BIRZEIT UNIVERSITY

Faculty of Engineering & Technology

ENCS3320

COMPUTER NETWORKS

Project #1

Instructor: Dr. Abdalkarim Awad

GitHub

Group members:

❖ Faris Abufarha	ID: 1200546	Section: 1
❖ Nadeen Moreb	ID: 1203437	Section: 1
❖ Omar Badawi	ID: 1202428	Section: 3

Date: 1/06/2023

Table of contents

Introduction

In this project, our assignment involved multiple tasks. In Part 1, we addressed various questions and implemented some commands. Part 2 focused on creating a basic UDP client-server broadcasting program. For Part 3, we developed a simple web server using socket programming, HTML, and CSS to design the front-end of the website. Throughout the project, our team collaborated using Discord for communication and utilized Git and GitHub for version control. Python served as our primary programming language.

Part1

Abstract

An overview of key computer network concepts and operations is given in this part. It covers Telnet, Ping, Tracert, Nslookup, and explains how to utilize each one on a local network. Ping and Tracert are used to check network connectivity and measure response times, Nslookup obtains DNS data, and Telnet enables remote command-line access. Each command's use on a local network is demonstrated in the associated instructions.

Question 1

Q1) In your own words, what are ping, tracert, nslookup, and telnet (write one sentence for each one)

1. **Ping**(Packet Internet Groper): is a command that is used to check if a particular destination IP address exists and can accept requests in computer networks and determine round trip time, RTT.

2. **Tracert:** this command is used to trace the route the packet takes to travel to destination.

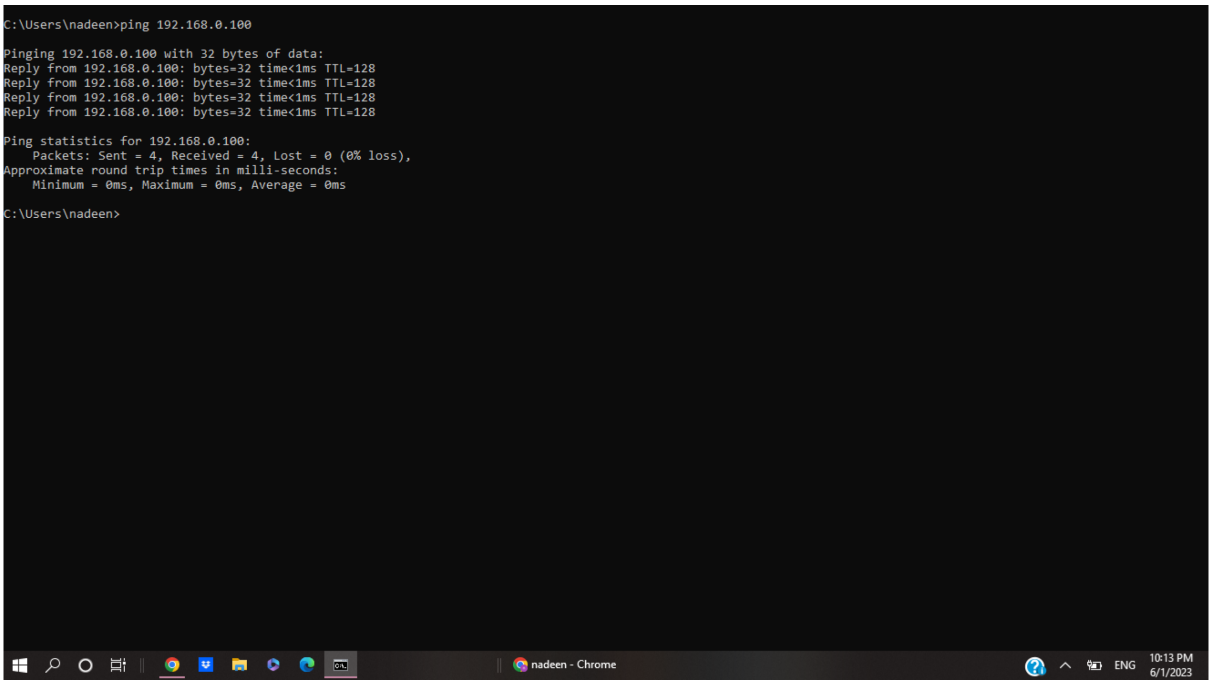
3. **Nslookup:** allows the users to lookup domain name system or IP address by entering the corresponding host name.

4. **Telnet:** a network protocol/interface that allows a local computer to establish a connection and communicate with remote servers and devices.

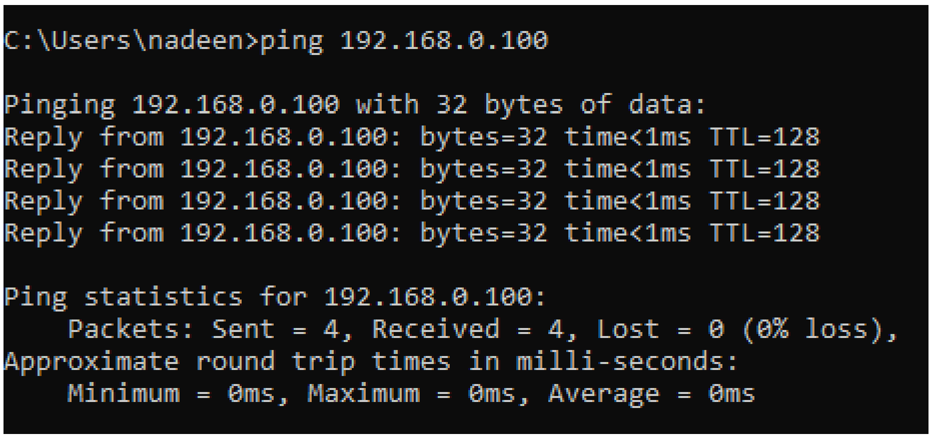
Question 2

Q2) Make sure that your computer is connected to the internet and then run the following commands:

1. **Ping a device in the same network, e.g. from a laptop to a smartphone**



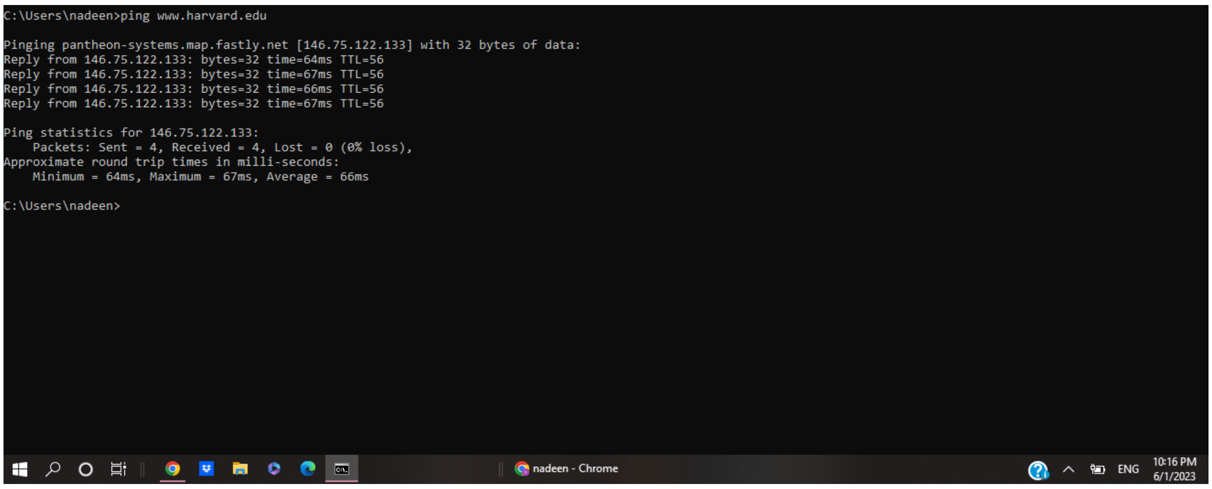
```
C:\Users\nadeen>ping 192.168.0.100

Pinging 192.168.0.100 with 32 bytes of data:
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\nadeen>
```



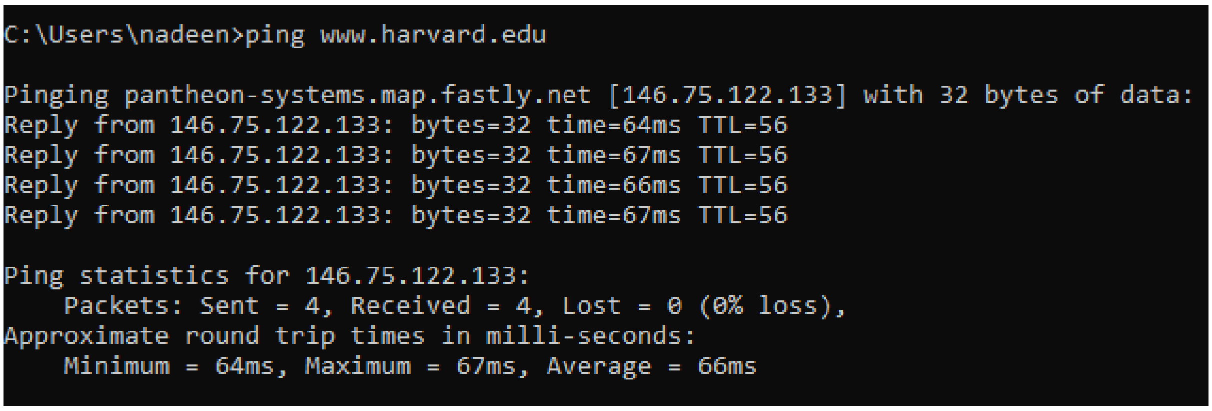
```
C:\Users\nadeen>ping 192.168.0.100

Pinging 192.168.0.100 with 32 bytes of data:
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128
Reply from 192.168.0.100: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

The packet is specified with a size of 32 bytes and each request will contain 32 bytes of data payload. The line reply indicates that the destination device has received the request packet successfully, and a reply has been received from the specified IP address.TTL has an initial value of 128.The output indicates that there was no loss, meaning that each packet request has had a corresponding packet response. Number of packets sent and received is 4. RTT maximum, minimum, and average value is 0 ms for each packet to travel from source to destination devices and back.

2. ping www.harvard.edu



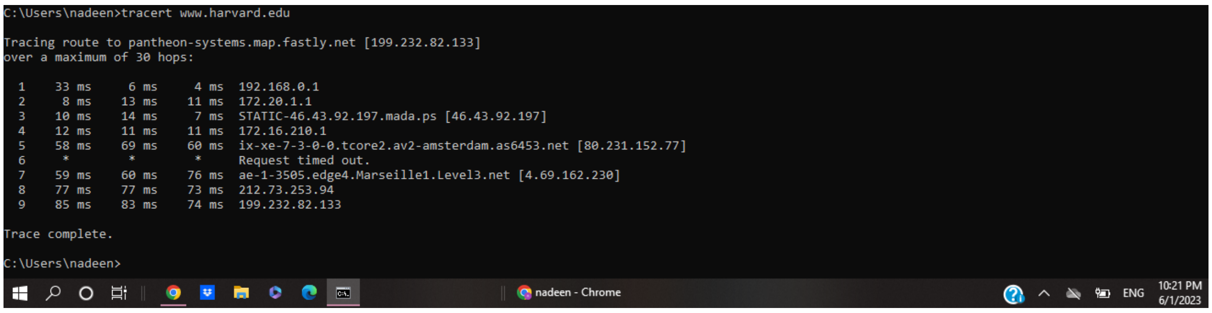
```
C:\Users\nadeen>ping www.harvard.edu

Pinging pantheon-systems.map.fastly.net [146.75.122.133] with 32 bytes of data:
Reply from 146.75.122.133: bytes=32 time=64ms TTL=56
Reply from 146.75.122.133: bytes=32 time=67ms TTL=56
Reply from 146.75.122.133: bytes=32 time=66ms TTL=56
Reply from 146.75.122.133: bytes=32 time=67ms TTL=56

Ping statistics for 146.75.122.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 64ms, Maximum = 67ms, Average = 66ms

C:\Users\nadeen>
```



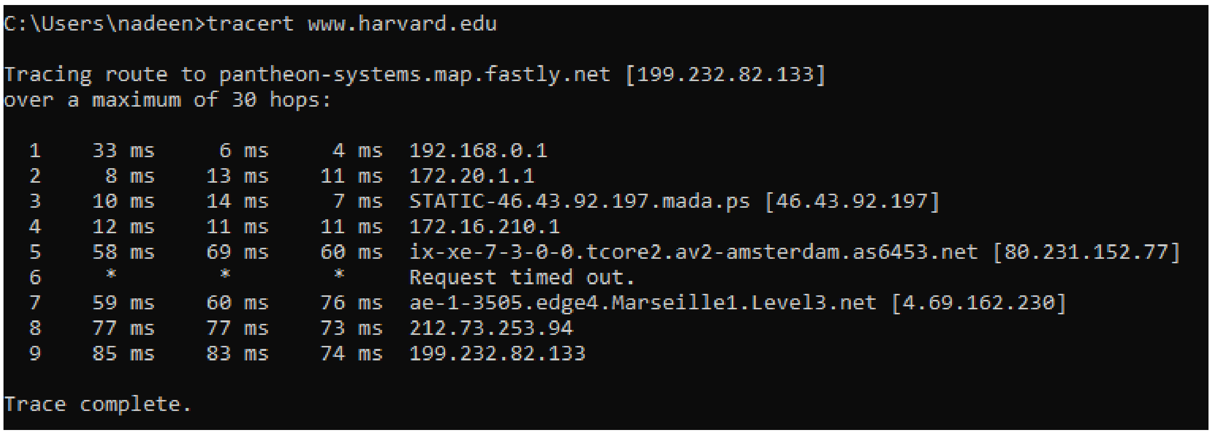
```
C:\Users\nadeen>ping www.harvard.edu

Pinging pantheon-systems.map.fastly.net [146.75.122.133] with 32 bytes of data:
Reply from 146.75.122.133: bytes=32 time=64ms TTL=56
Reply from 146.75.122.133: bytes=32 time=67ms TTL=56
Reply from 146.75.122.133: bytes=32 time=66ms TTL=56
Reply from 146.75.122.133: bytes=32 time=67ms TTL=56

Ping statistics for 146.75.122.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 64ms, Maximum = 67ms, Average = 66ms
```

The packet is specified with a size of 32 bytes and each request will contain 32 bytes of data payload. The line reply indicates that the destination device has received the request packet successfully, and a reply has been received from the specified IP address.TTL has an initial value of 56.The output indicates that there was no loss, meaning that each packet request has had a corresponding packet response. Number of packets sent and received is 4. RTT maximum, minimum, and average values are 67 ms, 64 ms, 66 ms respectively for each packet to travel from source to destination devices

3. **tracert www.harvard.edu**



```
C:\Users\nadeen>tracert www.harvard.edu

Tracing route to pantheon-systems.map.fastly.net [199.232.82.133]
over a maximum of 30 hops:

  1    33 ms     6 ms     4 ms  192.168.0.1
  2     8 ms    13 ms    11 ms  172.20.1.1
  3    10 ms    14 ms     7 ms  STATIC-46.43.92.197.mada.ps [46.43.92.197]
  4    12 ms    11 ms    11 ms  172.16.210.1
  5    58 ms    69 ms    60 ms  ix-xe-7-3-0-0.tcore2.av2-amsterdam.as6453.net [80.231.152.77]
  6     *        *        *     Request timed out.
  7    59 ms    60 ms    76 ms  ae-1-3505.edge4.Marseille1.Level3.net [4.69.162.230]
  8    77 ms    77 ms    73 ms  212.73.253.94
  9    85 ms    83 ms    74 ms  199.232.82.133

Trace complete.

C:\Users\nadeen>
```

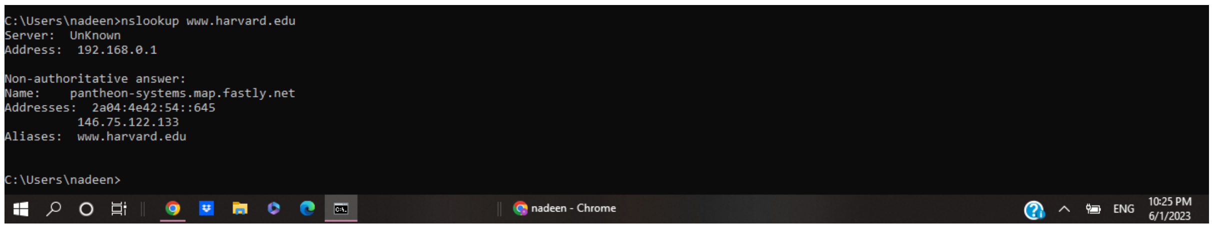


```
C:\Users\nadeen>tracert www.harvard.edu

Tracing route to pantheon-systems.map.fastly.net [199.232.82.133]
over a maximum of 30 hops:

  1    33 ms     6 ms     4 ms  192.168.0.1
  2     8 ms    13 ms    11 ms  172.20.1.1
  3    10 ms    14 ms     7 ms  STATIC-46.43.92.197.mada.ps [46.43.92.197]
  4    12 ms    11 ms    11 ms  172.16.210.1
  5    58 ms    69 ms    60 ms  ix-xe-7-3-0-0.tcore2.av2-amsterdam.as6453.net [80.231.152.77]
  6     *        *        *     Request timed out.
  7    59 ms    60 ms    76 ms  ae-1-3505.edge4.Marseille1.Level3.net [4.69.162.230]
  8    77 ms    77 ms    73 ms  212.73.253.94
  9    85 ms    83 ms    74 ms  199.232.82.133

Trace complete.
```

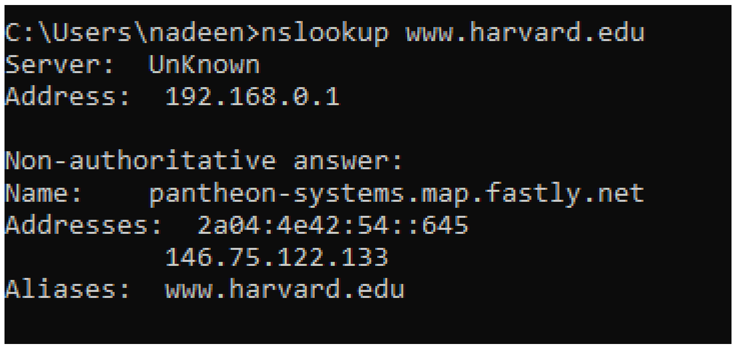
Maximum number of hops or network devices the command will attempt to reach before stopping the trace is 30 hops. For each hop, the output indicates the hop number, the round-trip time in ms, which starts at 6ms and ends at 83ms, for the ICMP or UDP packets to reach that hop and return, and the IP address of the router at that hop. The first IP Address is my network IP address, and the last IP address is usually associated with the destination IP Address if the trace is successfully completed.

"Trace completed" message indicates that the command has finished tracing the route, and the output provides information about the routers traversed from computer to reach www.harvard.edu.

4. nslookup www.harvard.edu



```
C:\Users\nadeen>nslookup www.harvard.edu
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
Name:    pantheon-systems.map.fastly.net
Addresses:  2a04:4e42:54::645
          146.75.122.133
Aliases:  www.harvard.edu

C:\Users\nadeen>
```



```
C:\Users\nadeen>nslookup www.harvard.edu
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
Name:    pantheon-systems.map.fastly.net
Addresses:  2a04:4e42:54::645
          146.75.122.133
Aliases:  www.harvard.edu
```

The output displays that the DNS server used for the query is unknown, at IP address 192.168.0.1. It also provides the IP addresses associated with the domain name www.harvard.edu, which are “2a04:4e42:54::645” and “146.75.122.133”. It also lists an alias, www.harvard.edu, which is an alternate name for the same entity.

Part2

Abstract

The purpose of this section is to create a Python application that consists of a UDP client and server. The server will be set up to listen on Port 8855, while each client will periodically send a broadcast message every 2 seconds. The content of the message will include the name of the client. On the server side, whenever a message is received, the application will print all the received messages, along with the updated timestamp from each client. In ordered to make the message a broadcast, we make some changes in the IP of the server depending on the subnet mask of the server, if the server has three '255' in his first three octets we change the last octet to 255 in the server IP, if first 2 octets are '255' we change the last 2 octets of the IP of the server to '255', and so on.

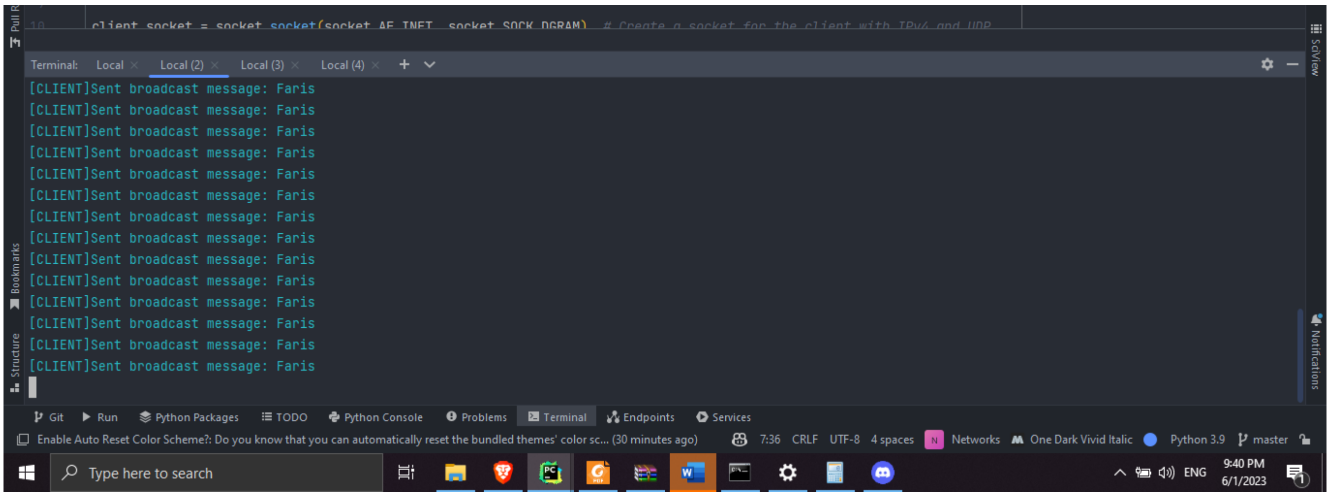
Code

```python
# server.py
import socket
import time
from colors import Colors

HOST_IP = socket.gethostbyname(socket.gethostname())  # my IP
HOST_PORT = 8855  # Port that will be used by the server
ADDRESS = (HOST_IP, HOST_PORT)  # Address of the server
FORMAT = 'utf-8'  # Format of the message that will be sent by the client

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create a socket
server_socket.bind(ADDRESS)  # Bind the address to the socket
last_received_msg = {}  # Dictionary to store last received message from each client


def get_current_time():  # Function to get current time
    # get current time
    t = time.localtime()
    current_time = time.strftime("%H:%M:%S", t)
    return current_time

def handle_client():  # Function to handle clients
    while True:  # Loop to make the server running forever
        client_message, client_address = server_socket.recvfrom(1024)  # Receive message from the
client
        received_time = get_current_time()  # Get current time
        msg = client_message.decode(FORMAT)  # Decode the message to get a string
        last_received_msg[
            client_address] = f'{msg} at {received_time}'  # Store the client address with its last
message

        print(f'{Colors.OKBLUE}-{Colors.ENDC}'*10,end='')
        server_msg = '[SERVER] UPDATED MESSAGES'
        print(f'{Colors.OKGREEN}{server_msg}{Colors.ENDC}',end='')
        print(f'{Colors.OKBLUE}-{Colors.ENDC}' * 10)

        cnt = 1  # Counter to print the number of the client
        for address, msg in last_received_msg.items():
            print(f'{cnt}- received message from  {address}  {msg}')
            cnt += 1
        print(f'{Colors.OKBLUE}-{Colors.ENDC}' * (20 + len(server_msg)))
        print()

def start():
   print(f'{Colors.OKGREEN}[ONLINE] server is running on {HOST_IP} {Colors.ENDC}')  #
Print that the server is running
   handle_client()


if __name__ == '__main__':
   start()
```

```
 # client.py
import socket
import time
from colors import Colors

HOST_IP = '192.168.1.255'  # my IP add 255 instead of last octet
HOST_PORT = 8855  # Port that will be used by the server
FORMAT = 'utf-8'  # Format of the message that will be sent by the client

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create a socket for the
client with IPv4 and UDP
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)  # Enable broadcasting
mode 1 is for True


student_name = input('Enter your name: ')   # Get the name of the student from the user
def udp_client(): # Function to handle the client
   while True:
       message = f"{student_name}".encode(FORMAT) # Encode the message to bytes
       client_socket.sendto(message, (HOST_IP, HOST_PORT)) # Send the message to the server
       print(f"{Colors.OKCYAN}[CLIENT]Sent broadcast message:
{message.decode(FORMAT)}{Colors.ENDC}") # Print the message that will be sent
       time.sleep(2) # Wait for 2 seconds before sending the next message


if __name__ == '__main__':
   udp_client()
```

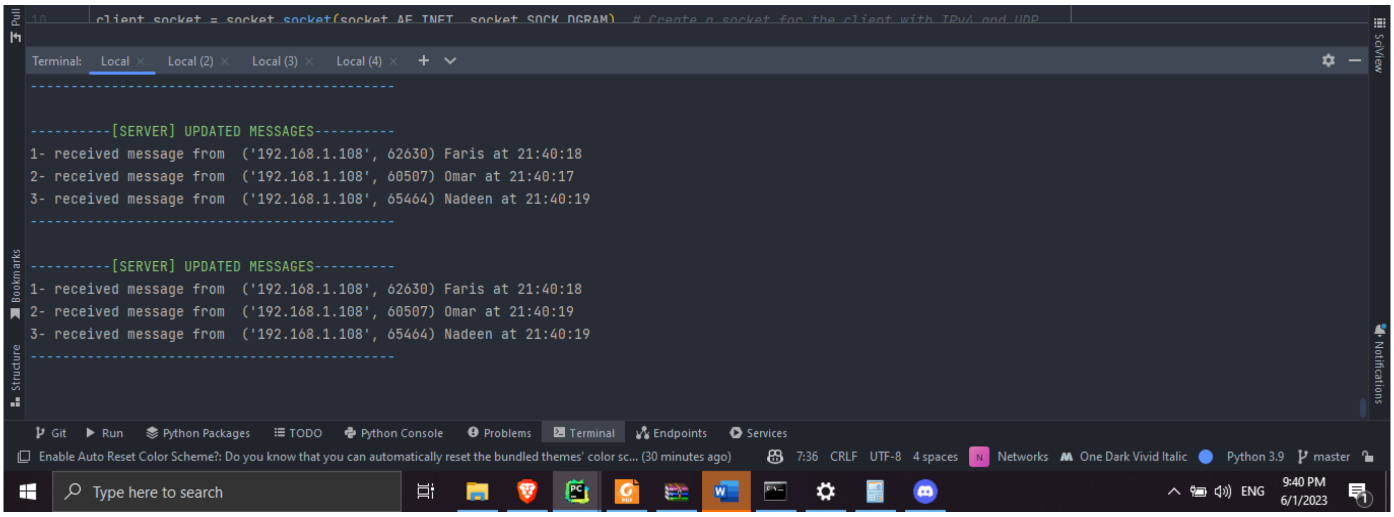
```python
# colors.py
# This file contains the colors that will be used in the server.py, client.py files to print colored messages.
class Colors:
   HEADER = '\033[95m'
   OKBLUE = '\033[94m'
   OKCYAN = '\033[96m'
   OKGREEN = '\033[92m'
   WARNING = '\033[93m'
   FAIL = '\033[91m'
   ENDC = '\033[0m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
```

Results



```
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
[CLIENT]Sent broadcast message: Faris
```

Client terminal printing the broadcast message that has been sent



```
---------------------------------------------

----------[SERVER] UPDATED MESSAGES----------
1- received message from  ('192.168.1.108', 62630) Faris at 21:40:18
2- received message from  ('192.168.1.108', 60507) Omar at 21:40:17
3- received message from  ('192.168.1.108', 65464) Nadeen at 21:40:19
---------------------------------------------

----------[SERVER] UPDATED MESSAGES----------
1- received message from  ('192.168.1.108', 62630) Faris at 21:40:18
2- received message from  ('192.168.1.108', 60507) Omar at 21:40:19
3- received message from  ('192.168.1.108', 65464) Nadeen at 21:40:19
---------------------------------------------
```

Server terminal printing the broadcasts with updated times whenever a new message received

Note: in these screenshots, client and server are the same PC, so we consider both IP and Port number, we tried on different PCs and it works as well.

Part3

Abstract

This part presents the implementation of a simple HTTP server using Python sockets. The server listens for connections and handles HTTP requests from clients. The server's goal is to serve requested files and redirect specific URLs.

Built with the socket module in Python, the server follows a loop-based structure, continuously listening for new connections. Upon accepting a connection, it parses the client's HTTP request to extract the requested URL path. Based on the path, the server determines the file to serve and sets the appropriate content type.

The server supports various file types like HTML, CSS, PNG, and JPG. It handles requests for different language versions of the main page and provides a 404 Not Found response for invalid paths or requests using a predefined "NotFound.html" file.

Additionally, the server includes URL redirection for specific paths. Requests to "/yt" are redirected to "https://www.youtube.com", requests to "/so" are redirected to "https://stackoverflow.com", and requests to "/rt" are redirected to "https://ritaj.birzeit.edu".

The implementation showcases error handling for file not found and I/O errors. Detailed status messages are printed during execution for debugging and verification.

Overall, this part demonstrates a basic HTTP server implementation using Python sockets, offering file serving and URL redirection capabilities.

Code

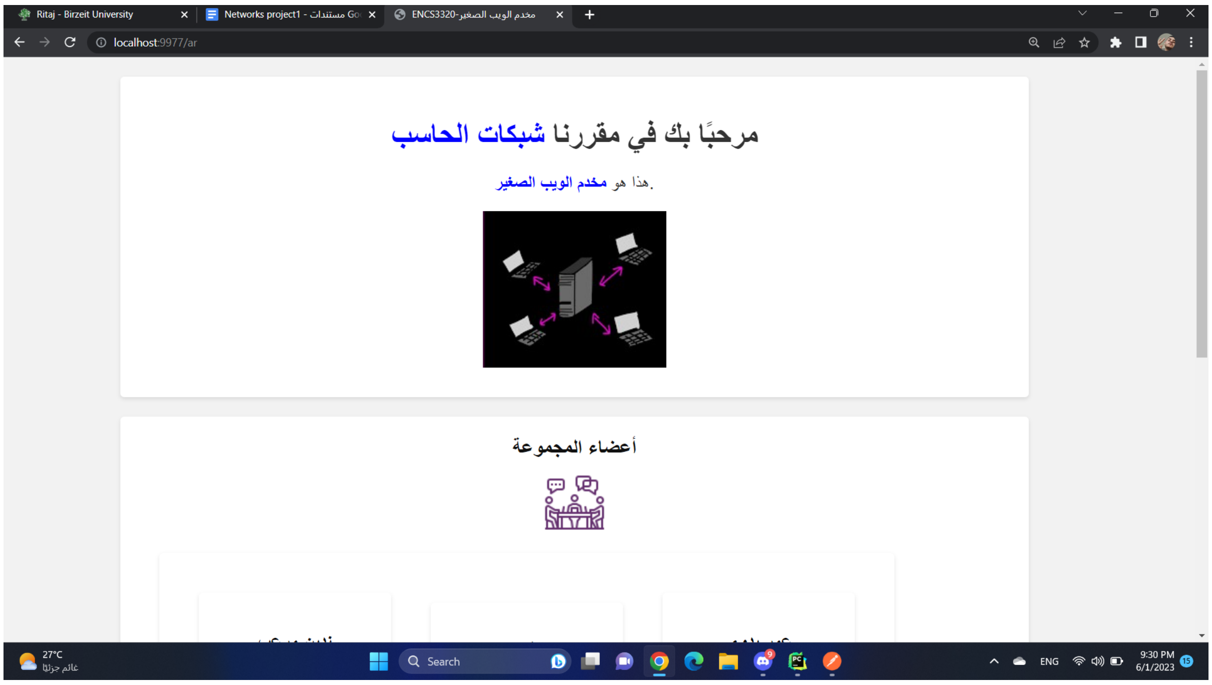
```python
from socket import *

# Define the server port
serverPort = 9977

# Create a TCP socket
serverSocket = socket(AF_INET, SOCK_STREAM)

# Bind the socket to the specified port
serverSocket.bind(("", serverPort))

# Listen for incoming connections
serverSocket.listen(1)

# Print a message indicating that the server is ready to receive connections
print("The server is ready to receive")

while True:
  try:
    # Accept a connection from a client
    connectionSocket, addr = serverSocket.accept()

    # Receive the request sentence from the client and decode it
    sentence = connectionSocket.recv(2048).decode()

    # Print the client's address and the received sentence
    print(addr)
    print(sentence)

    # Extract the client's IP address and port number
    ip = addr[0]
    port = addr[1]

    # Check the requested URL
    if sentence.startswith("GET /") and (" HTTP/1.1" in sentence or " HTTP/1.0" in sentence):
      # Extract the requested path
      path = sentence.split(" ")[1]

      # Redirect requests to specific paths
      if path == "/yt":
        connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
        connectionSocket.send("Location: https://www.youtube.com\r\n".encode())
        connectionSocket.send("\r\n".encode())
        connectionSocket.close()
        continue
      elif path == "/so":
        connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
        connectionSocket.send("Location: https://stackoverflow.com\r\n".encode())
```

```
            connectionSocket.send("\r\n".encode())
            connectionSocket.close()
            continue
        elif path == "/rt":
            connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
            connectionSocket.send("Location: https://ritaj.birzeit.edu\r\n".encode())
            connectionSocket.send("\r\n".encode())
            connectionSocket.close()
            continue

        # Set default content type to "text/html"
        content_type = "text/html"

        # Determine the requested file based on the path and set appropriate content type
        if path in ["/", "/index.html", "/main_en.html", "/en"]:
            filename = "main_en.html"
            content_type = "text/html"
        elif path == "/ar":
            filename = "main_ar.html"
            content_type = "text/html"
        elif path.endswith(".html"):
            filename = path[1:]
            content_type = "text/html"
        elif path.endswith((".css")):
            filename = path[1:]
            content_type = "text/css"
        elif path.endswith((".png")):
            filename = path[1:]
            content_type = "image/png"
        elif path.endswith((".jpg")):
            filename = path[1:]
            content_type = "image/jpg"
        else:
            # If path is not found, send a 404 Not Found response
            with open("NotFound.html", "rb") as file:
                data = file.read()
            connectionSocket.send("HTTP/1.1 404 Not Found\r\n".encode())
            connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
            connectionSocket.send("\r\n".encode())
            connectionSocket.send(data)
            connectionSocket.close()
            continue

        try:
            # Open and send the requested file
            with open(filename, "rb") as file:
                data = file.read()
                connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
                connectionSocket.send(f"Content-Type: {content_type}; charset=utf-8\r\n".encode())
                connectionSocket.send("\r\n".encode())
```

```python
                connectionSocket.send(data)
        except FileNotFoundError:
            # Send a 404 Not Found response if the file doesn't exist
            with open("NotFound.html", "rb") as file:
                data = file.read()
            connectionSocket.send("HTTP/1.1 404 Not Found\r\n".encode())
            connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
            connectionSocket.send("\r\n".encode())
            connectionSocket.send(data)
            connectionSocket.close()
    else:
        # Send a 404 Not Found response for invalid requests
        with open("NotFound.html", "rb") as file:
            data = file.read()
        connectionSocket.send("HTTP/1.1 404 Not Found\r\n".encode())
        connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
        connectionSocket.send("\r\n".encode())
        connectionSocket.send(data)
        connectionSocket.close()

    connectionSocket.close()
except OSError:
    print("IO error")
else:
    print("OK")
```

Results

# http://localhost:9977/ar

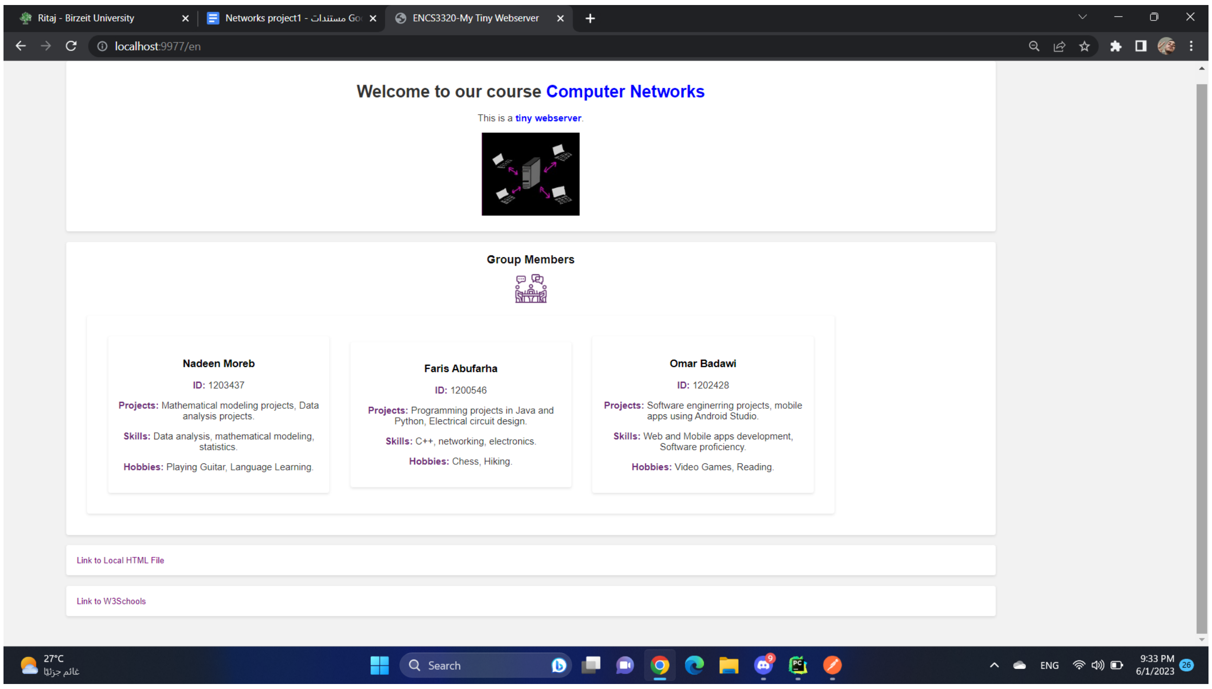

localhost:9977/ar

مرحبًا بك في مقررنا **شبكات الحاسب**

هذا هو **مدخم الويب الصغير**.

أعضاء المجموعة

# http://localhost:9977/en


localhost:9977/en

Welcome to our course Computer Networks

This is a **tiny webserver**.

Group Members

Nadeen Moreb

ID: 1203437

Projects: Mathematical modeling projects, Data analysis projects.

Skills: Data analysis, mathematical modeling, statistics.

Hobbies: Playing Guitar, Language Learning.

Faris Abufarha

ID: 1200546

Projects: Programming projects in Java and Python, Electrical circuit design.

Skills: C++, networking, electronics.

Hobbies: Chess, Hiking.

Omar Badawi

ID: 1202428

Projects: Software enginerring projects, mobile apps using Android Studio.

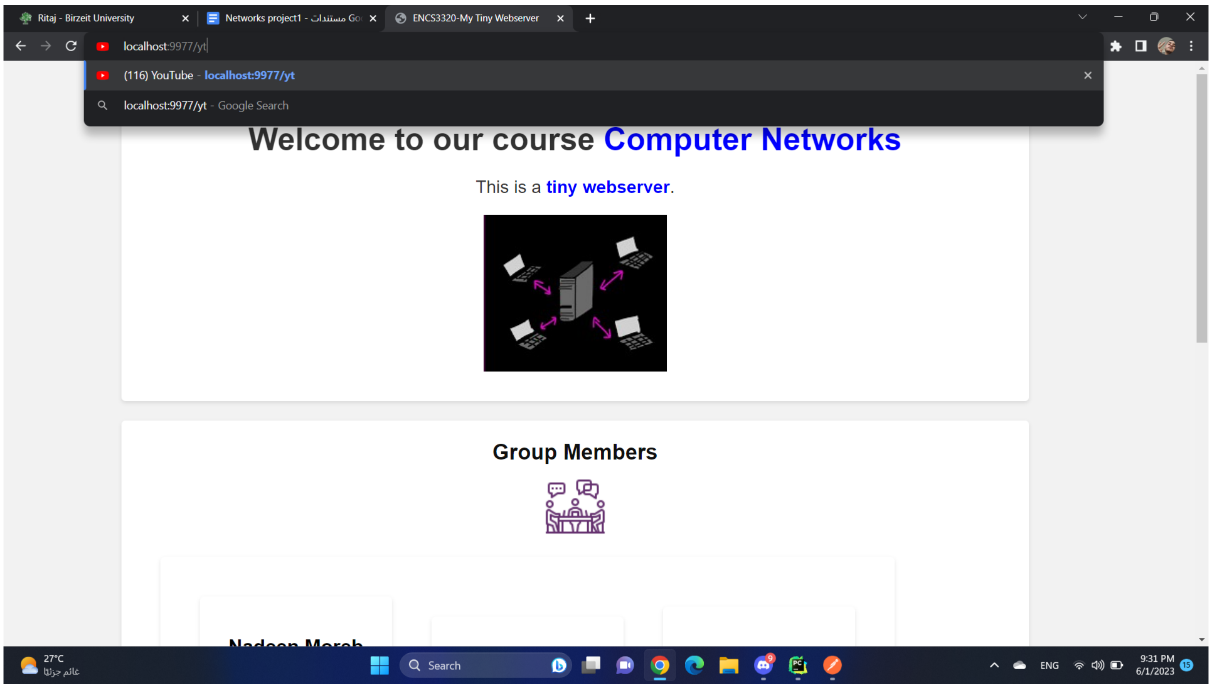
Skills: Web and Mobile apps development, Software proficiency.

Hobbies: Video Games, Reading.

Link to Local HTML File

Link to W3Schools

# redirecting to youtube


Ritaj - Birzeit University
Networks project1 - مستندات Go
ENCS3320-My Tiny Webserver
localhost:9977/yt
(116) YouTube - localhost:9977/yt
localhost:9977/yt - Google Search
Search
ENG
9:31 PM
6/1/2023

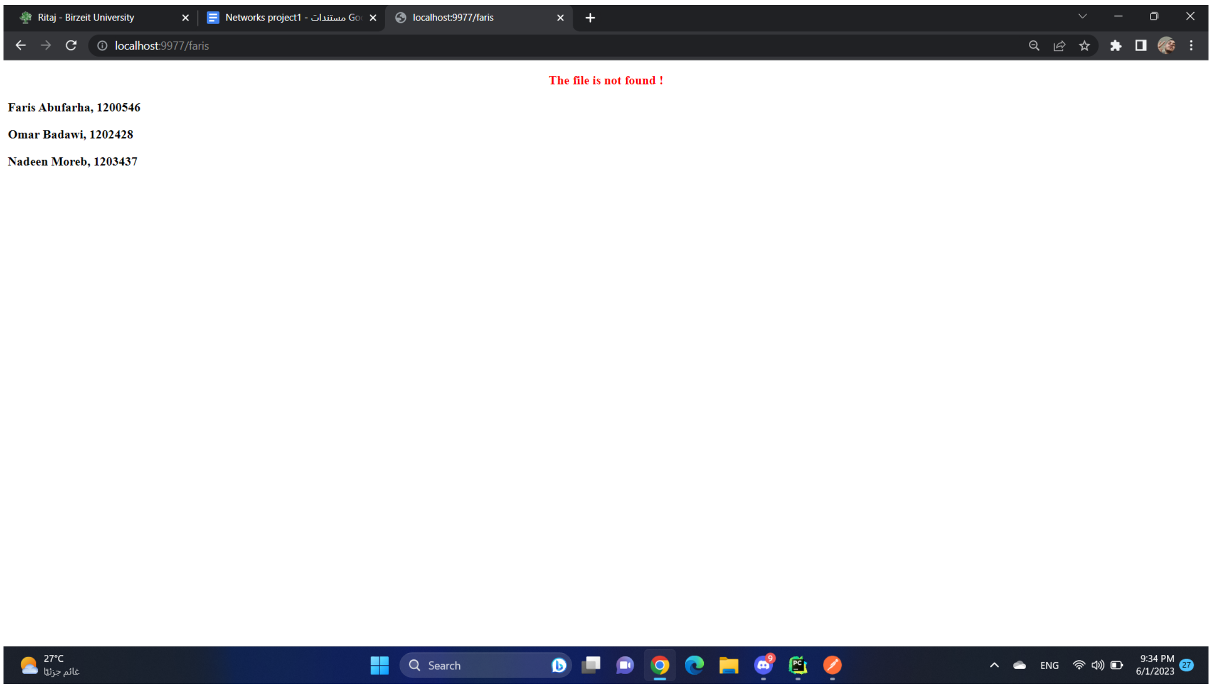
Welcome to our course **Computer Networks**

This is a **tiny webserver**.

Group Members

Nadeen Moreb

# Unfound file :



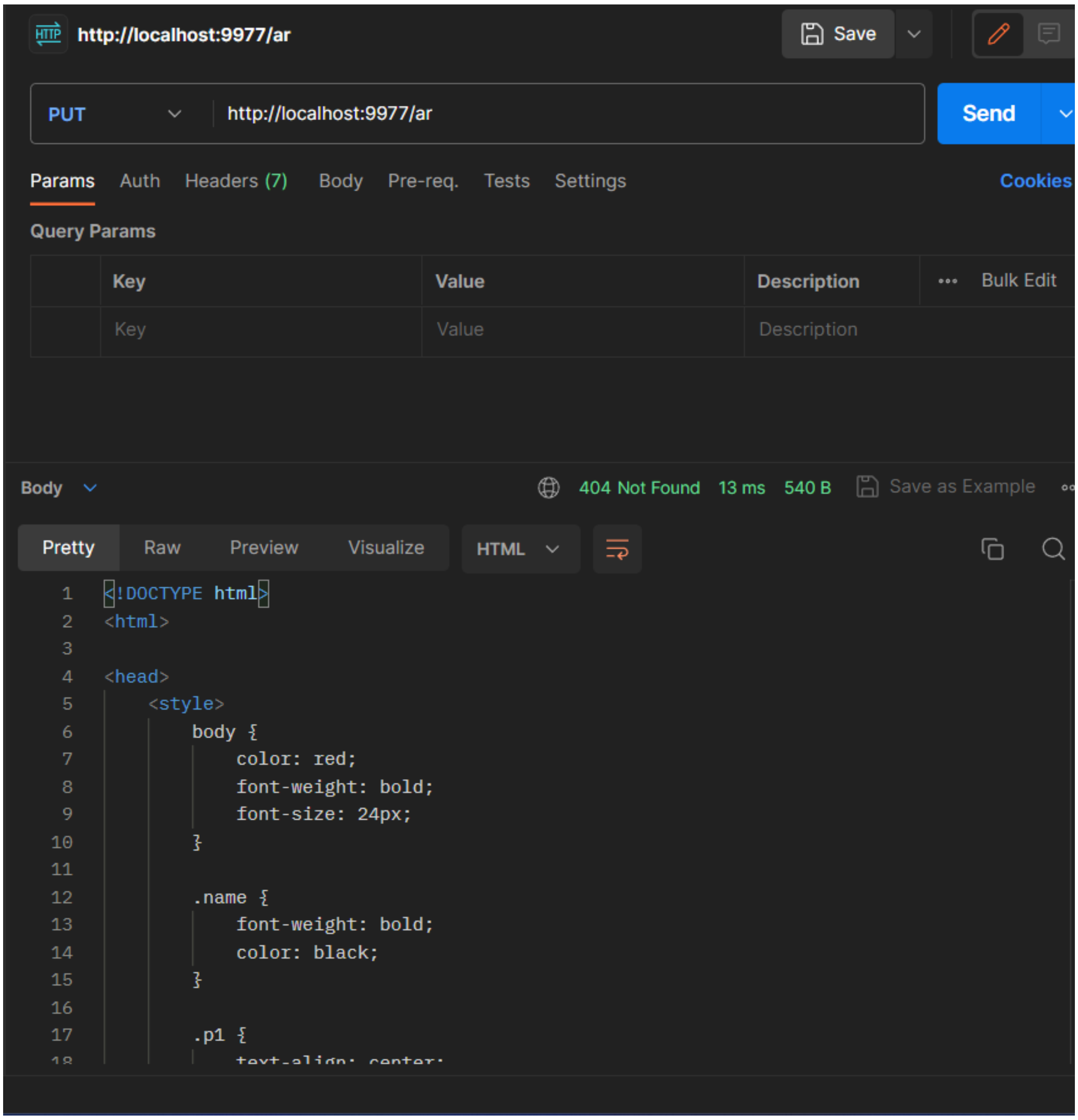
The file is not found !

Faris Abufarha, 1200546

Omar Badawi, 1202428

Nadeen Moreb, 1203437

# using PUT HTTP request with postman



http://localhost:9977/ar

PUT http://localhost:9977/ar

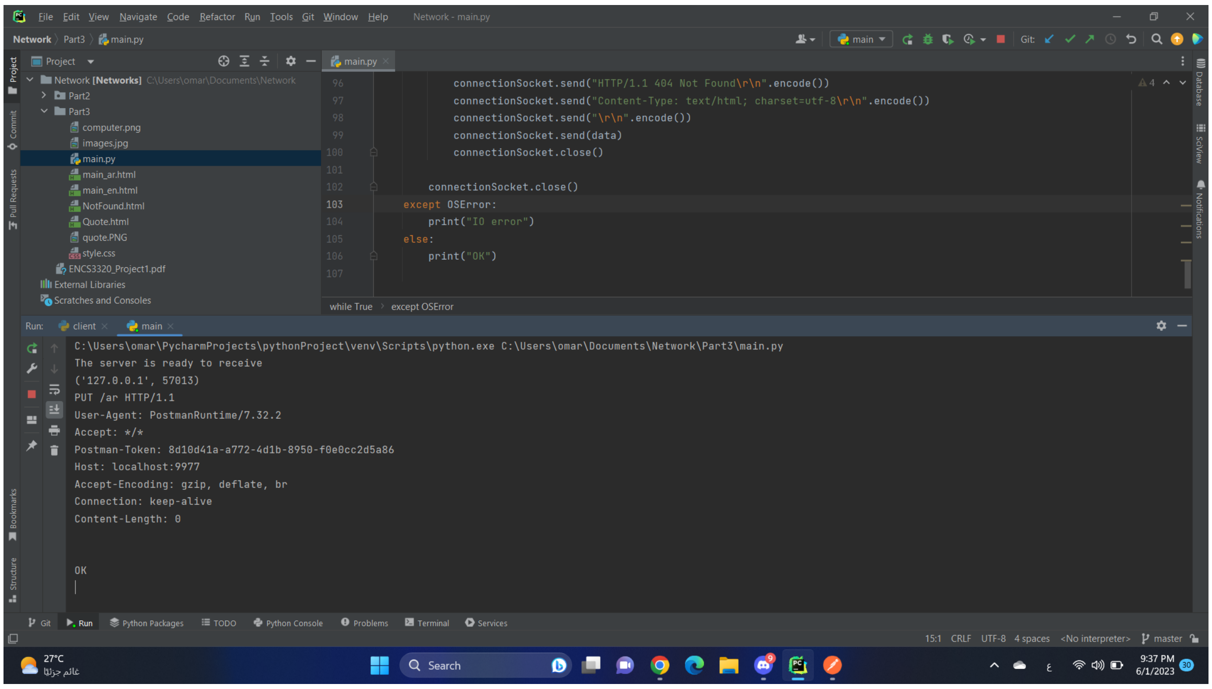
Query Params

Key	Value	Description
Key	Value	Description

Body 404 Not Found 13 ms 540 B

```
<!DOCTYPE html>
<html>

<head>
    <style>
        body {
            color: red;
            font-weight: bold;
            font-size: 24px;
        }

        .name {
            font-weight: bold;
            color: black;
        }

        .p1 {
            text-align: center;
```

# printing request details :



```python
                connectionSocket.send("HTTP/1.1 404 Not Found\r\n".encode())
                connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
                connectionSocket.send("\r\n".encode())
                connectionSocket.send(data)
                connectionSocket.close()

            connectionSocket.close()
        except OSError:
            print("IO error")
        else:
            print("OK")
```

```
C:\Users\omar\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\omar\Documents\Network\Part3\main.py
The server is ready to receive
('127.0.0.1', 57013)
PUT /ar HTTP/1.1
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Postman-Token: 8d10d41a-a772-4d1b-8950-f0e0cc2d5a86
Host: localhost:9977
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 0


OK
```