands on the terminal; ping a device in the same network, ping www.yale.edu website, tracert and nslookup it. The second part is implementing server and client application both for TCP and for UDP: A client continuously sends the numbers from 0 to 1000,000 to a server listening on port 5566. The server counts the received messages. The last part is to implement a simple but a complete web server in python that is listening on port 7788. Both html and css are used for the design of our website.

Contents

Table Of Figures

Part 1

1- What are ping, tracert, nslookup

- **ping**: is a command-line utility acts as a test to see if a networked device reachable**.**

- **tracert**: is a command-line utility that can be used to trace the path for a protocol packet takes to its destination.

- **nslookup**: is a command that queries to domain name and addresses servers.

2- Screenshot for some commands

Ping a device in the same network

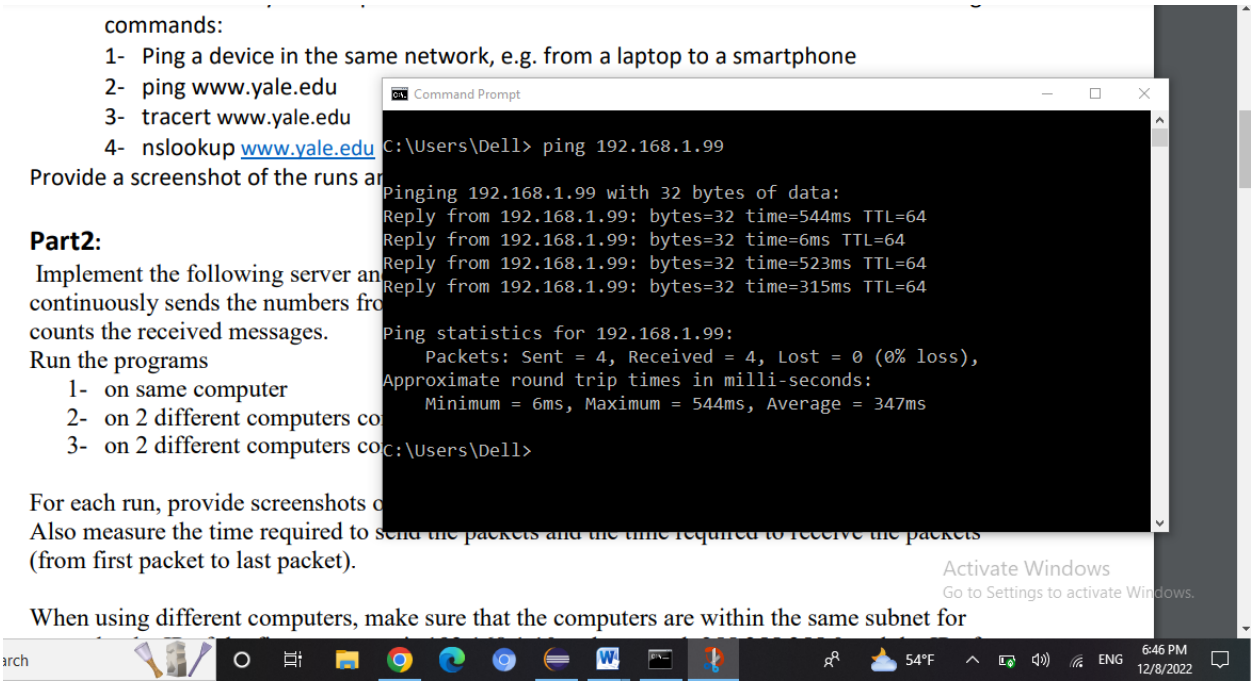


Figure 1: ping a device on the same network

Here a successful reply from 192.168.1.99 is received, so the device is connected to the internet, and all hardware such as cables, the network card, the router and the modem are all working correctly, including the internet service provider. Four packets response were received from 192.168.1.99 with their TTL, with an average of 347ms delay.

ping www.yale.edu

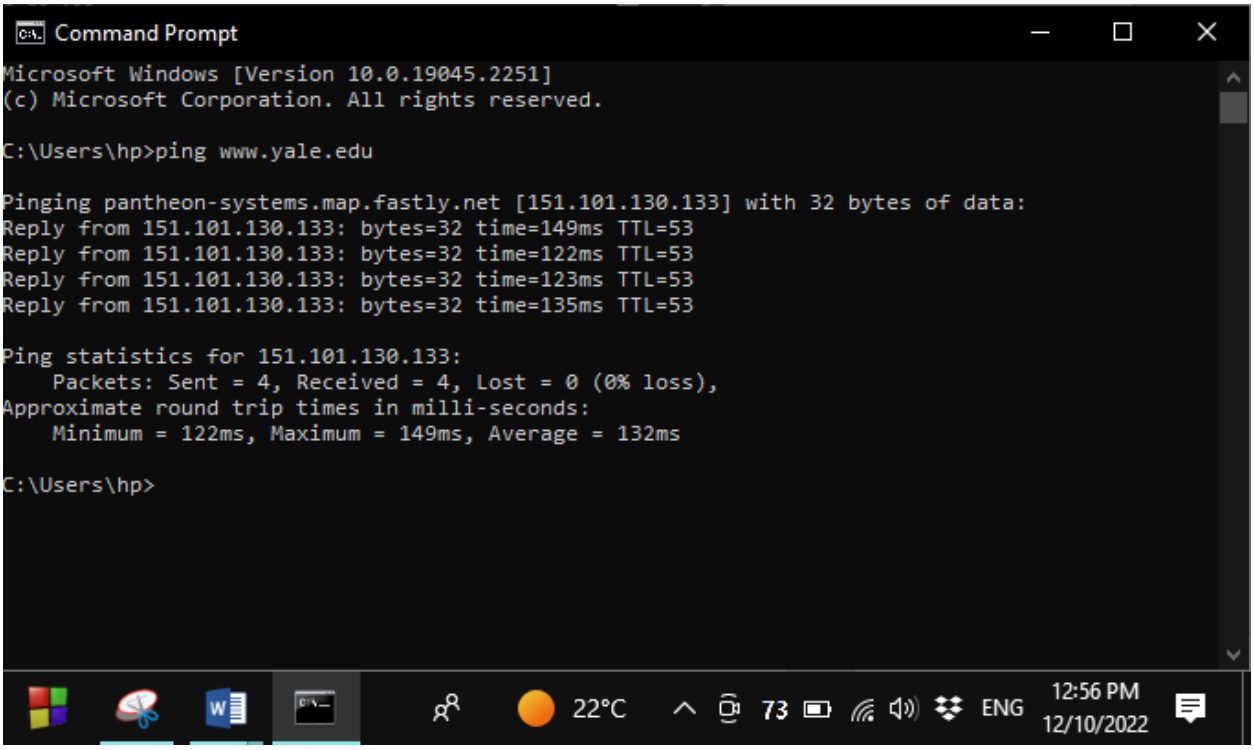


Figure 2: ping www.yale.edu

Pinging a website is actually pinging the domain name and not its IP address. Because also name resolution issues related to DNS is tested, as the IP address is not memorized. Here a successful reply from **www.yale.edu** is received, so the device is connected to the internet, and all hardware such as cables, the network card, the router and the modem are all working correctly, including the internet service provider. Four packets response were received from 151.101.130.133 with their TTL, with an average of 132ms delay.

tracert www.yale.edu

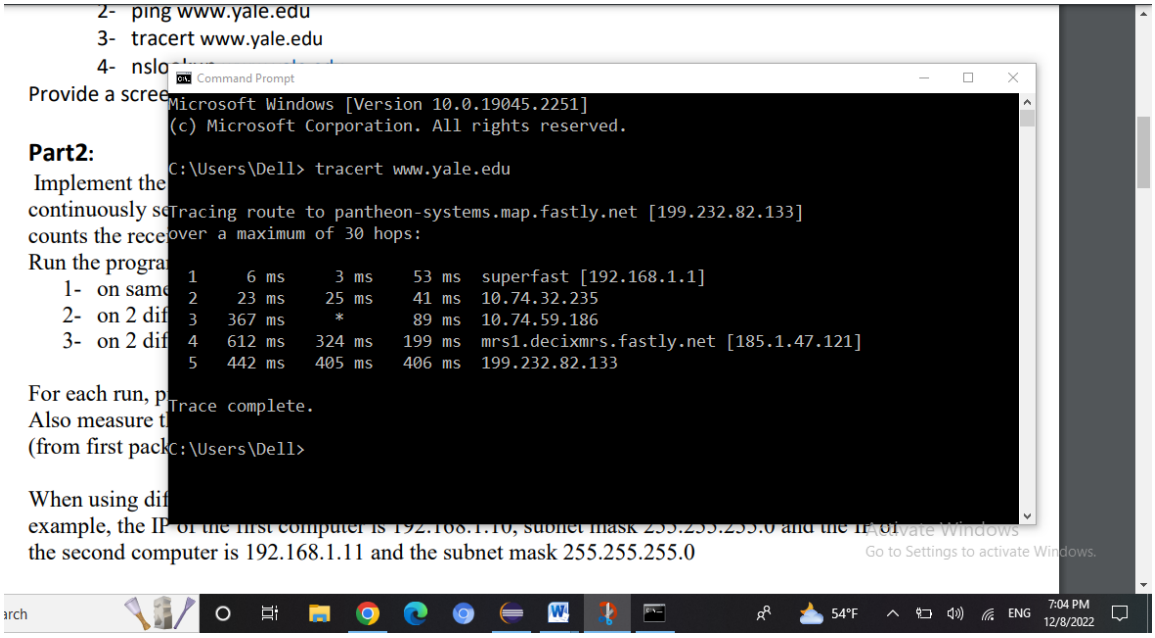


Figure 3 : tracert www.yale.edu

Tracert command (tracert) is a utility designed can be used for troubleshooting of connection issue, displays the time it takes for a packet of information to travel between a local device and a destination of IP address or domain. After performing this command for **www.yale.edu**, the results displayed are the hops that data packets take along their path to destination desired. When the measurements are stars (*) in request timed out and indicates that the router at that hop doesn't respond to the traceroute requests.

nslookup www.yale.edu

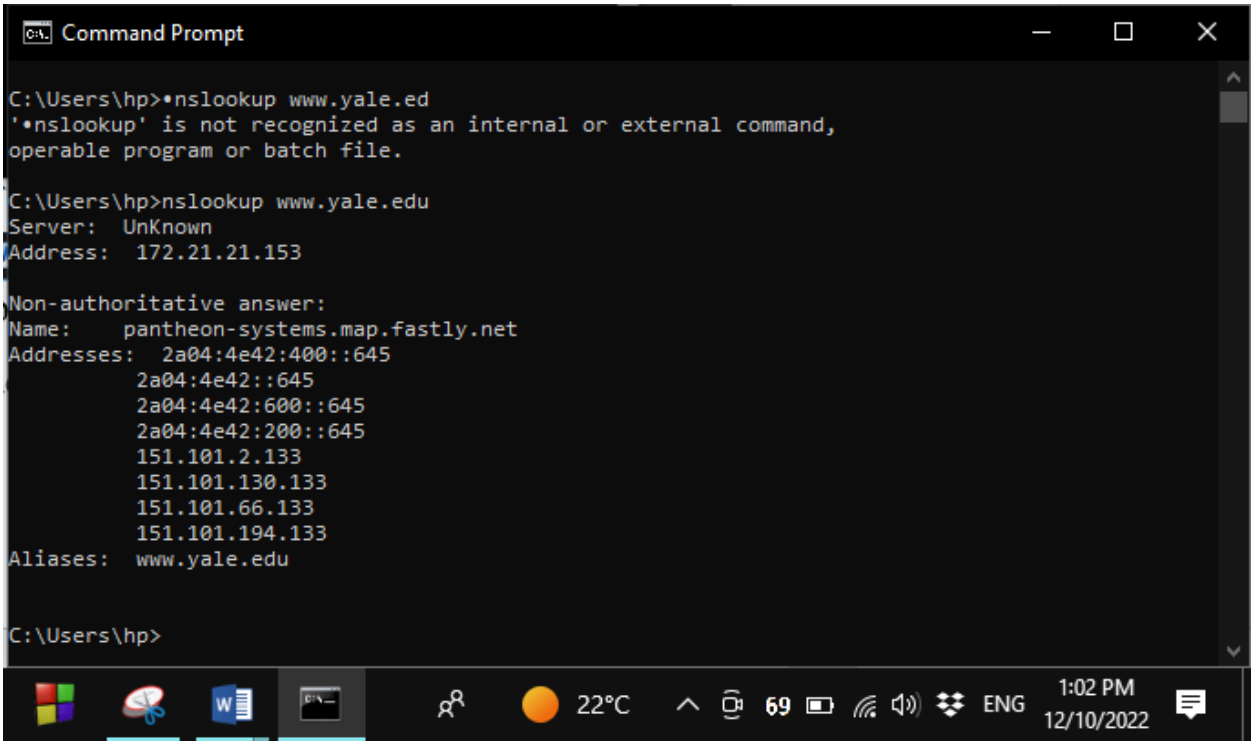


Figure 4: nslookup www.yale.edu

Command of nslookup followed by the domain name displays a record IP address of the domain. It queries to domain name and addresses servers and gets the details. This can be shown in the figure above.

Part 2: Implement a server and client application

Server and client application was implemented both for TCP and for UDP: A client continuously sends the numbers from 0 to 1000,000 to a server listening on port 5566. The server counts the received messages.

The Same Computer

- First we run a python code at the same computer, then we estimate the time that it takes over the following connections

UDP connection

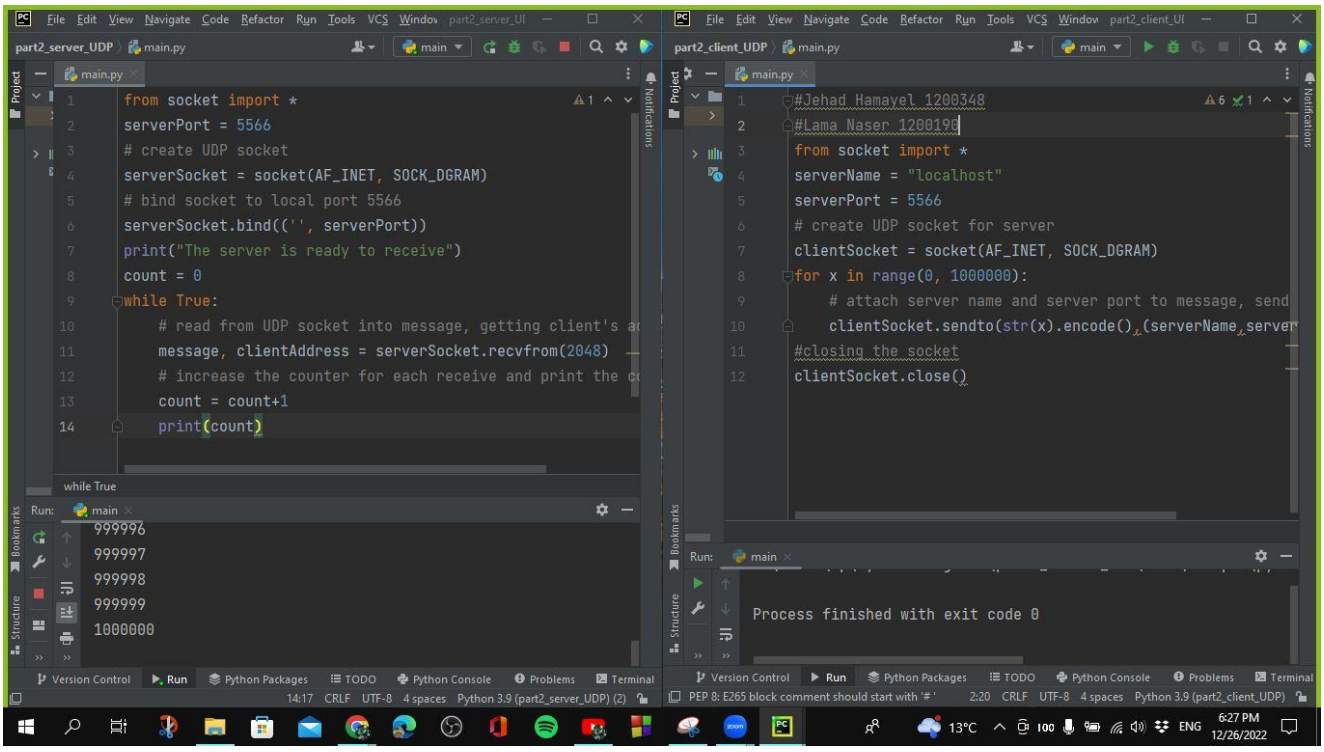


Figure 5: UDP connection on the same computer

The time that this application toke to ends its work is roughly 2 minutes and 34 seconds, without any loss of the data.

TCP connection

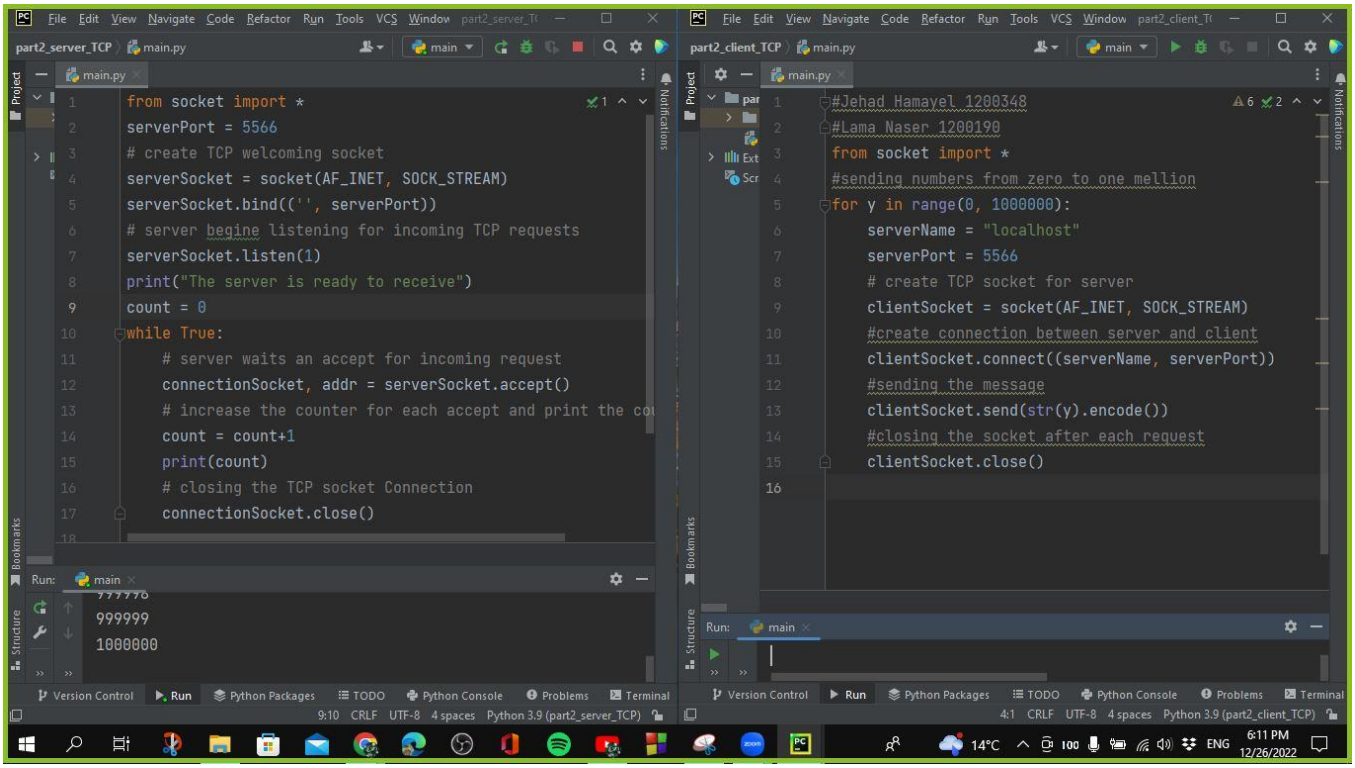


Figure 6: TCP connection on the same computer

The time that this application toke to ends its work is roughly 9 minutes, without any loss of the data.

On different computers connected with a cable

- Second we run a python code at different computers connected with a cable, then we estimate the time that it takes over the following connections

TCP connection

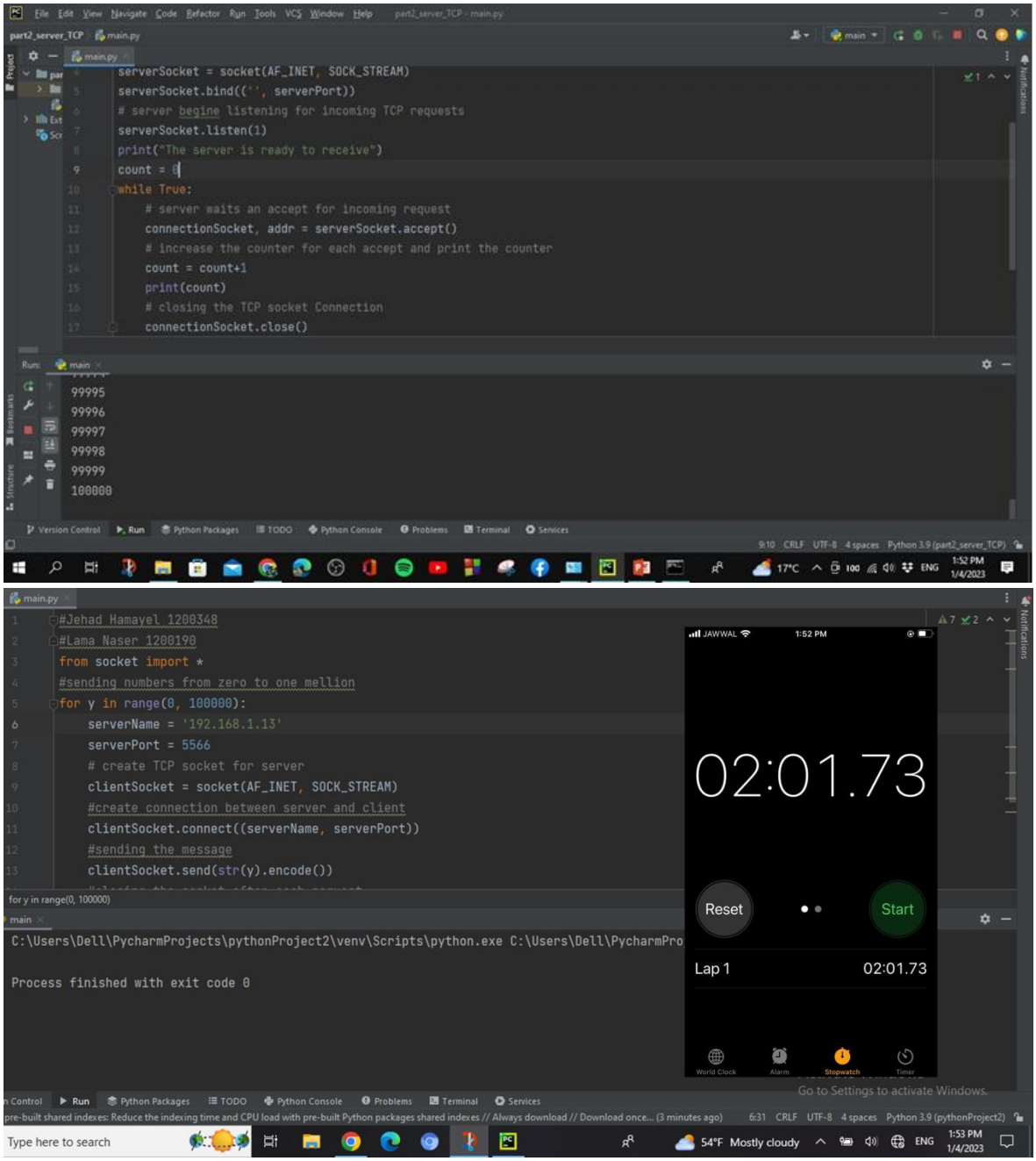


Figure 7: TCP connection on two computers connected through cable

As note in the above pictures for the server and client just 100,000 messages were sent in 2 minutes, we tried to send 1 million messages through the cable but every time the following error was appear will sending-receiving messages.

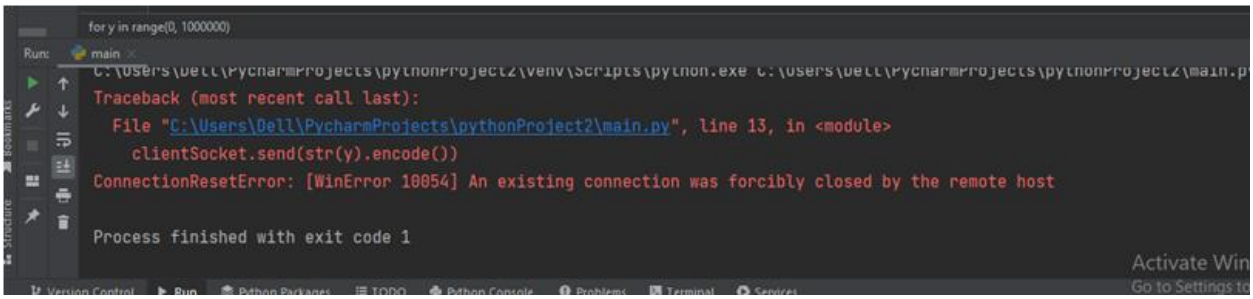


Figure 8: the error message

We searched about this error and we find out that it may be caused by deferent reasons such as: the network link between server and client may be temporarily going down, running out of system resources, or sending malformed data.

UDP connection

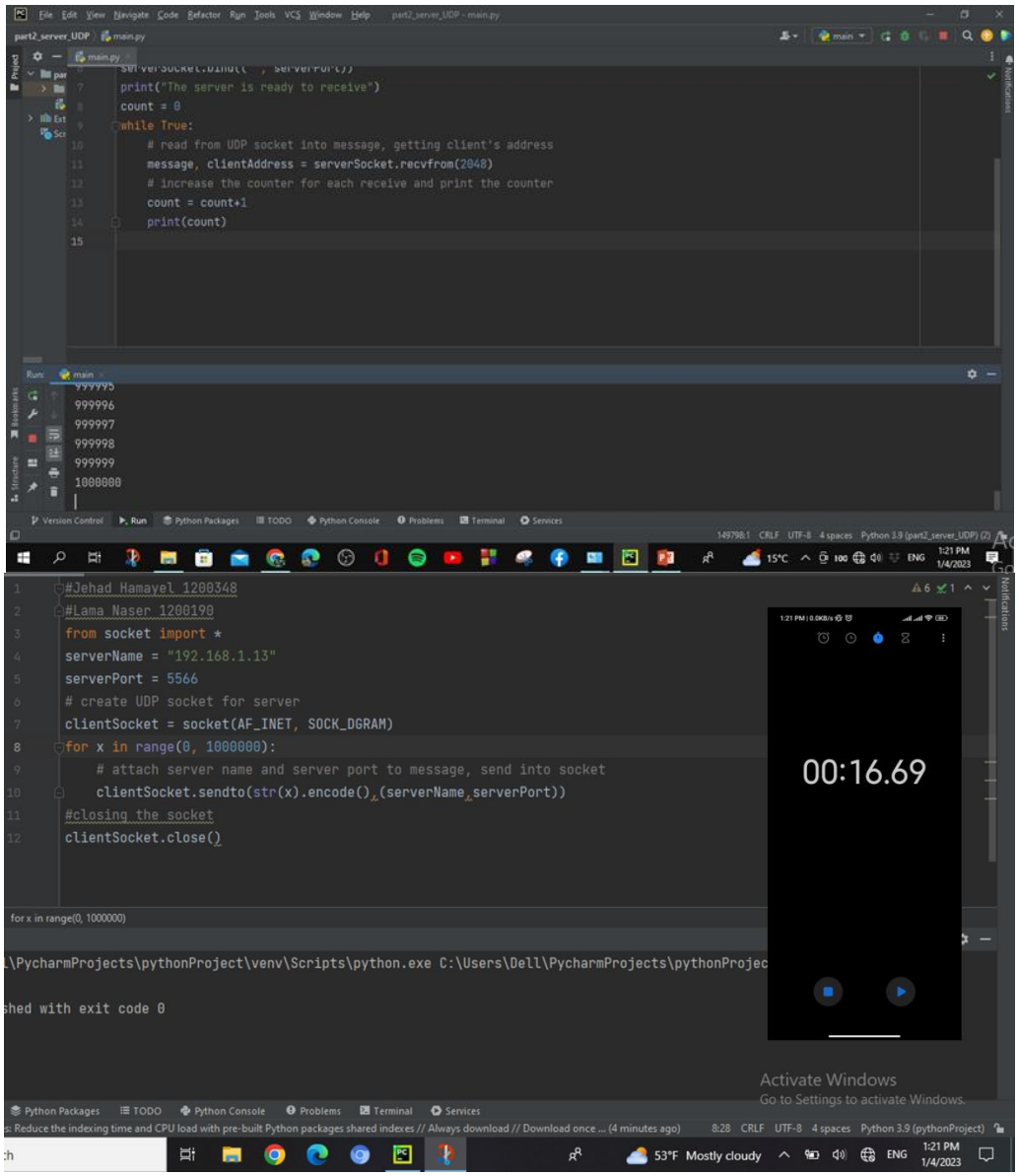


Figure 9: UDP connection on two computers connected through cable without loss

Here 1 million messages were sent in 16 seconds using UDP connection through a cable without any losing in the data.

❖ Also we tried this operation again and we noticed that there was some loss in the data that is because UDP connection does not give a guarantee for reliable data transfer, but on the other hand, it sends the data so fast in compare with TCP.

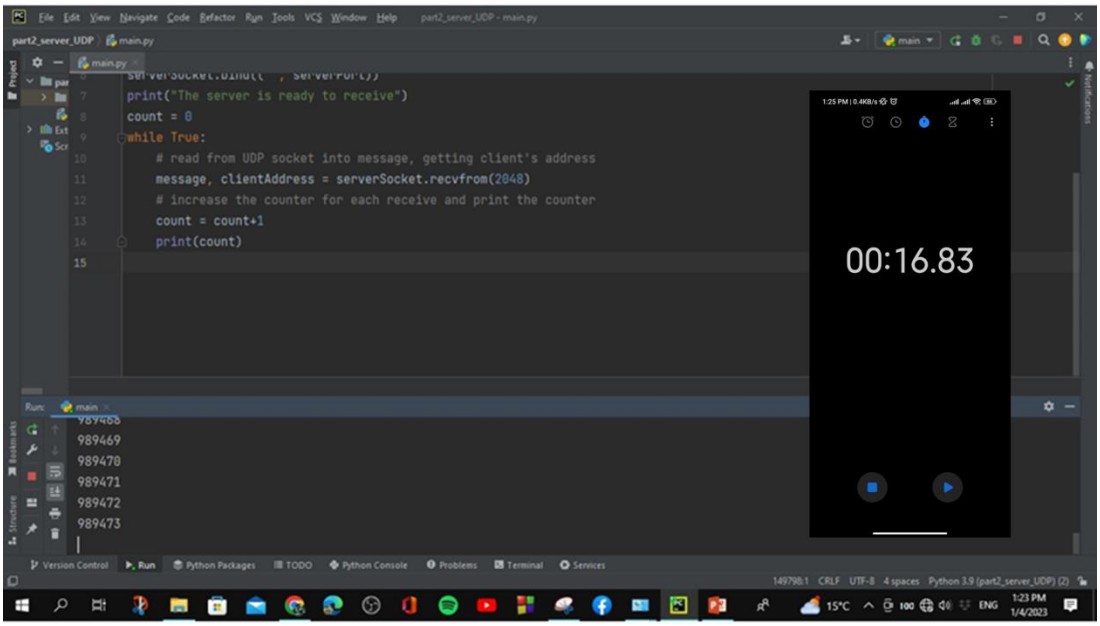


Figure 10: UDP connection on two computers connected through cable with loss

On different computers connected on the same network

- Finally, we run a python code at different computers connected on the same network, here we used the virtual box with Ubuntu operating system to be the client and the origin one as the server, then we estimate the time that it takes over the following connections

TCP connection

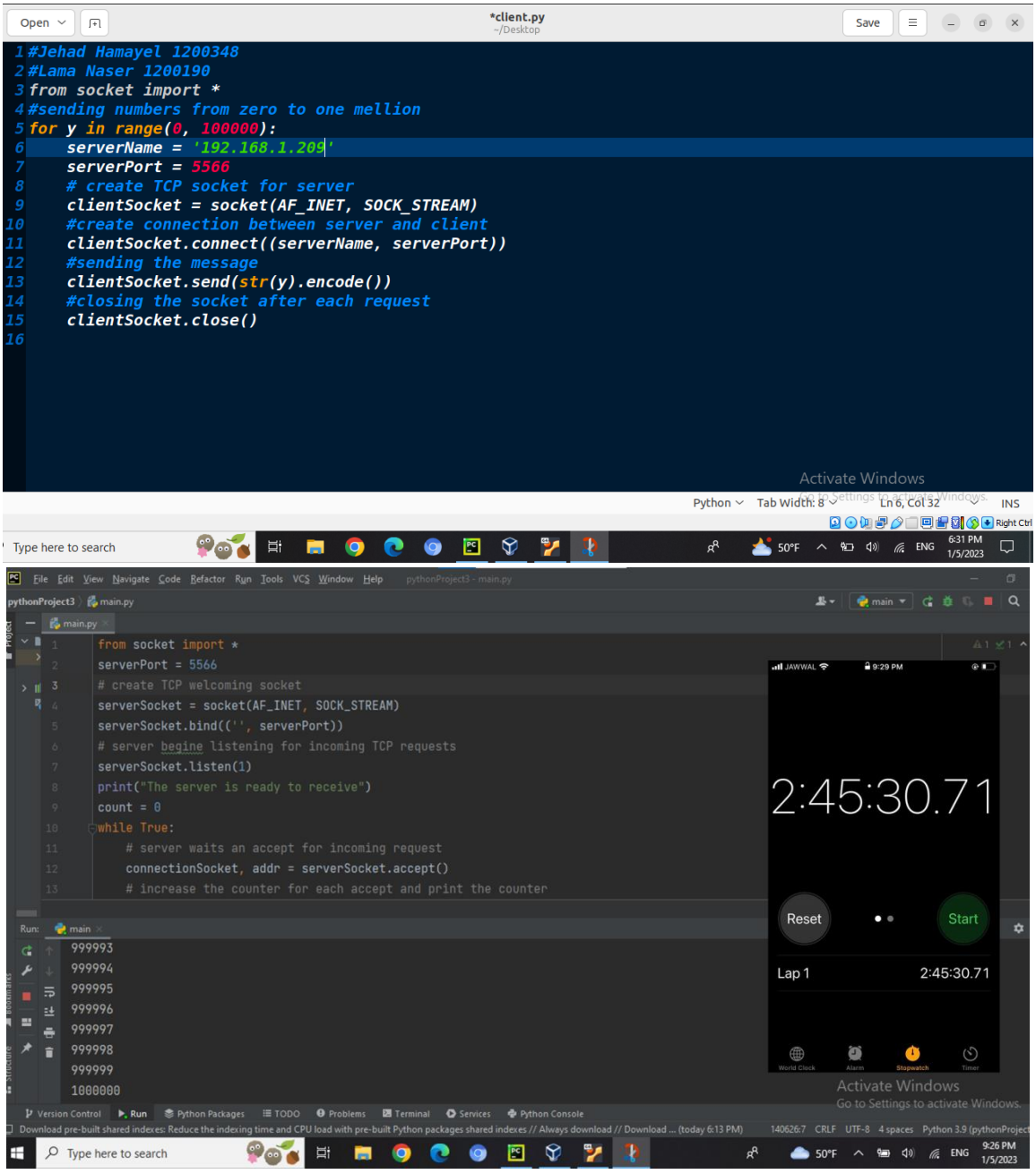


Figure 11: TCP connection on two computers connected through a network

It is very clear that sending data through a network is slower than wired connection that is because the network has many instances where interference and distance can slow down the communication.

UDP connection

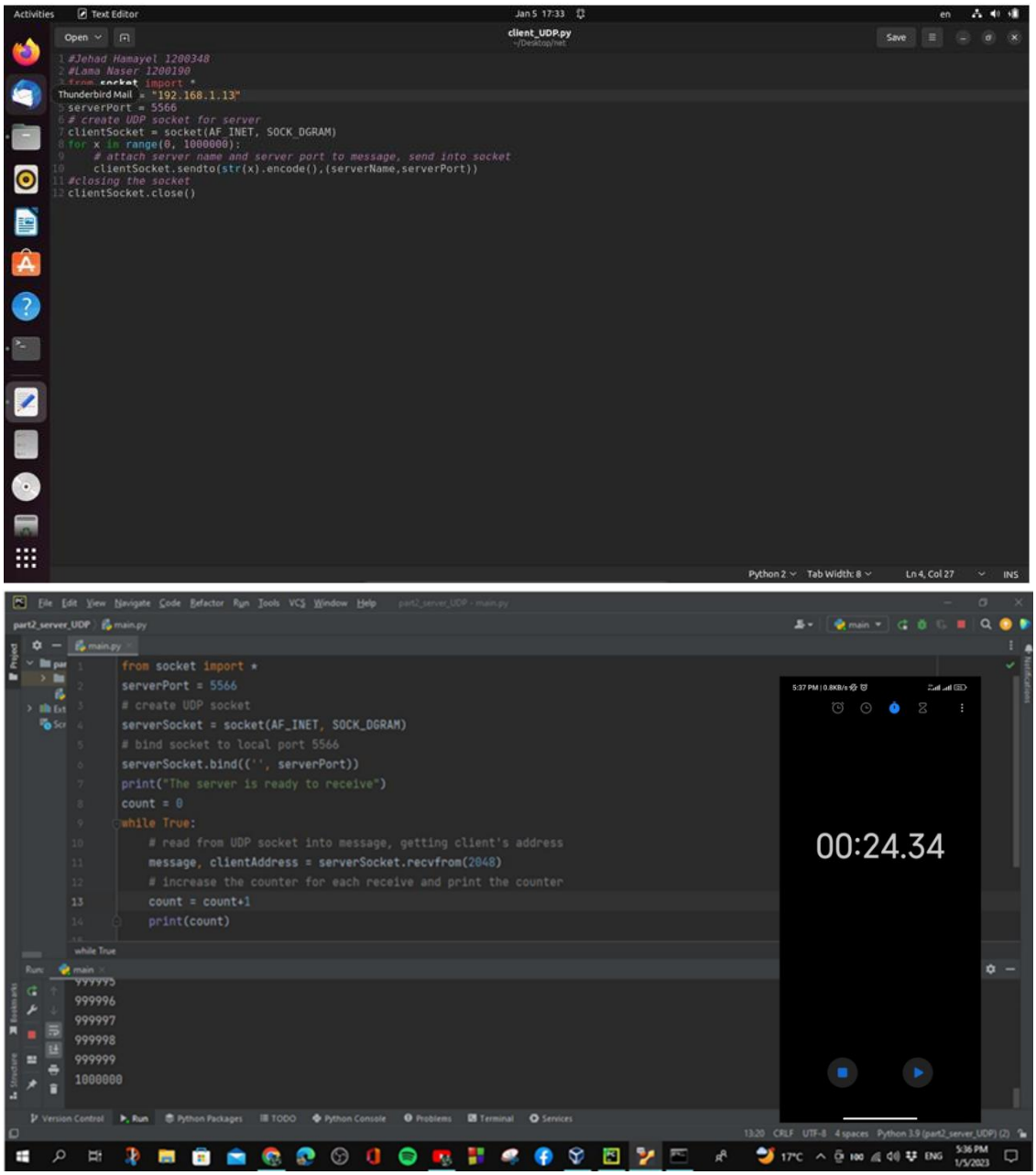


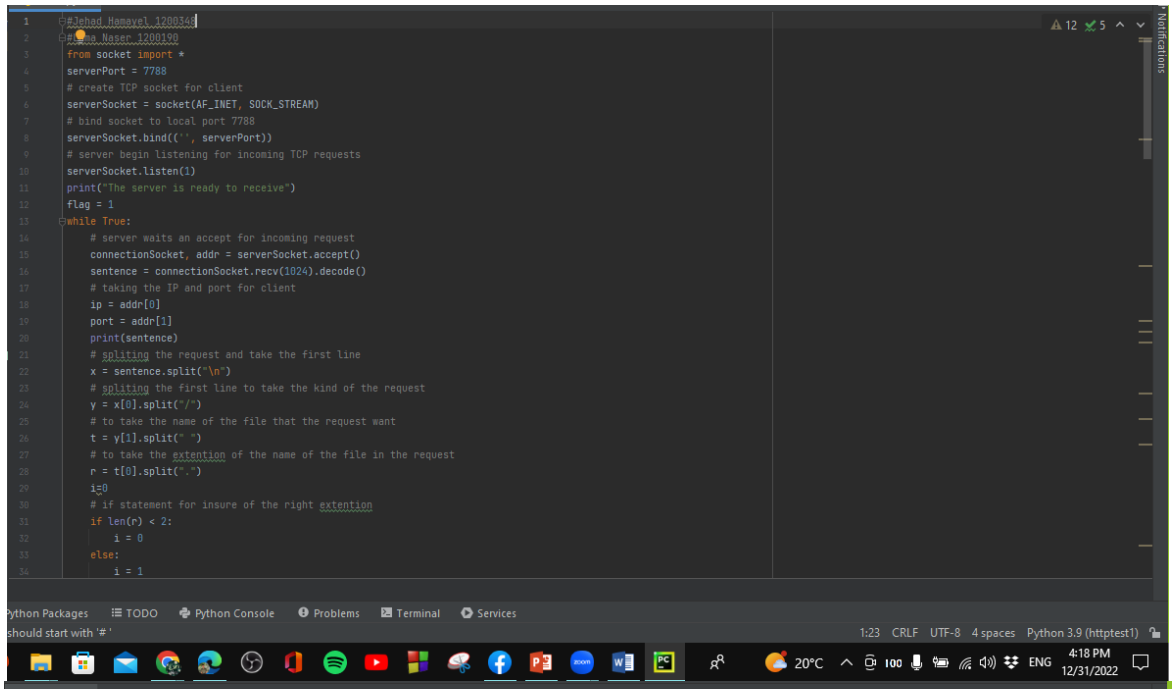
Figure 12: TCP connection on two computers connected through a network

Part 3: Web Server Application.

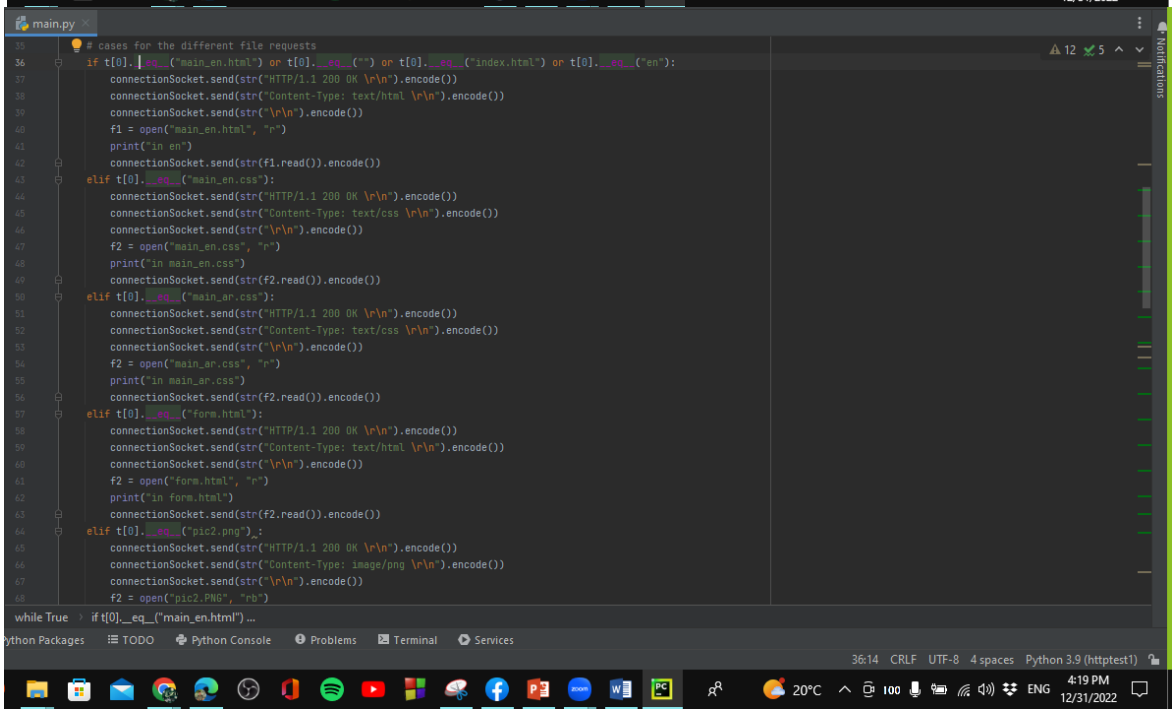
In this part socket programming was used to implement a simple but a complete web server listening on port 7788 in python language.

Our Code:

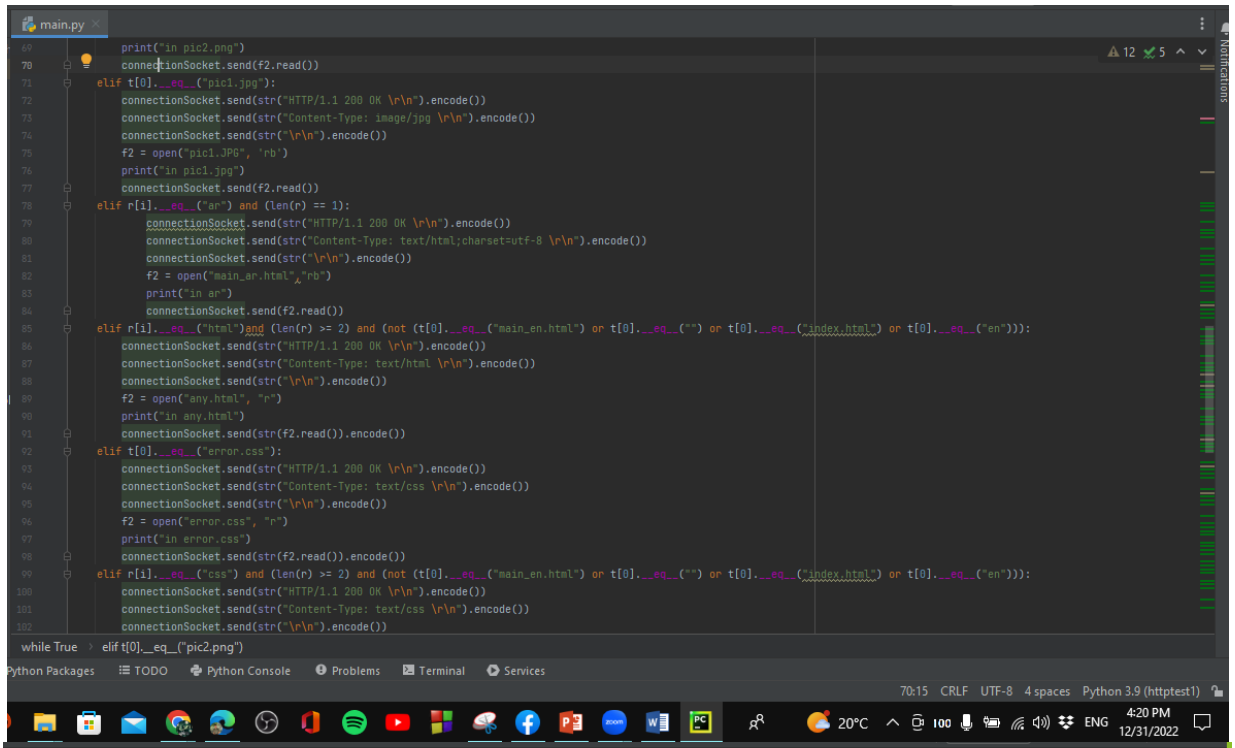
Main.py code:



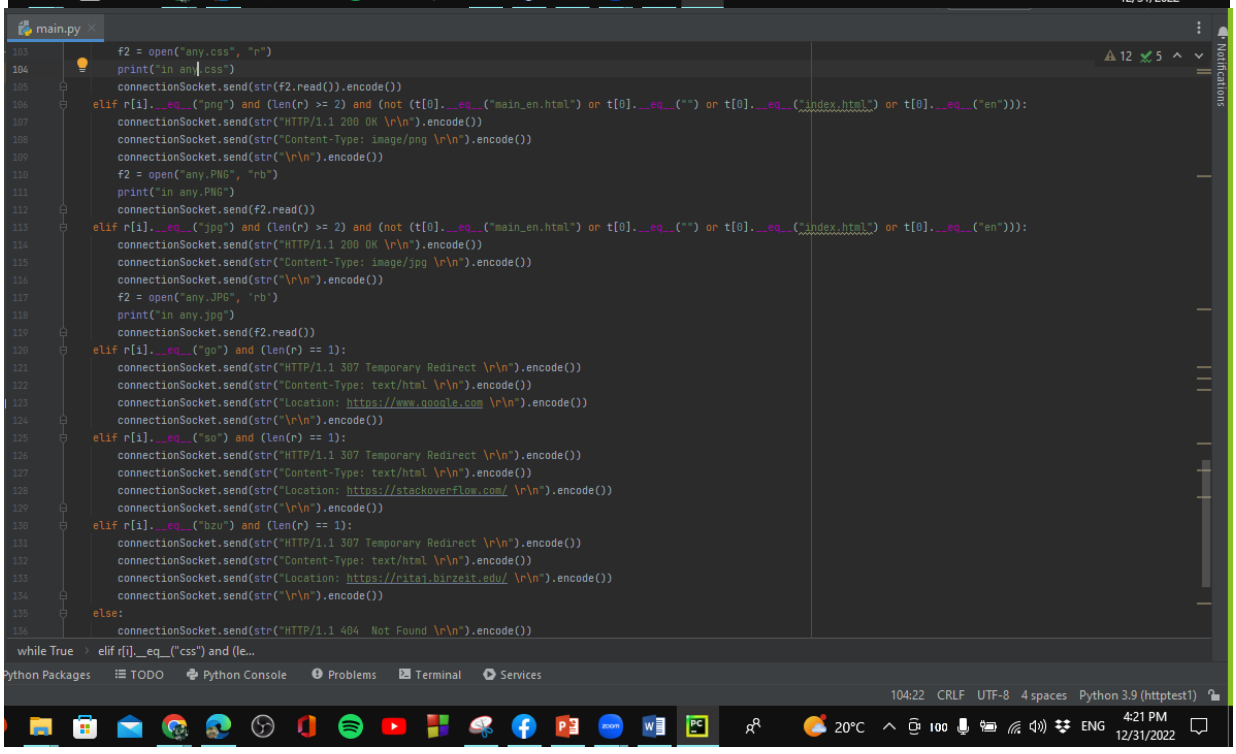
```
#Jehad Hamayel 1200340
#Rana Naser 1200190
from socket import *
serverPort = 7788
# create TCP socket for client
serverSocket = socket(AF_INET, SOCK_STREAM)
# bind socket to local port 7788
serverSocket.bind(('', serverPort))
# server begin listening for incoming TCP requests
serverSocket.listen(1)
print("The server is ready to receive")
flag = 1
while True:
    # server waits an accept for incoming request
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    # taking the IP and port for client
    ip = addr[0]
    port = addr[1]
    print(sentence)
    # spliting the request and take the first line
    x = sentence.split("\n")
    # spliting the first line to take the kind of the request
    y = x[0].split("/")
    # to take the name of the file that the request want
    t = y[1].split(" ")
    # to take the extention of the name of the file in the request
    r = t[0].split(".")
    i=0
    # if statement for insure of the right extention
    if len(r) < 2:
        i = 0
    else:
        i = 1
```



```
# cases for the different file requests
if t[0].__eq__("main_en.html") or t[0].__eq__("") or t[0].__eq__("index.html") or t[0].__eq__("en"):
    connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
    connectionSocket.send(str("Content-Type: text/html \r\n").encode())
    connectionSocket.send(str("\r\n").encode())
    f1 = open("main_en.html", "r")
    print("in en")
    connectionSocket.send(str(f1.read()).encode())
elif t[0].__eq__("main_en.css"):
    connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
    connectionSocket.send(str("Content-Type: text/css \r\n").encode())
    connectionSocket.send(str("\r\n").encode())
    f2 = open("main_en.css", "r")
    print("in main_en.css")
    connectionSocket.send(str(f2.read()).encode())
elif t[0].__eq__("main_ar.css"):
    connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
    connectionSocket.send(str("Content-Type: text/css \r\n").encode())
    connectionSocket.send(str("\r\n").encode())
    f2 = open("main_ar.css", "r")
    print("in main_ar.css")
    connectionSocket.send(str(f2.read()).encode())
elif t[0].__eq__("form.html"):
    connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
    connectionSocket.send(str("Content-Type: text/html \r\n").encode())
    connectionSocket.send(str("\r\n").encode())
    f2 = open("form.html", "r")
    print("in form.html")
    connectionSocket.send(str(f2.read()).encode())
elif t[0].__eq__("pic2.png"):
    connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
    connectionSocket.send(str("Content-Type: image/png \r\n").encode())
    connectionSocket.send(str("\r\n").encode())
    f2 = open("pic2.PNG", "rb")
```



```
        print("in pic2.png")
        connectionSocket.send(f2.read())
    elif t[0].__eq__("pic1.jpg"):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: image/jpg \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
        f2 = open("pic1.JPG", 'rb')
        print("in pic1.jpg")
        connectionSocket.send(f2.read())
    elif r[i].__eq__("ar") and (len(r) == 1):
            connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
            connectionSocket.send(str("Content-Type: text/html;charset=utf-8 \r\n").encode())
            connectionSocket.send(str("\r\n").encode())
            f2 = open("main_ar.html","rb")
            print("in ar")
            connectionSocket.send(f2.read())
    elif r[i].__eq__("html")and (len(r) >= 2) and (not (t[0].__eq__("main_en.html") or t[0].__eq__("") or t[0].__eq__("index.html") or t[0].__eq__("en"))):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: text/html \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
        f2 = open("any.html", "r")
        print("in any.html")
        connectionSocket.send(str(f2.read()).encode())
    elif t[0].__eq__("error.css"):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: text/css \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
        f2 = open("error.css", "r")
        print("in error.css")
        connectionSocket.send(str(f2.read()).encode())
    elif r[i].__eq__("css") and (len(r) >= 2) and (not (t[0].__eq__("main_en.html") or t[0].__eq__("") or t[0].__eq__("index.html") or t[0].__eq__("en"))):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: text/css \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
```



```
        f2 = open("any.css", "r")
        print("in any.css")
        connectionSocket.send(str(f2.read()).encode())
    elif r[i].__eq__("png") and (len(r) >= 2) and (not (t[0].__eq__("main_en.html") or t[0].__eq__("") or t[0].__eq__("index.html") or t[0].__eq__("en"))):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: image/png \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
        f2 = open("any.PNG", "rb")
        print("in any.PNG")
        connectionSocket.send(f2.read())
    elif r[i].__eq__("jpg") and (len(r) >= 2) and (not (t[0].__eq__("main_en.html") or t[0].__eq__("") or t[0].__eq__("index.html") or t[0].__eq__("en"))):
        connectionSocket.send(str("HTTP/1.1 200 OK \r\n").encode())
        connectionSocket.send(str("Content-Type: image/jpg \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
        f2 = open("any.JPG", 'rb')
        print("in any.jpg")
        connectionSocket.send(f2.read())
    elif r[i].__eq__("go") and (len(r) == 1):
        connectionSocket.send(str("HTTP/1.1 307 Temporary Redirect \r\n").encode())
        connectionSocket.send(str("Content-Type: text/html \r\n").encode())
        connectionSocket.send(str("Location: https://www.google.com \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
    elif r[i].__eq__("so") and (len(r) == 1):
        connectionSocket.send(str("HTTP/1.1 307 Temporary Redirect \r\n").encode())
        connectionSocket.send(str("Content-Type: text/html \r\n").encode())
        connectionSocket.send(str("Location: https://stackoverflow.com/ \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
    elif r[i].__eq__("bzu") and (len(r) == 1):
        connectionSocket.send(str("HTTP/1.1 307 Temporary Redirect \r\n").encode())
        connectionSocket.send(str("Content-Type: text/html \r\n").encode())
        connectionSocket.send(str("Location: https://ritaj.birzeit.edu/ \r\n").encode())
        connectionSocket.send(str("\r\n").encode())
    else:
        connectionSocket.send(str("HTTP/1.1 404  Not Found \r\n").encode())
```

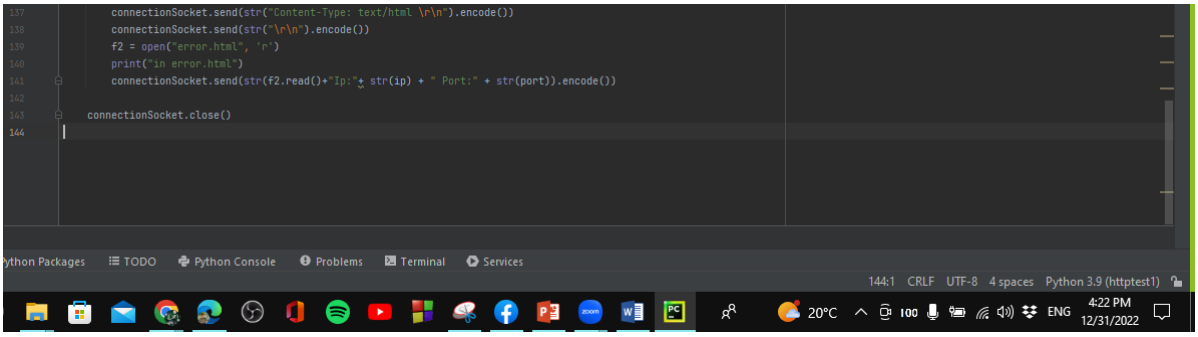


Figure 13:Our Code in python For web Server

Testing the application on the same computer

- if the request is / or /index.html or /main_en.html or /en (for example localhost:7788/Or localhost:7788/en) then the server sends main_en.html file with Content-Type: text/html as shown in the figure below.

on command line (en):

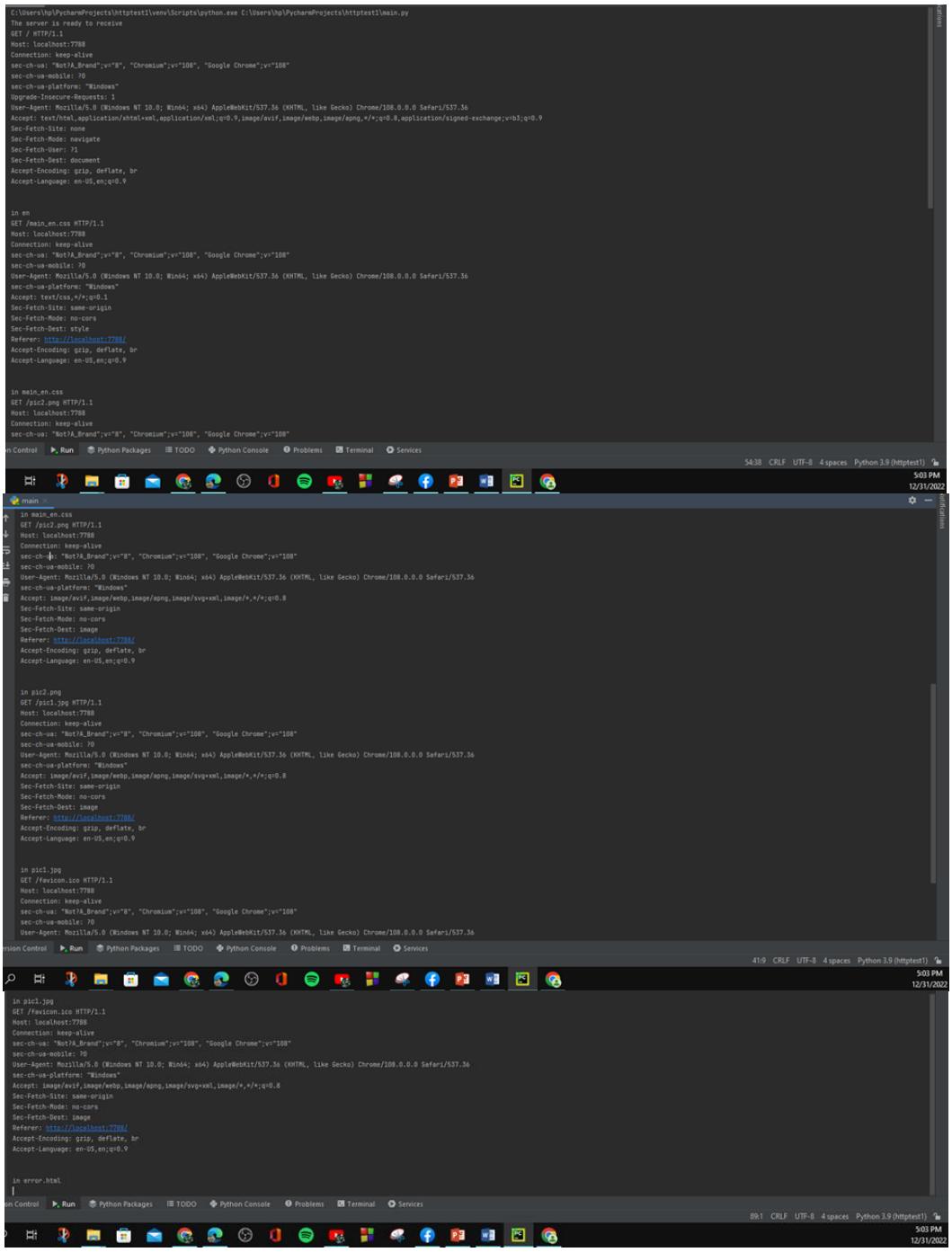


Figure 14:the request messages for / or /index.html or /main_en.html or /en

on the browser(en):

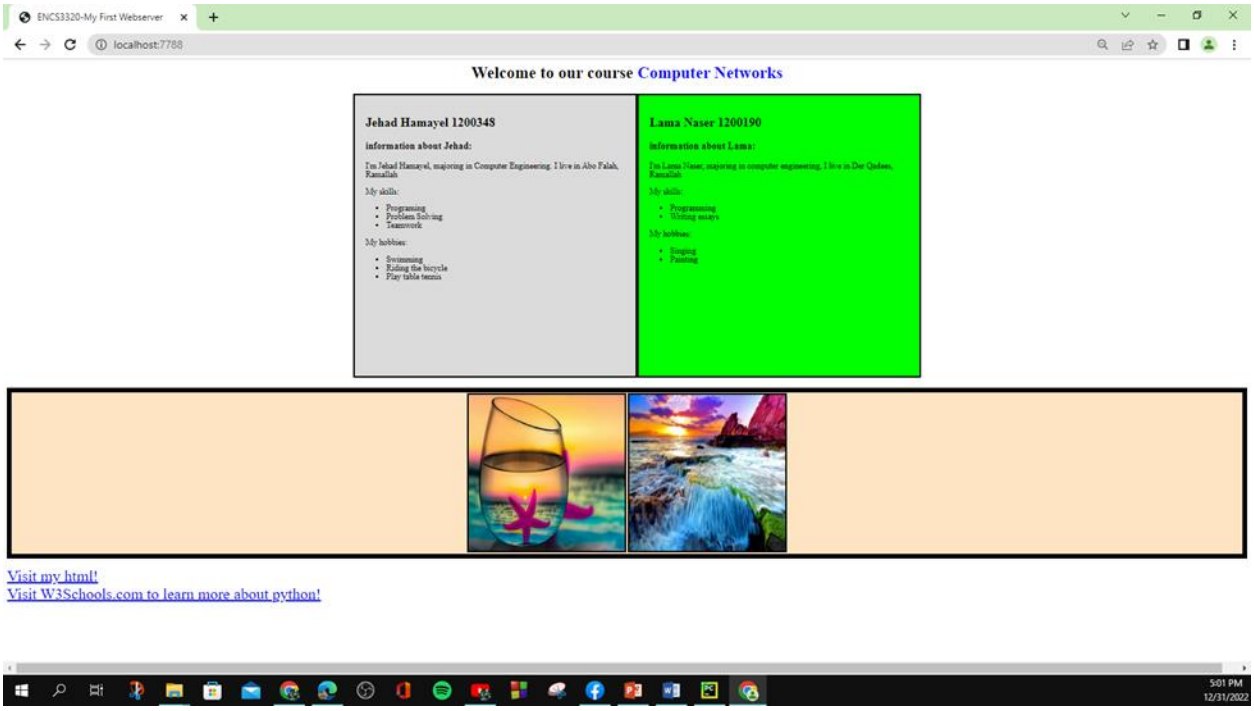
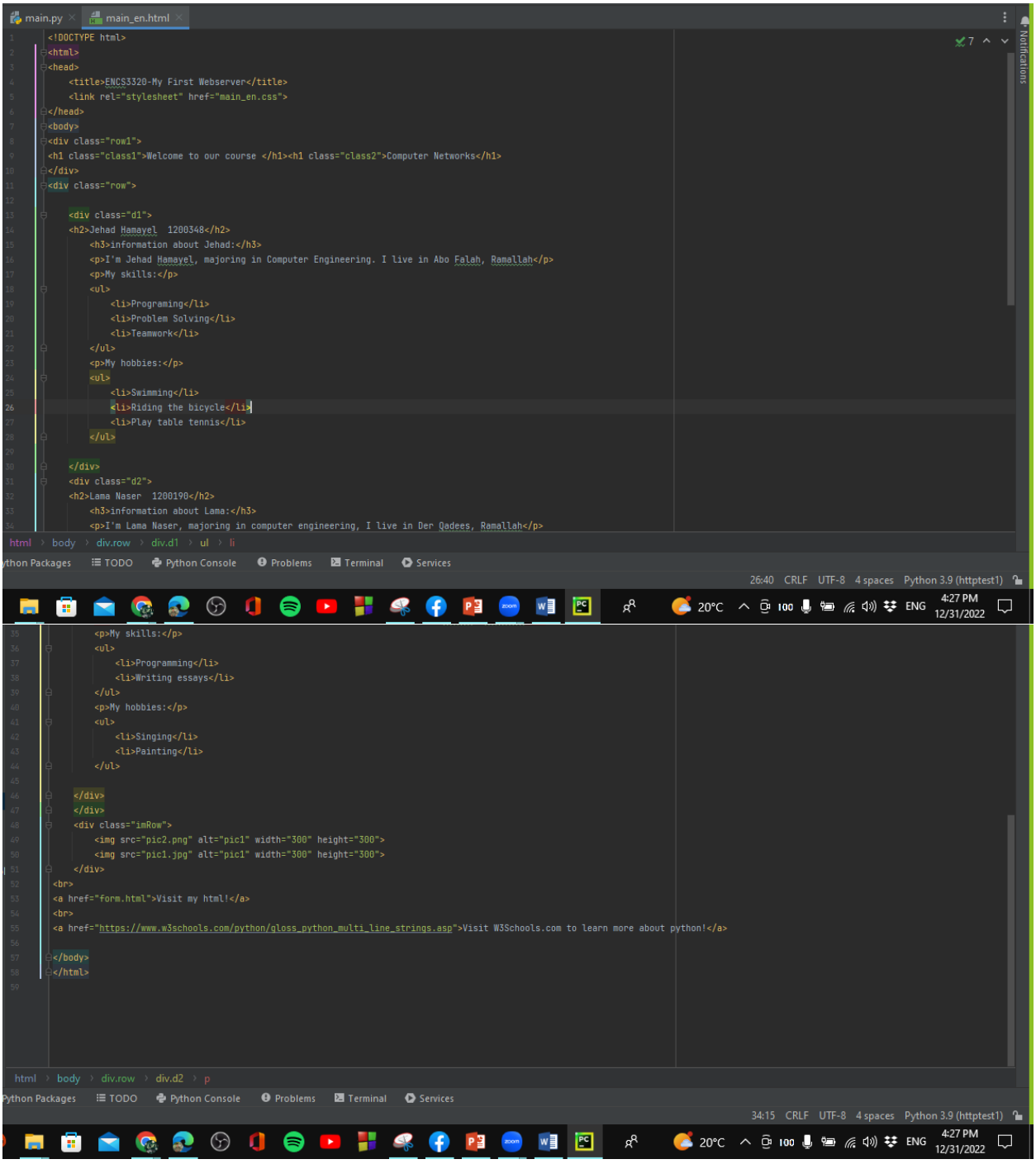


Figure 15: the result of the request in the web page in English

For the HTML code, “ENCS3320-My First Webserver” in the title, “Welcome to our course computer networks” and our names and IDs were printed on the page of our website. And some information about us. This is the page of the English version of the website. All these texts were styled in the CSS code.

main_en.html code:



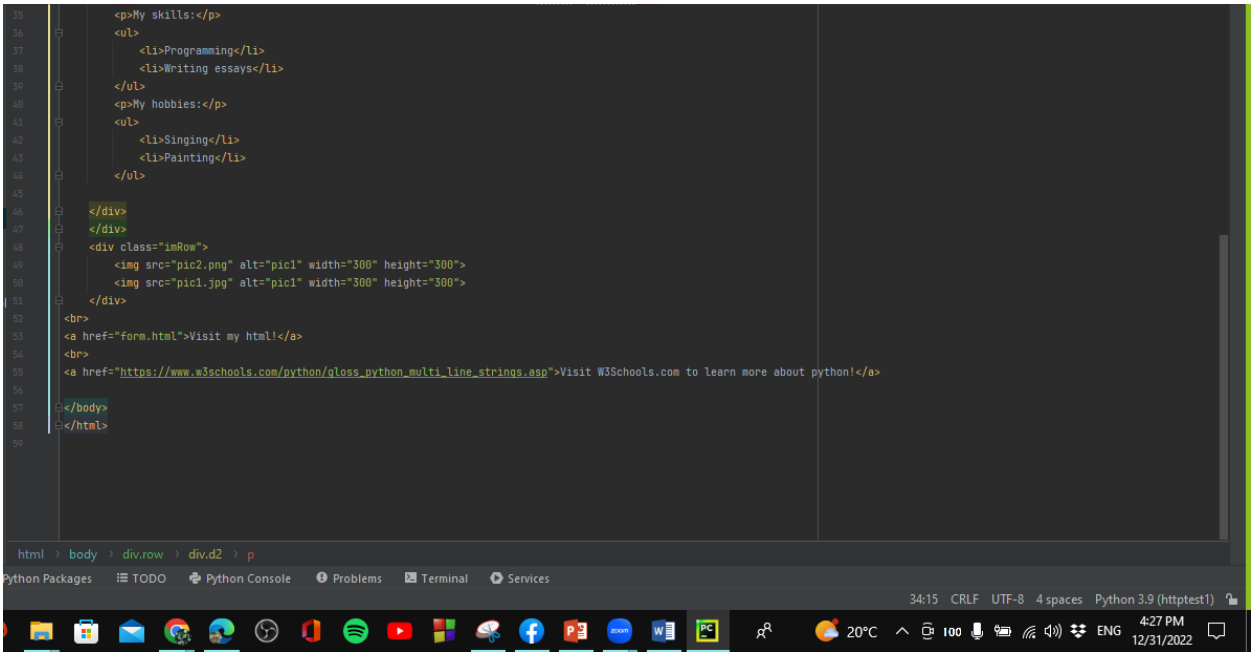


Figure 16: main_en.html code

main_en.css code:

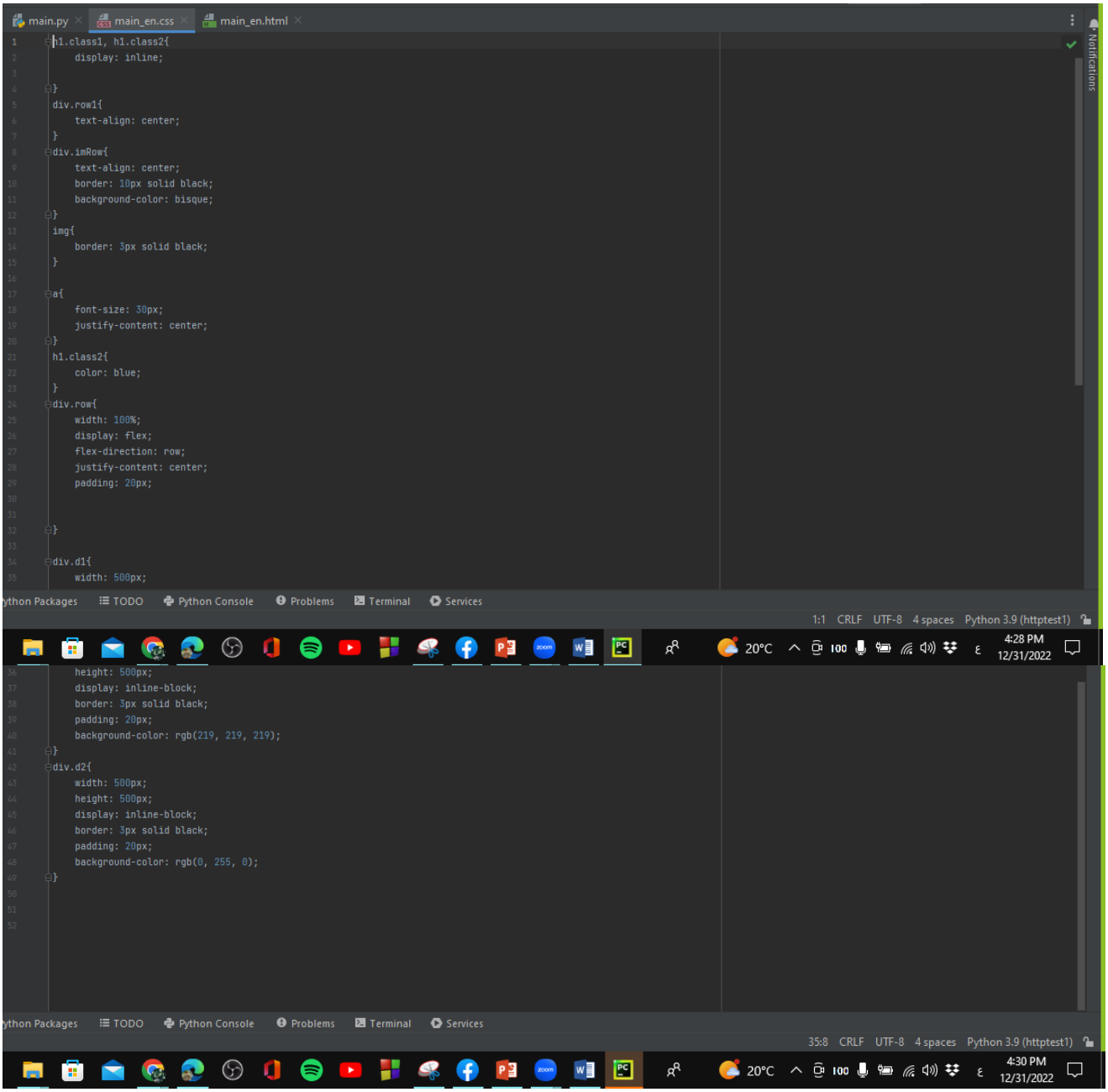


Figure 17: main_en.css code

When “visit my html” clicked

When “visit my html” clicked, then a local html file will appear.

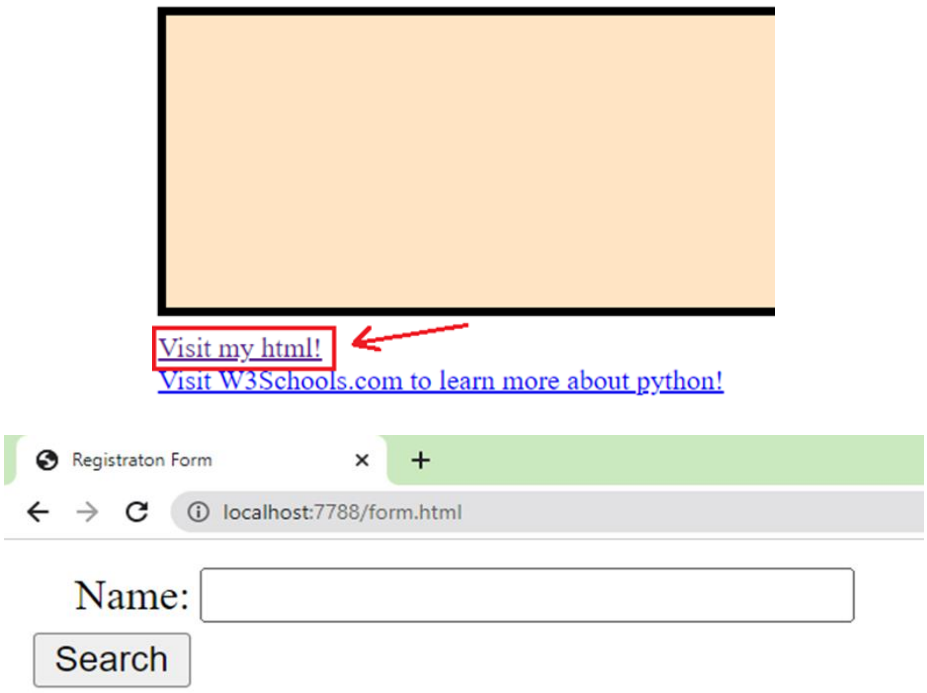


Figure 18:visit my html

The request message on the command line:

```
GET /form.html HTTP/1.1
Host: localhost:7788
Connection: keep-alive
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:7788/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9


in form.html
```

Figure 19:request message

form.html code (local html file):

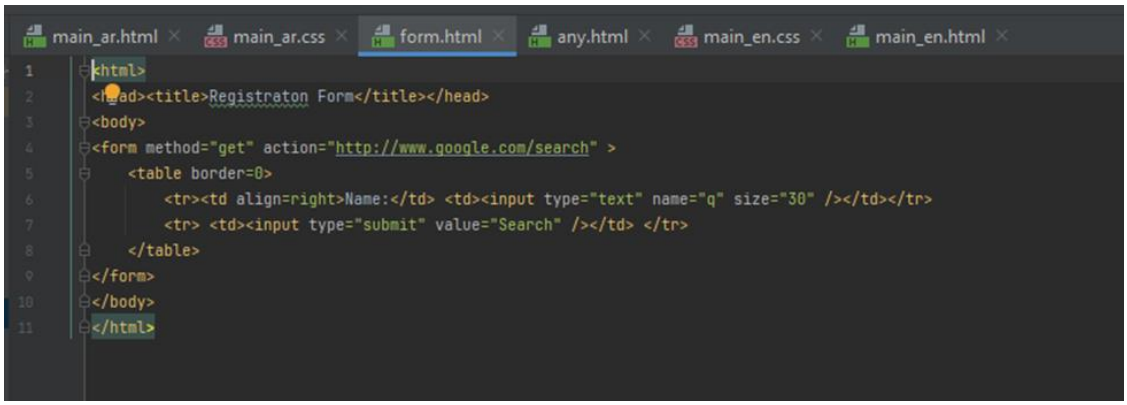
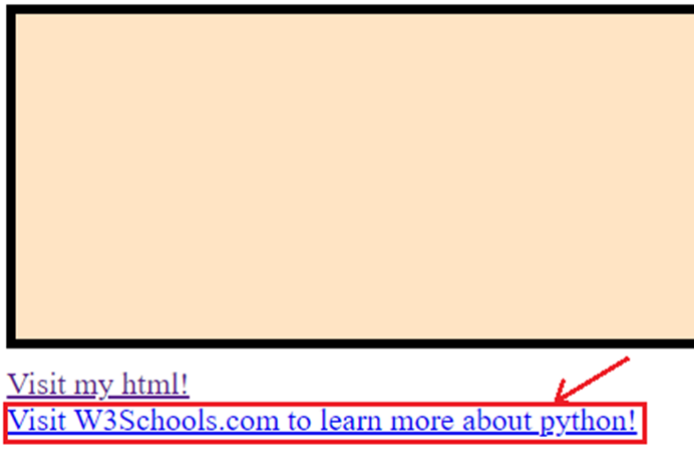


Figure 20:form.html code

3Wschools link is clicked

If this link is clicked then the following page will appear



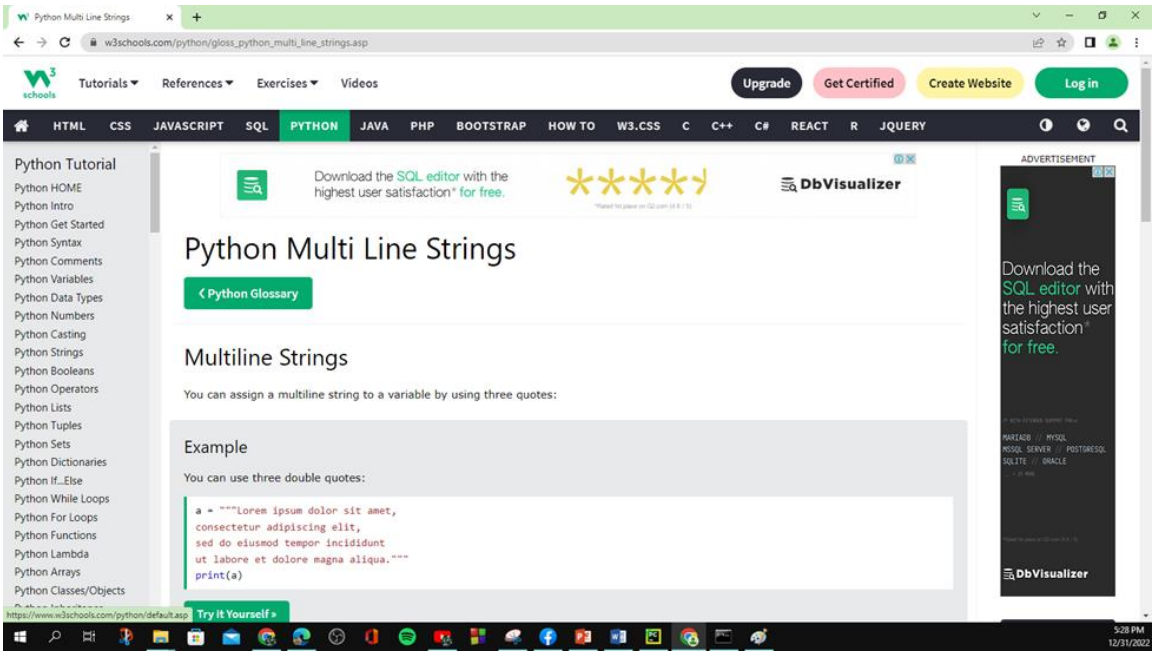


Figure 21:w3Schools Page

- If the request is /ar then the server response with main_ar.html which is an Arabic version of main_en.html

on command line(ar):

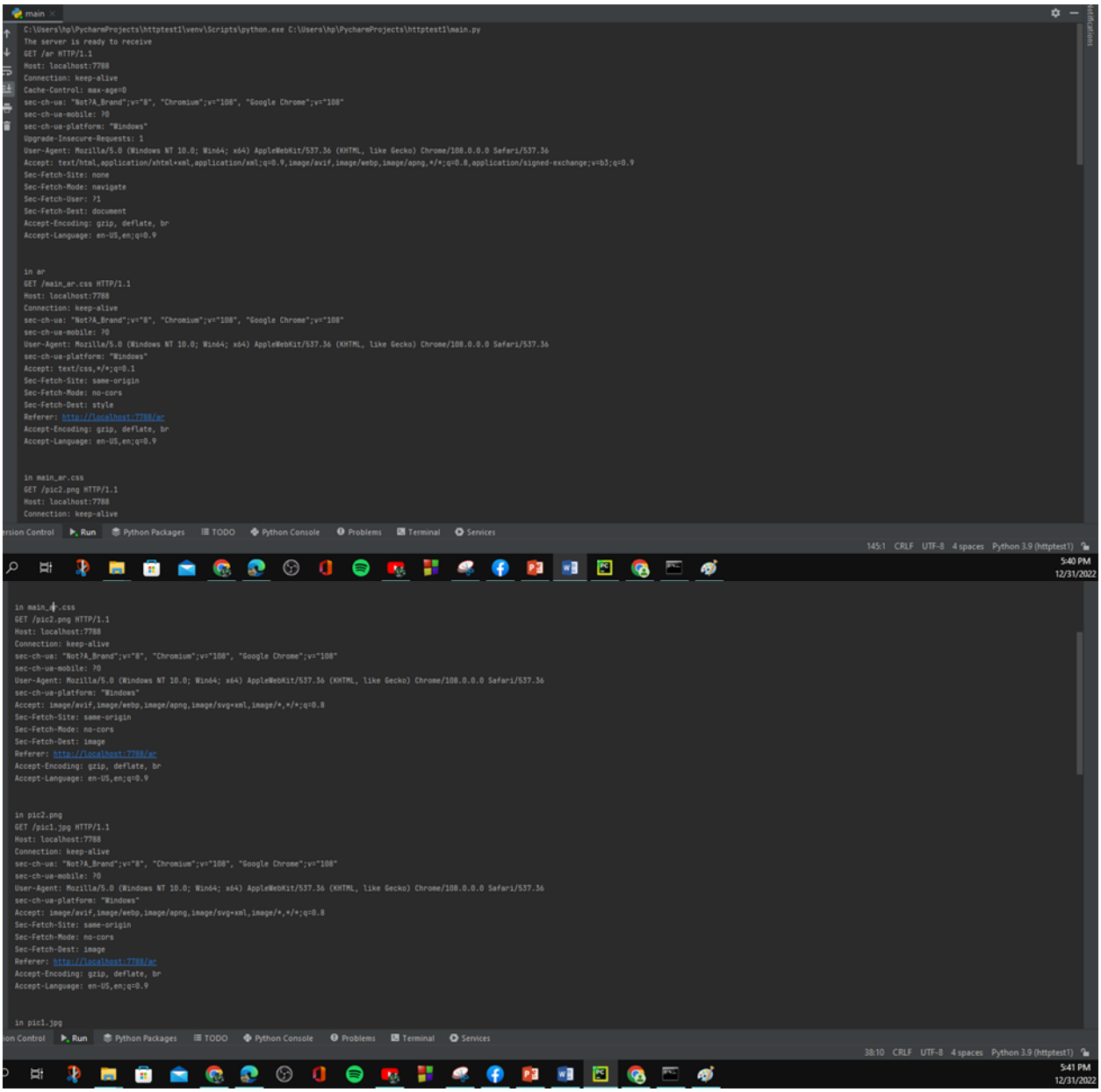


Figure 22: the request message when we enter /ar

on the browser(ar):

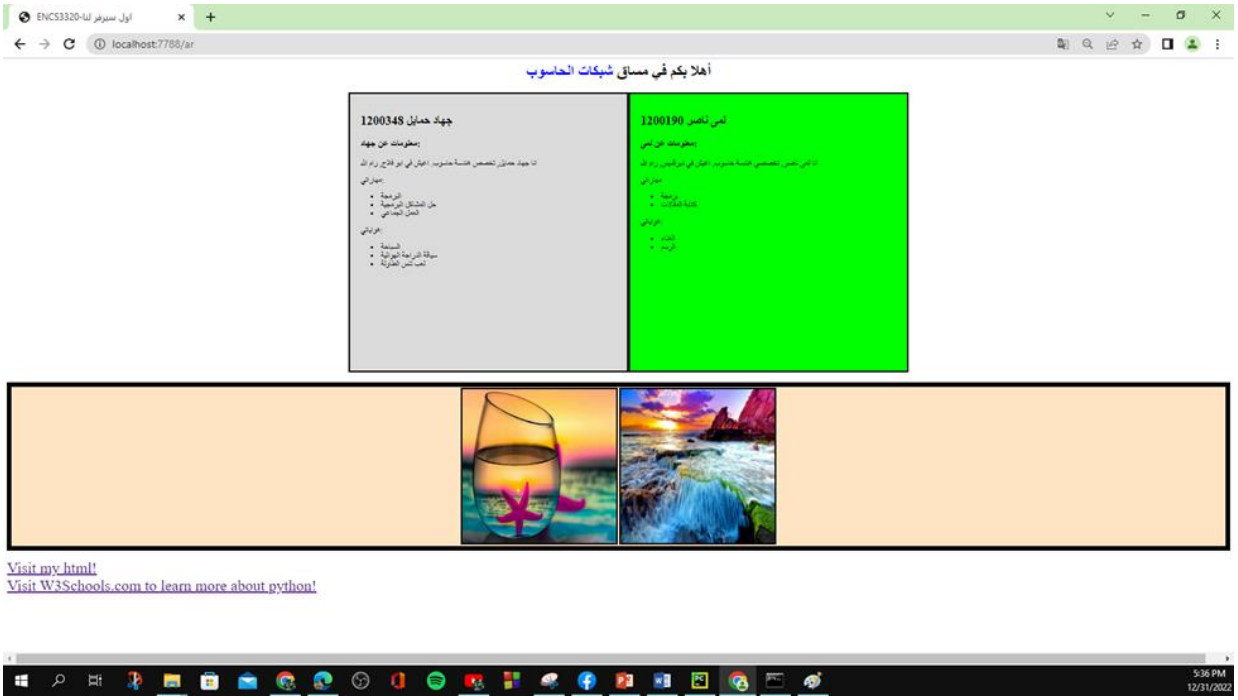
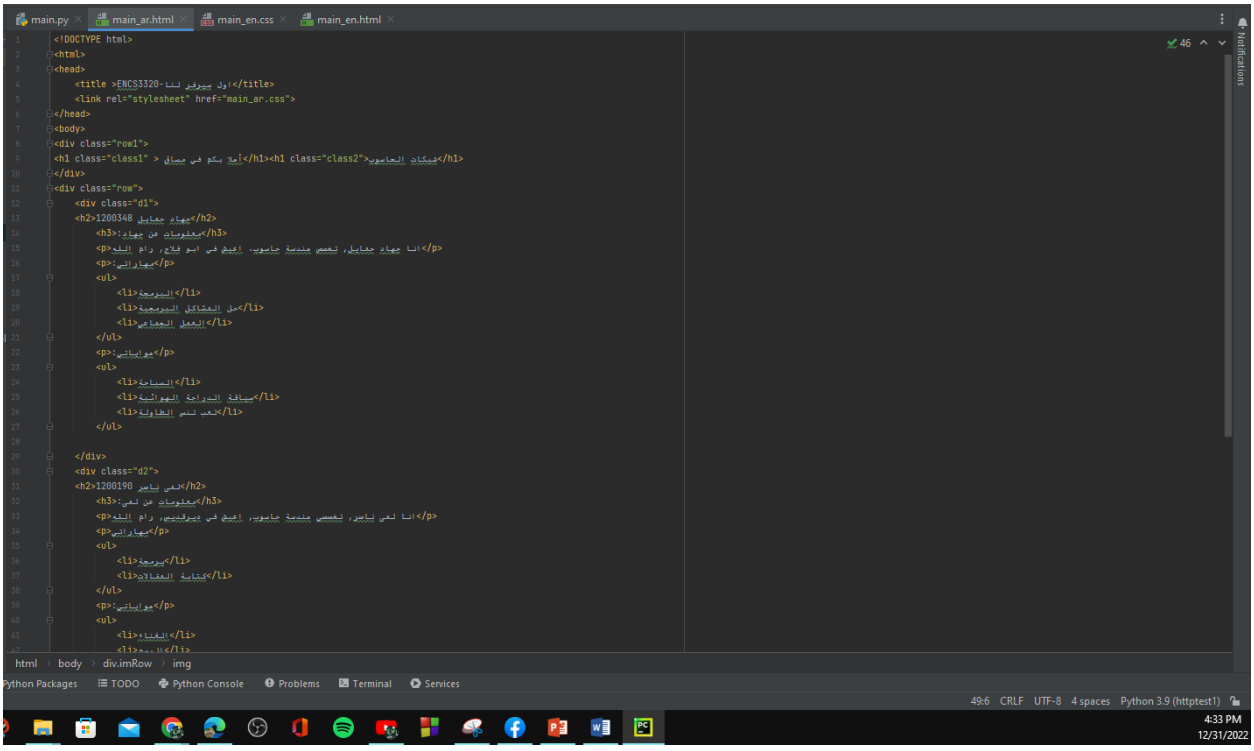


Figure 23:the result of the request in the web page in Arabic

main_ar.html code:



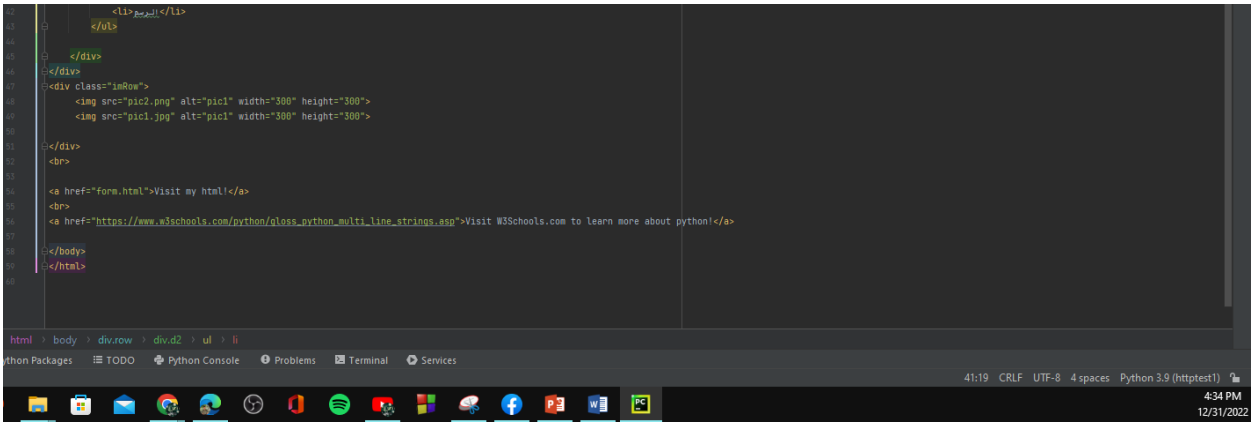
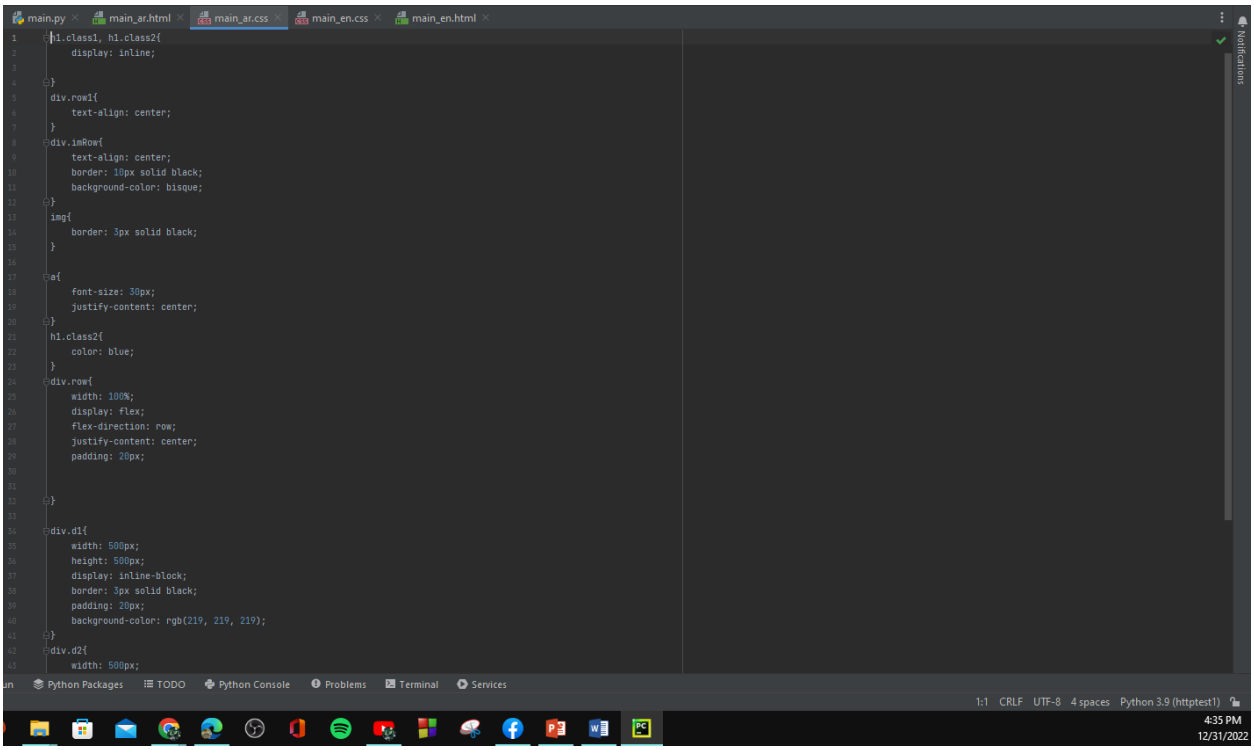


Figure 24: main_ar.html code

main_ar.css code:



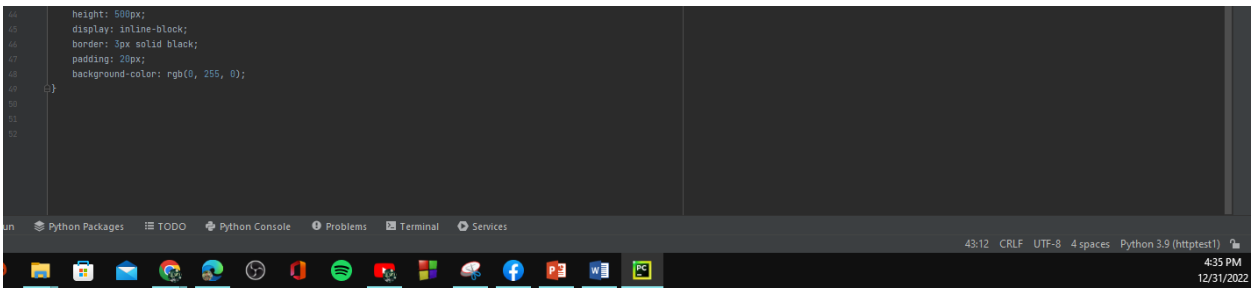


Figure 25: main_ar.css code

- if the request is an .html file then the server sends the a.html file with Content-Type: text/html.

on the command line(any html file):

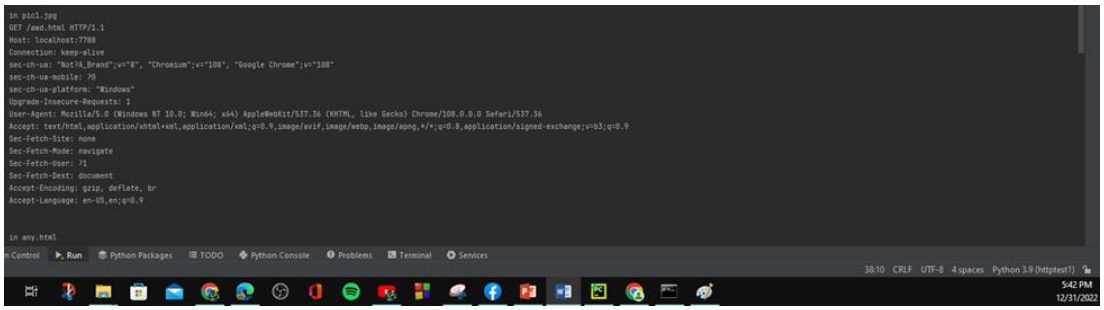


Figure 26: any html file request

on the browser(any html file):

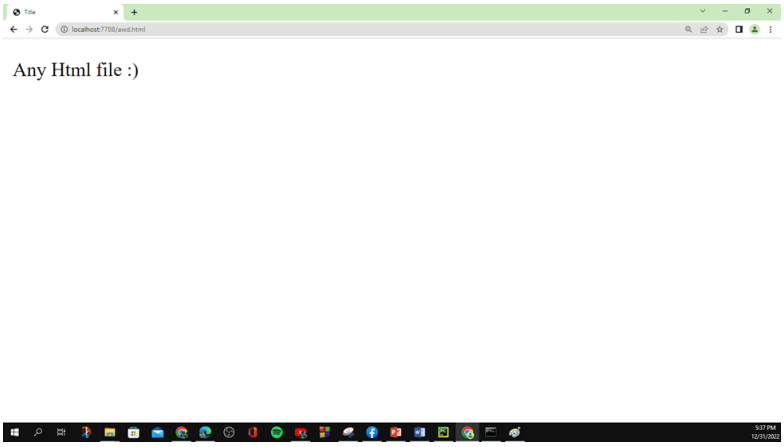


Figure 27: the any html file on website

a .html code:

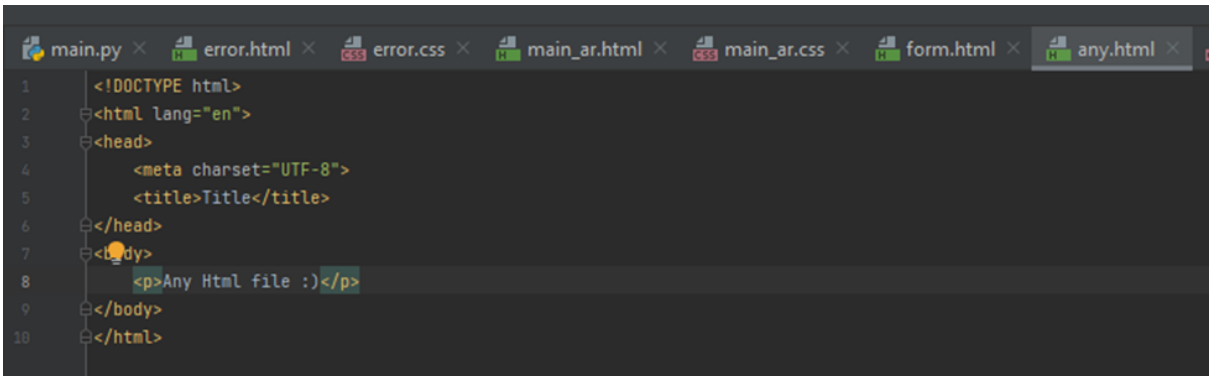


Figure 28:any.html file

- if the request is a .css file then the server sends the a.css file with Content-Type: text/css

on the command line(any css file):

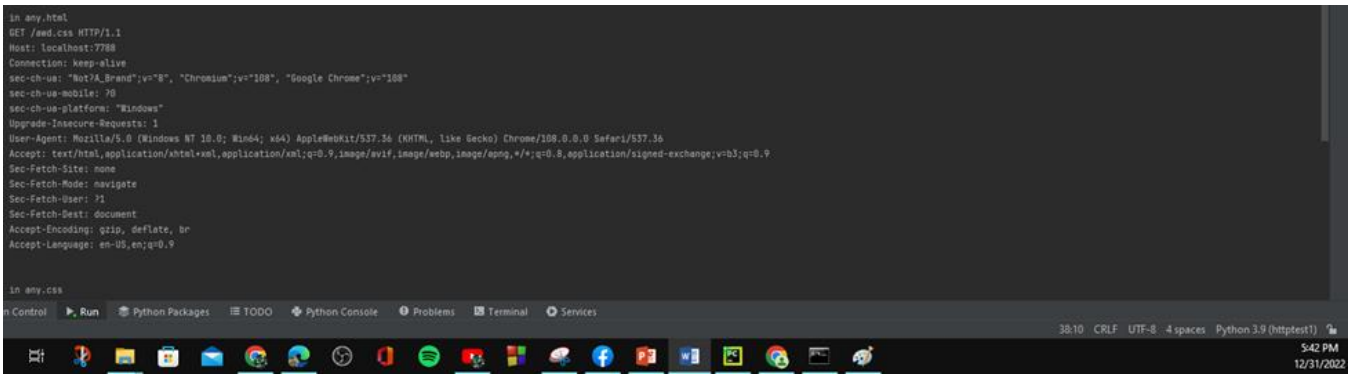


Figure 29: the request message for a css file

on the browser(any css file):

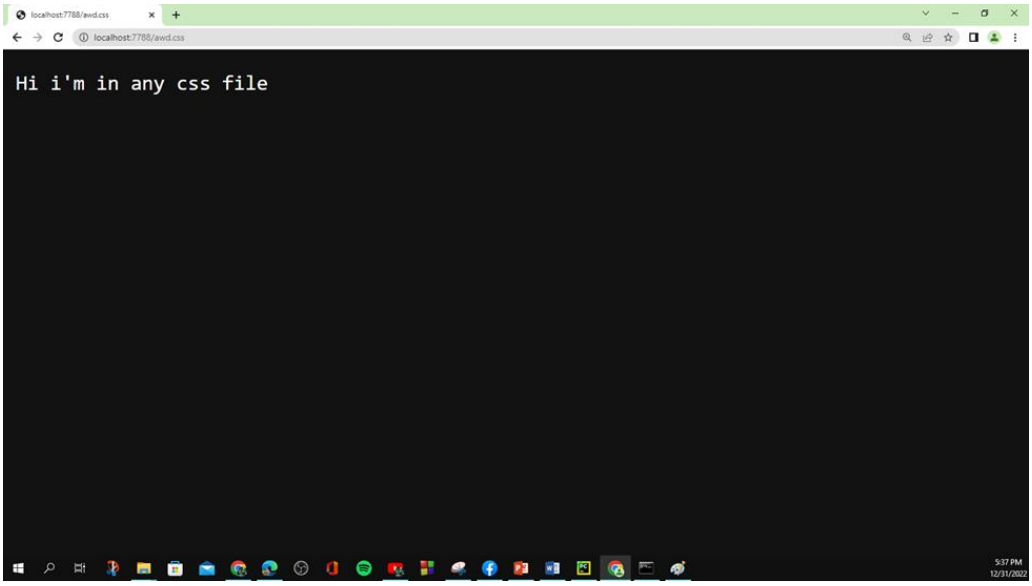


Figure 30: the css file on website

a .css file:

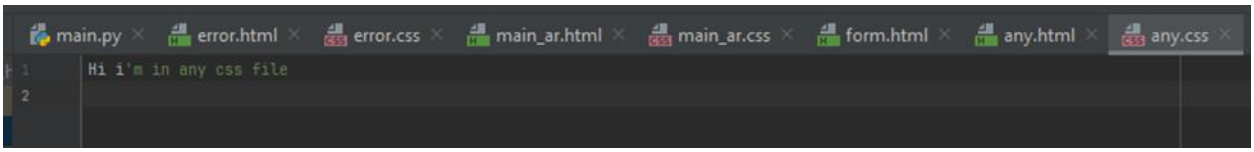


Figure 31: .css file

- if the request is a .png then the server sends any.png image with Content-Type: image/png.

on the command line(any png file):

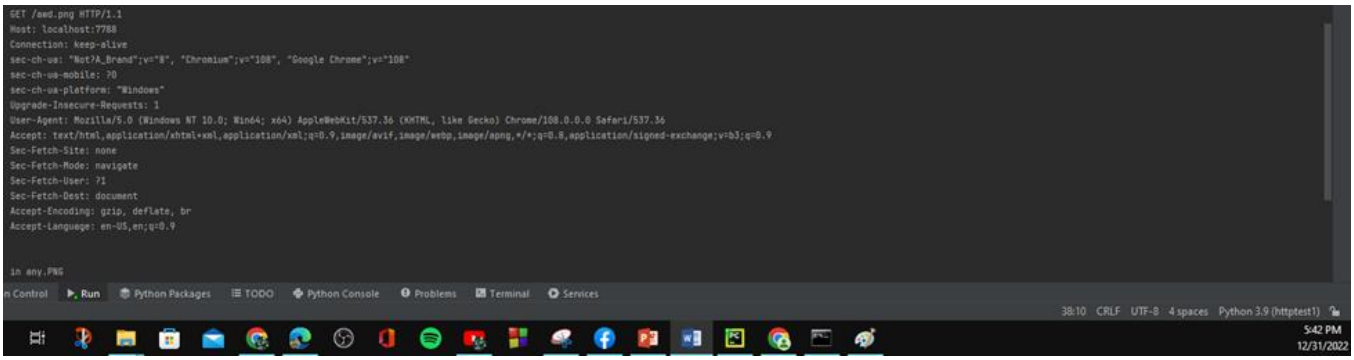


Figure 32: the request message for a png file

on the browser(any png file):

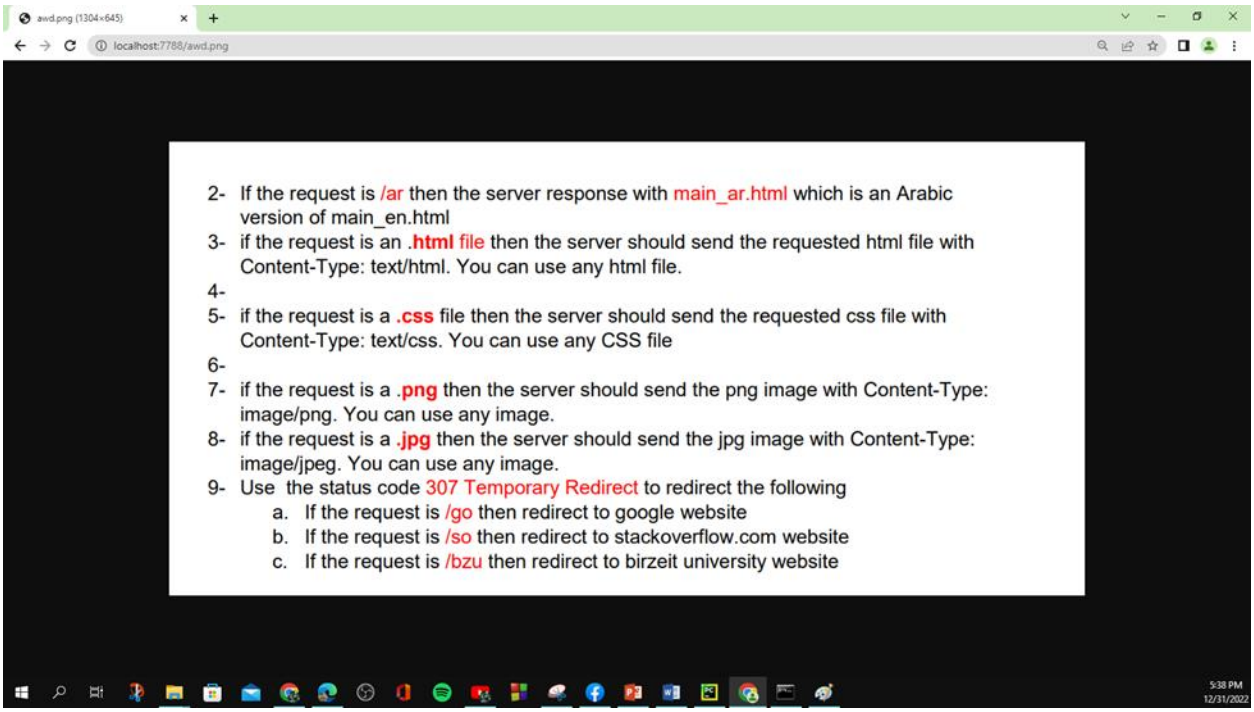


Figure 33: the png file on website

- ❖ if the request is a .jpg then the server sends any.jpg image with Content-Type: image/jpeg

on the command line(any jpg file):

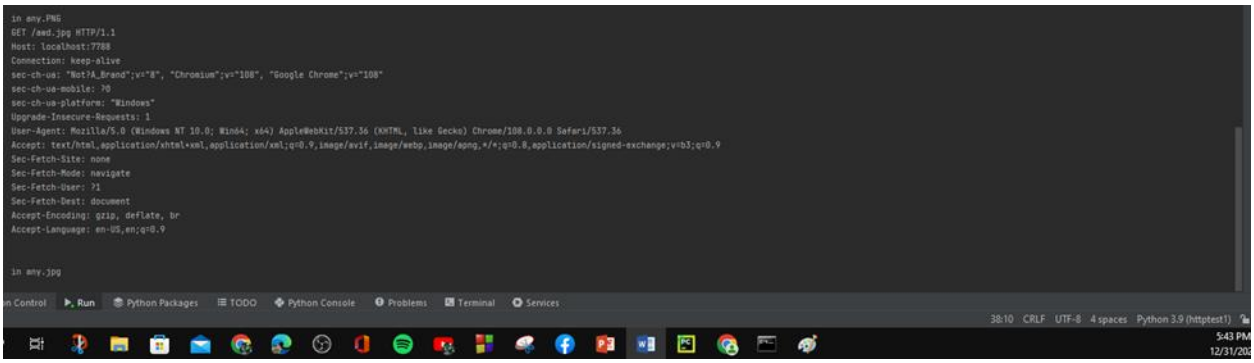


Figure 34: the request message for a jpg file

on the browser(any jpg file):

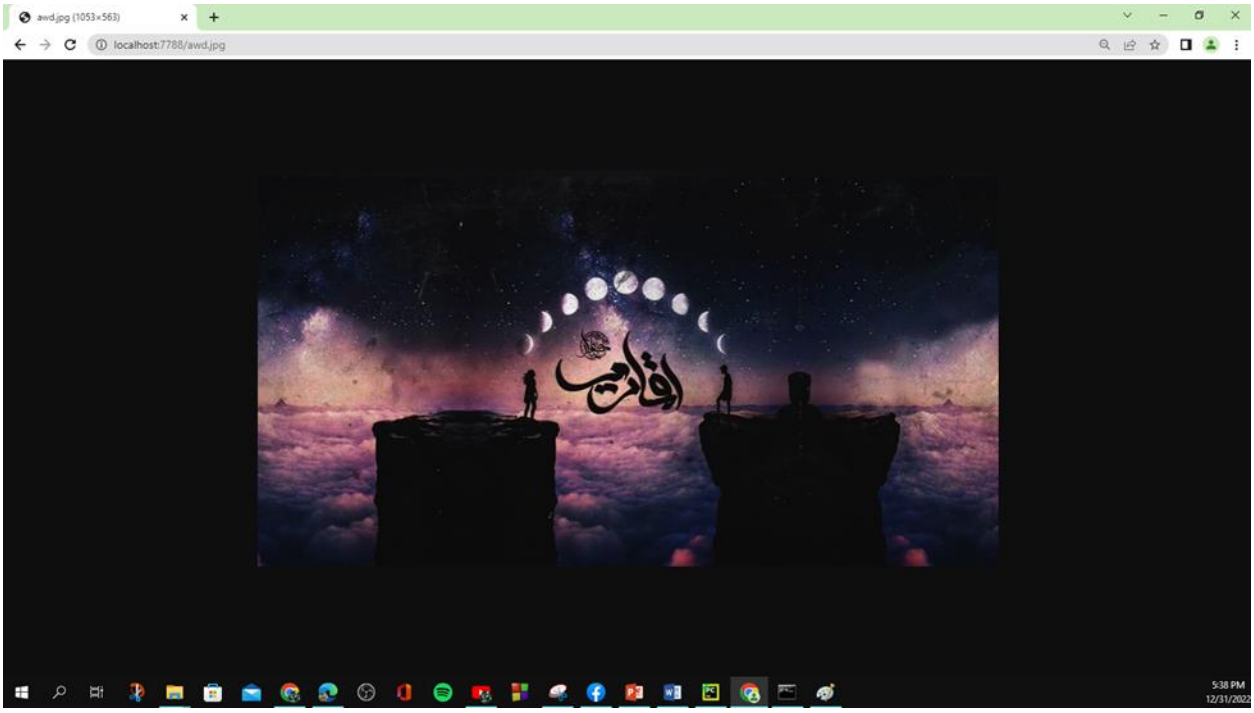


Figure 35:the jpg file on website

Google Website:

- If the request is /go then redirect to google website

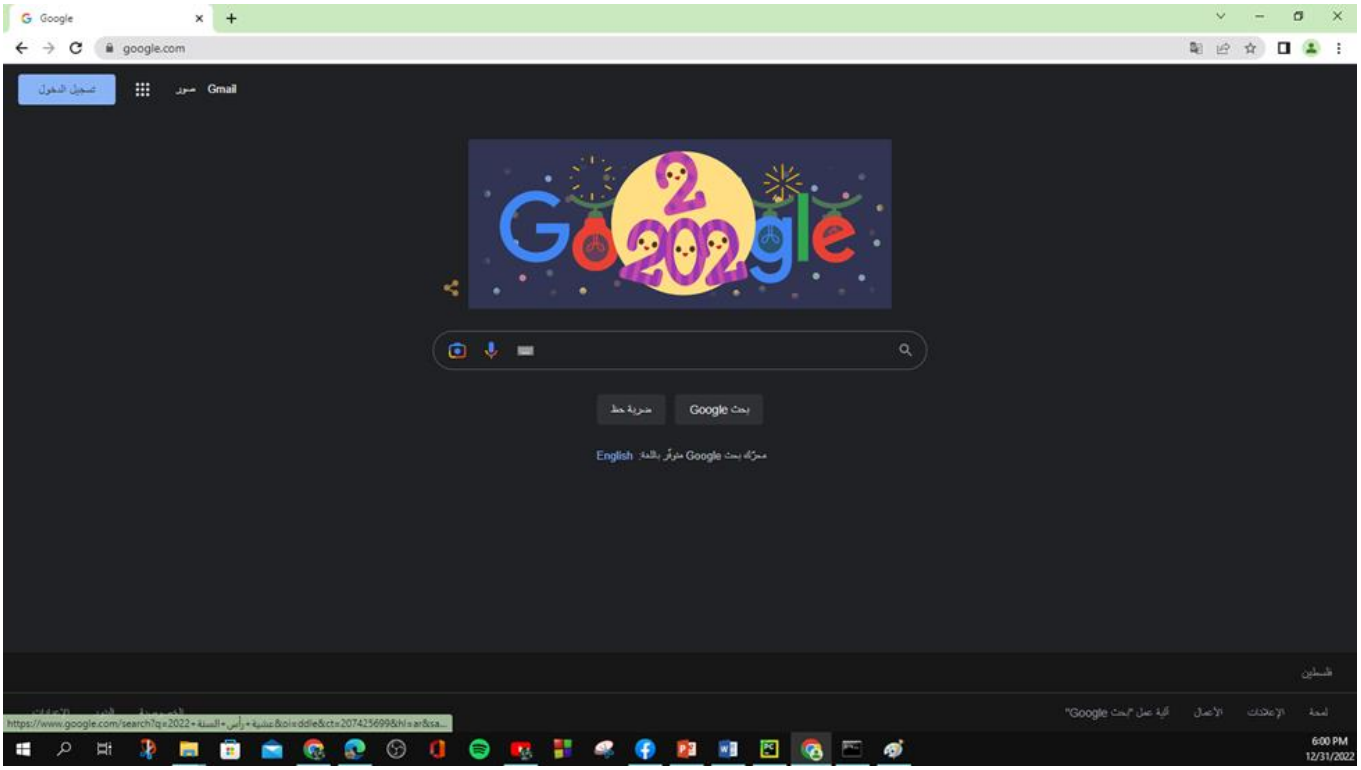


Figure 36: google website

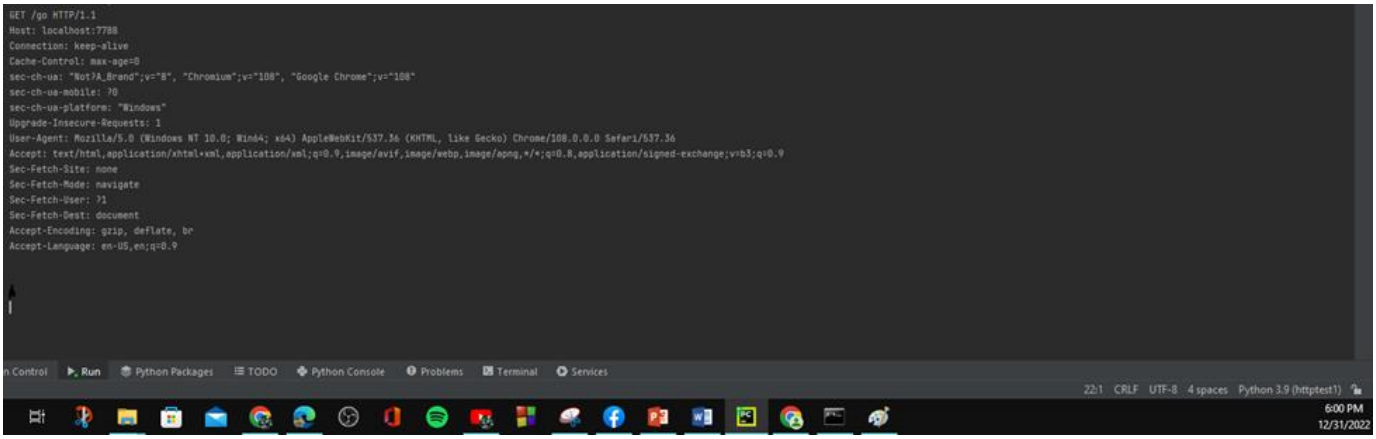


Figure 37: google website request message

Stackoverflow website:

- If the request is /so then redirect to stackoverflow.com website

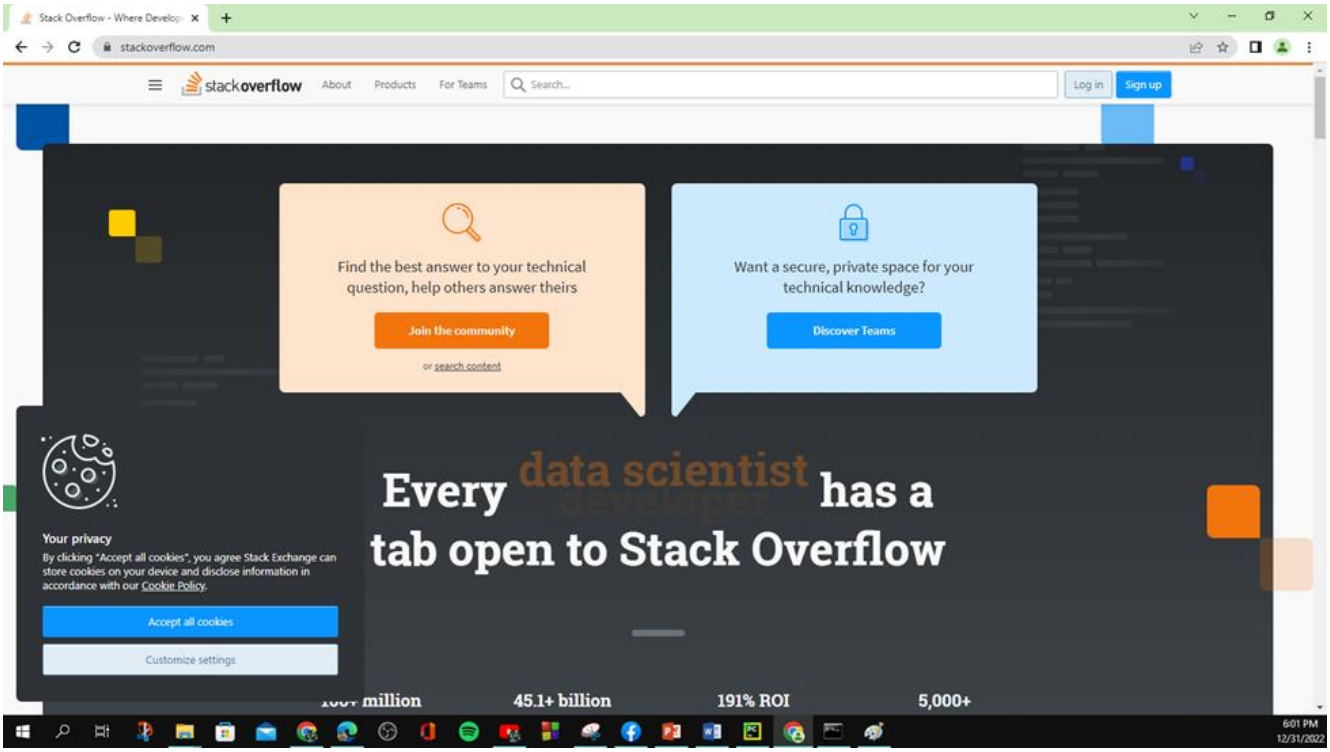


Figure 38: stackoverflow website

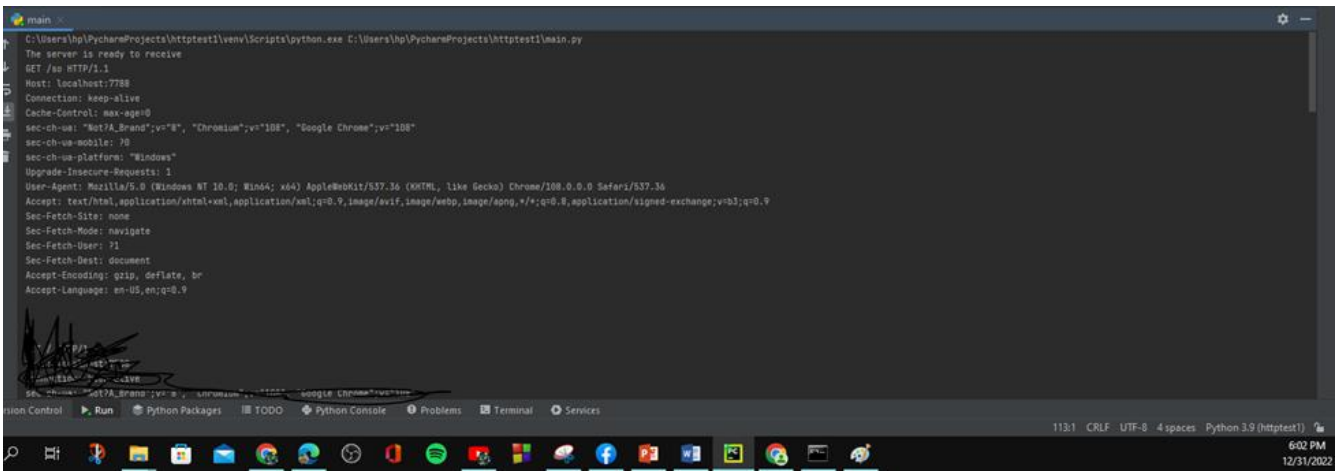


Figure 39:stackoverflow website requst message

Birzeit university website

- If the request is /bzu then redirect to birzeit university website

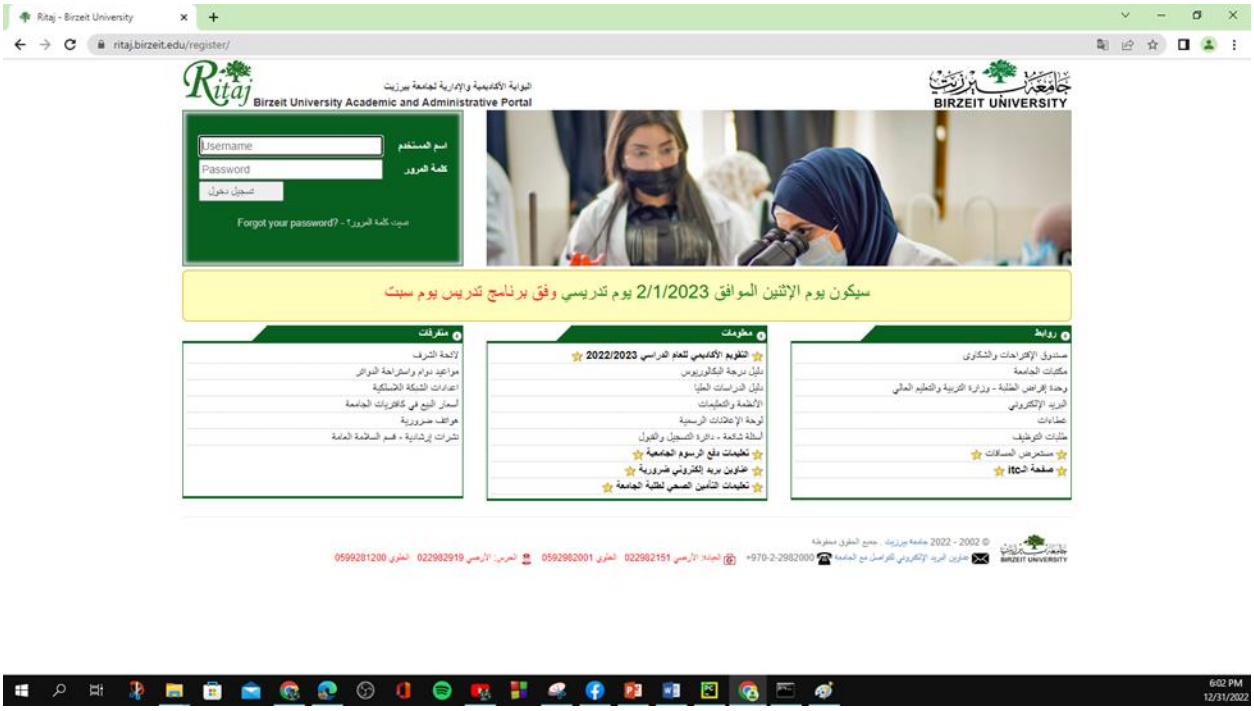


Figure 40:birzeit university website

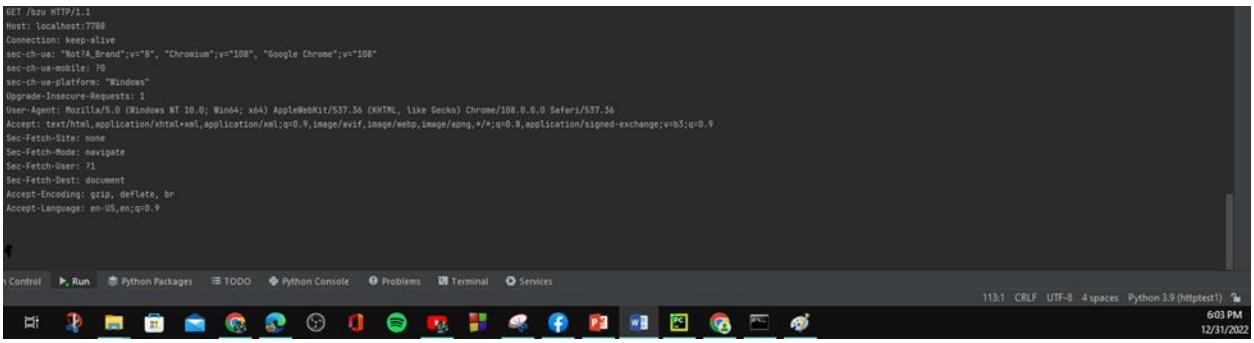


Figure 41:birzeit university website request message

Error Webpage

- If the request is wrong or the file doesn't exist the server returns a simple HTML Webpage with Content-Type text/html, that contains:
  1- "HTTP/1.1 404 Not Found" in the response status
  2- "Error" in the title
  3- "The file is not found" in the body in red
  4- Our names and IDs in Bold
  5- The IP and port number of the client

Here is an example in the figure below

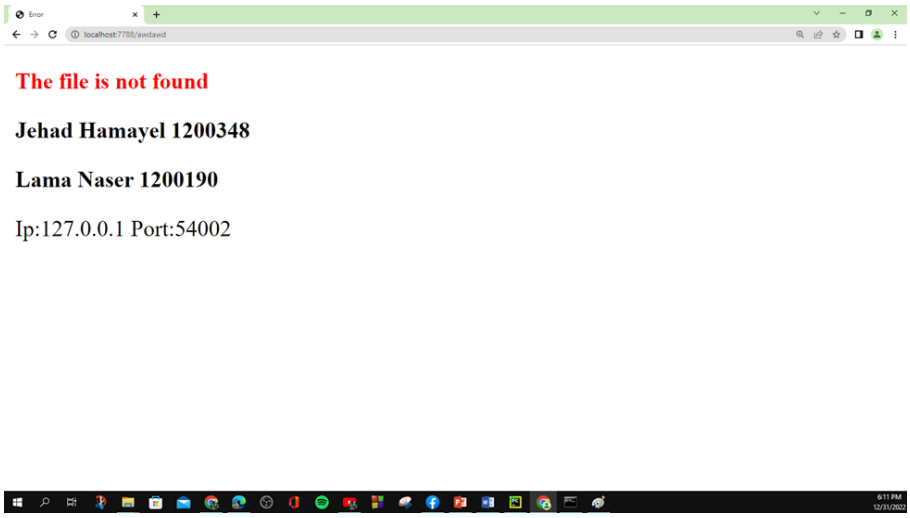


Figure 42: error webpage

On command line(Error Webpage):

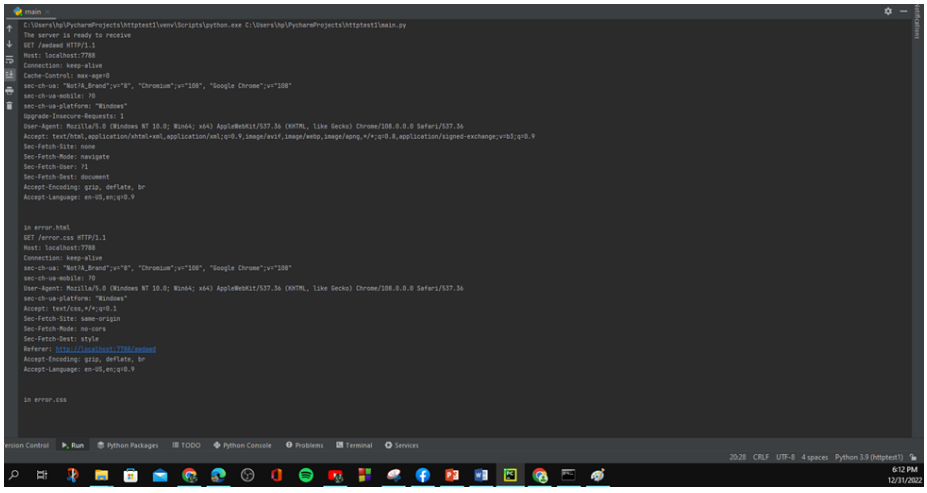


Figure 43: error webpage request message

error.html code:

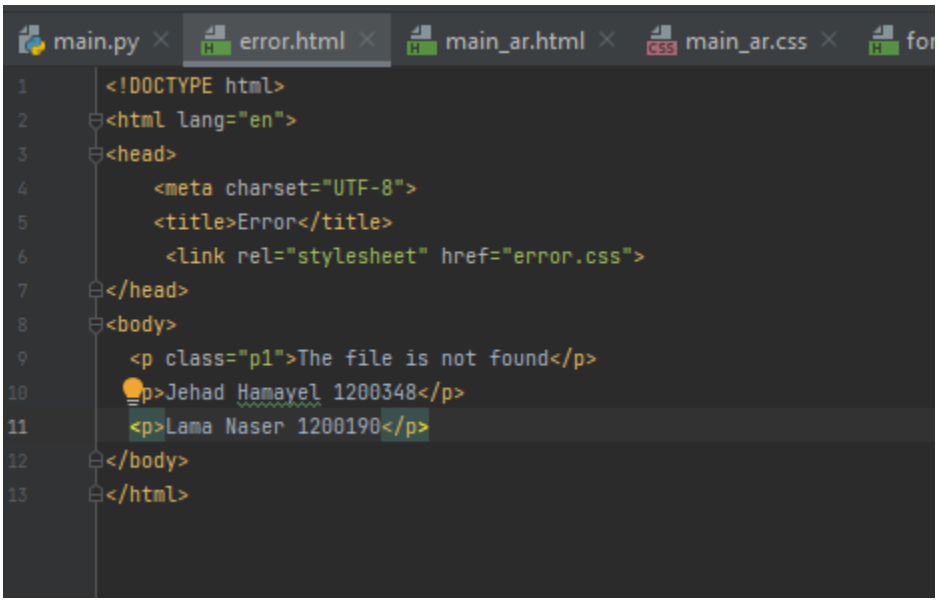


Figure 44: error.html code

error.css code:

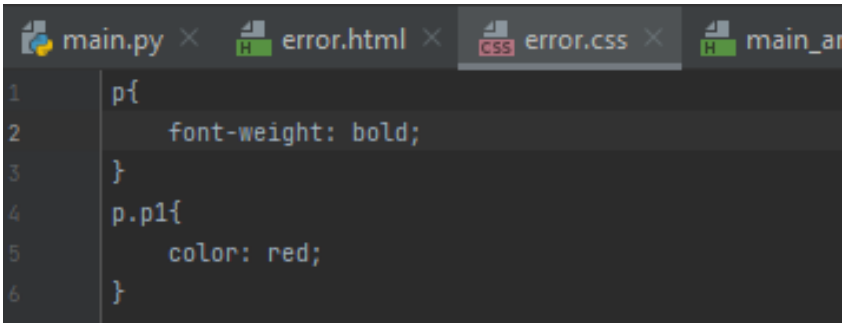


Figure 45: error.css code

Testing from another device

We tested the project from a phone, everything was going well.

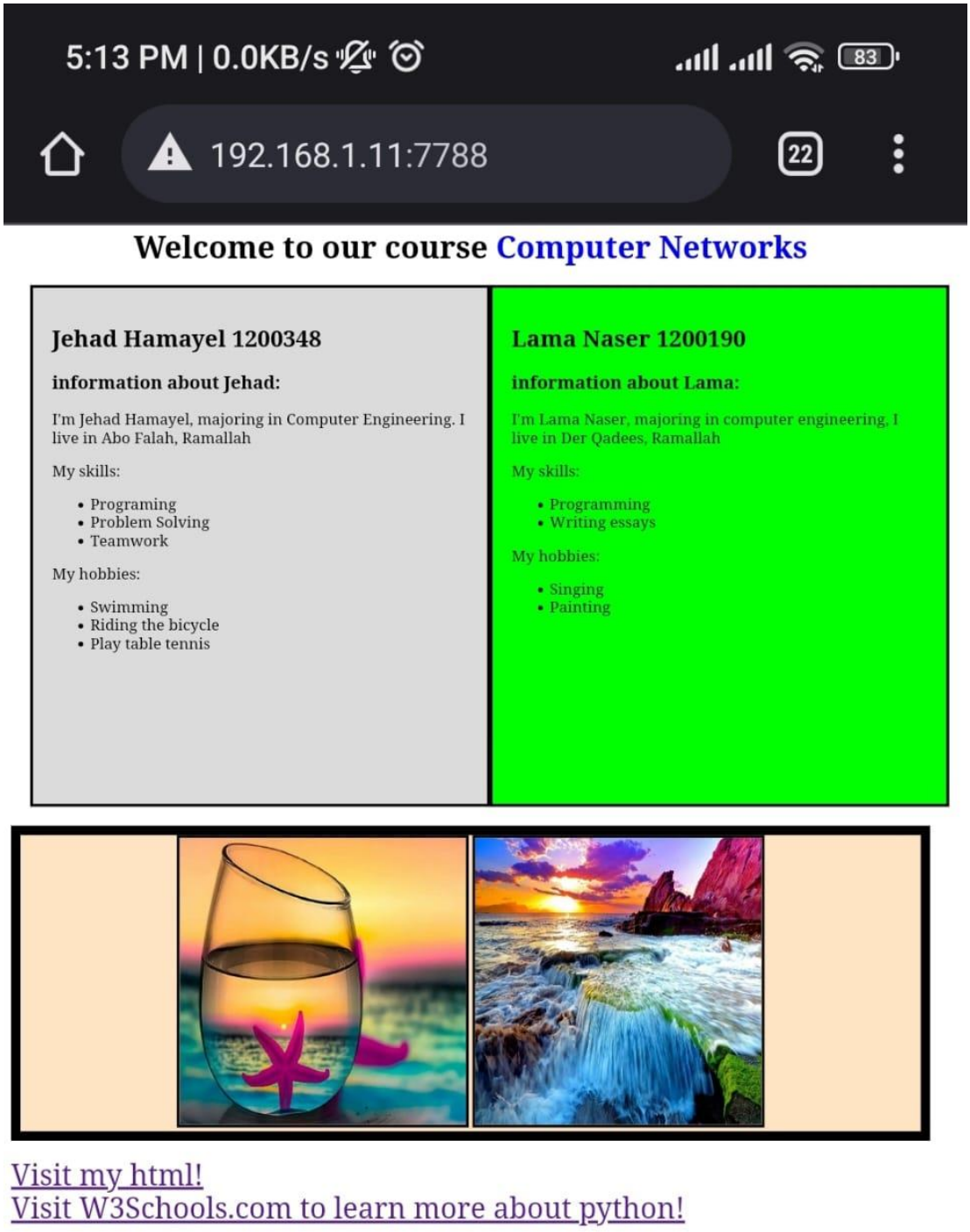


Figure 46:the test on the phone

On command line Testing from another device:

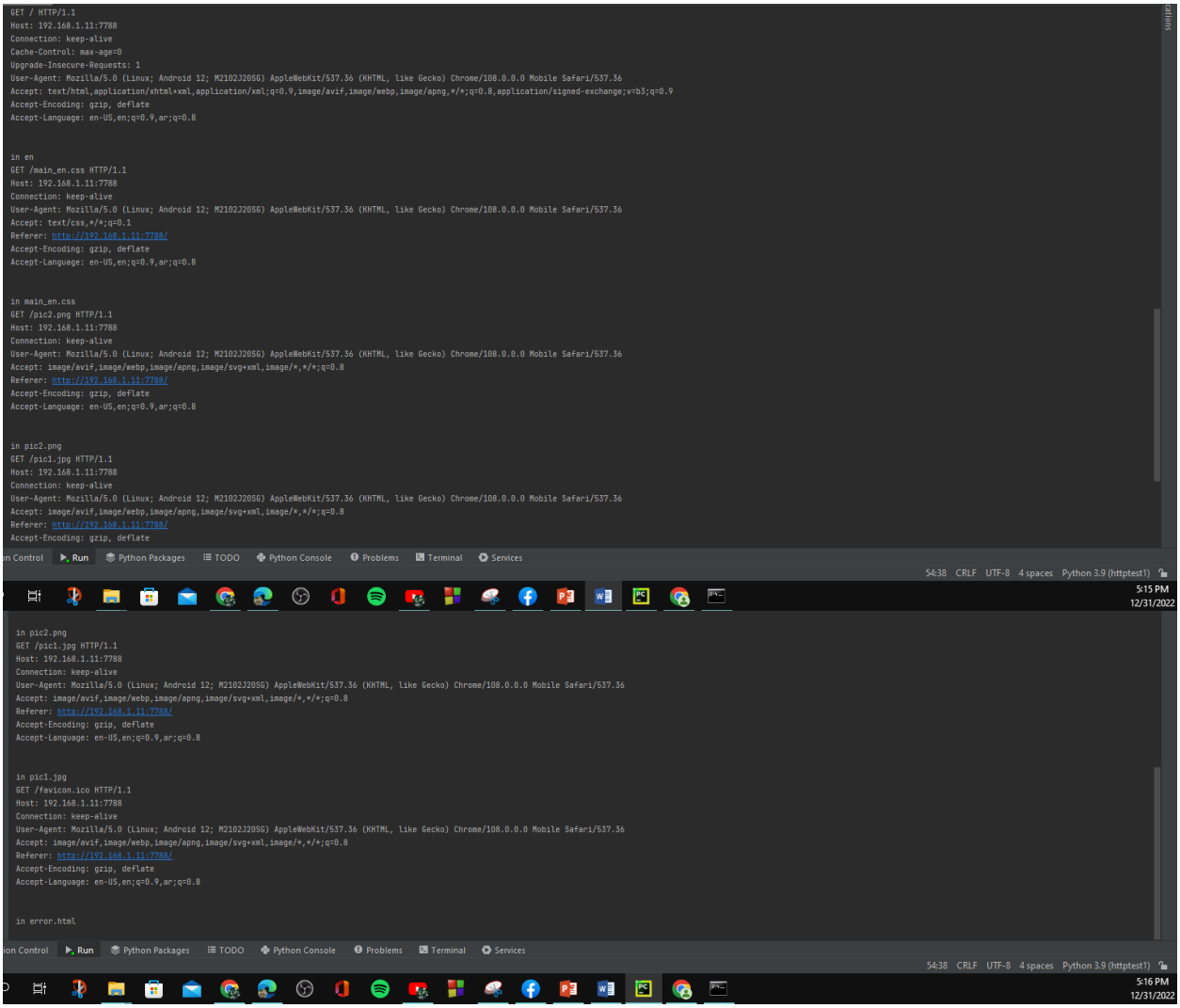


Figure 47: the request message

Conclusion

To sum up in this project, processes on different hosts communicate through network applications and we tried in this project to write simple applications in python to understand how connection happens and which type of protocols can be more appropriate for different applications