Eduvos
Your Education. Your Future.

Individual Assessment Coversheet

To be attached to the front of the assessment.

Campus: Midrand____

Faculty: Information Technology_______________________________

Module Code: ITPNA____

Group: 1____

Lecturer's Name: Mr Rametse ______________________________________

Student Full Name: Asimdumise Zwane___________________________________

Student Number: Eduv4935855______________________________________

Indicate	Yes	No
Plagiarism report attached		

Declaration:

I declare that this assessment is my own original work except for source material explicitly acknowledged. I also declare that this assessment or any other of my original work related to it has not been previously, or is not being simultaneously, submitted for this or any other course. I am aware of the AI policy and acknowledge that I have not used any AI technology to generate or manipulate data, other than as permitted by the assessment instructions. I also declare that I am aware of the Institution's policy and regulations on honesty in academic work as set out in the Conditions of Enrolment, and of the disciplinary guidelines applicable to breaches of such policy and regulations.

X Asimdumise Zwane Student **Signature**	**Date: 17/03/2025**

Lecturer's Comments:

Marks Awarded:	**%**

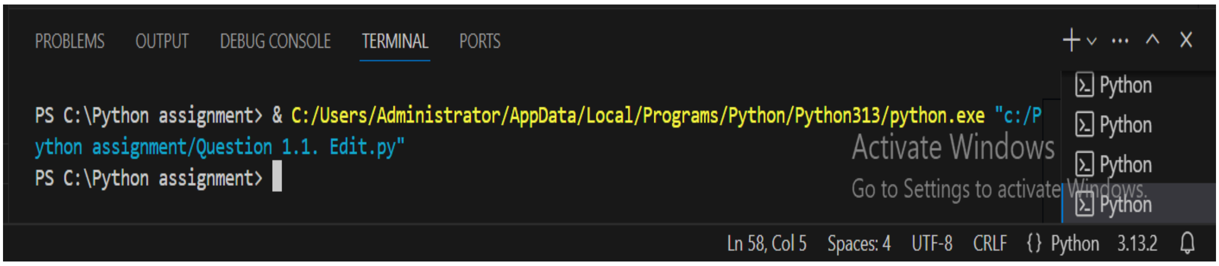
Signature	Date

Table of Contents

Question 1

1.1.

```
import socket
import ssl
import os

# Server configuration
HOST = '0.0.0.0'  # Listen on all available network interfaces
PORT = 8443       # Secure port
CERT_FILE = "certificate.pem"
KEY_FILE = "private.key"
SAVE_DIR = "received_files"

# Ensure save directory exists
os.makedirs(SAVE_DIR, exist_ok=True)

def handle_client(conn):
    """ Handles incoming file transfer from the client """
    try:
        # Receive file name length and name
        file_name_len = int.from_bytes(conn.recv(2), 'big')
        file_name = conn.recv(file_name_len).decode()

        # Receive file size
        file_size = int.from_bytes(conn.recv(8), 'big')

        # Save file
        file_path = os.path.join(SAVE_DIR, file_name)
        with open(file_path, 'wb') as f:
            received = 0
            while received < file_size:
                chunk = conn.recv(min(4096, file_size - received))
                if not chunk:
                    break
                f.write(chunk)
                received += len(chunk)

        print(f"File '{file_name}' received successfully.")
    except Exception as e:
        print(f"Error during file transfer: {e}")
    finally:
        conn.close()

def start_tls_server():
    """ Starts the TLS server for secure file transfers """
    context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain(certfile=CERT_FILE, keyfile=KEY_FILE)

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
server_socket:
        server_socket.bind((HOST, PORT))
        server_socket.listen(5)
        print(f"[*] Secure file transfer server listening on
{HOST}:{PORT}...")

        while True:
            client_socket, addr = server_socket.accept()
            print(f"[+] Connection from {addr}")
            with context.wrap_socket(client_socket, server_side=True) as
tls_conn:
                handle_client(tls_conn)

if "_name_" == "_main_":
    start_tls_server()
```



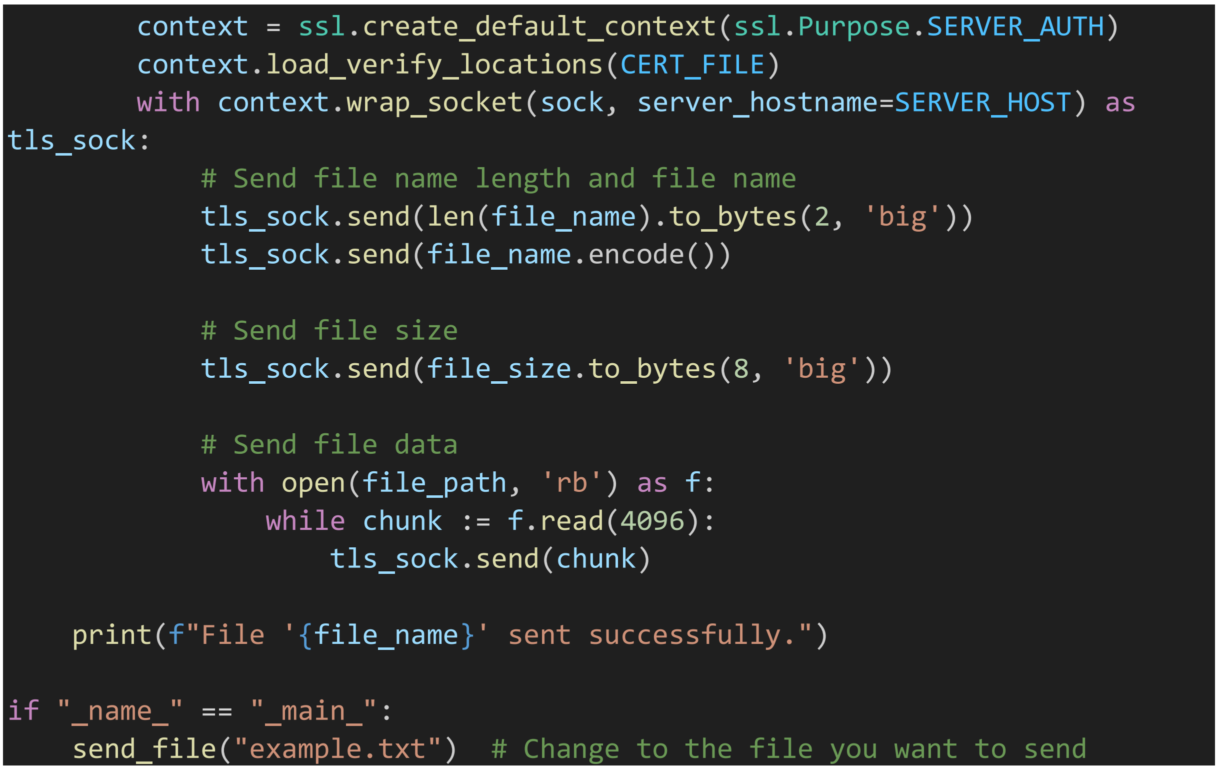
```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/P
ython assignment/Question 1.1. Edit.py"
PS C:\Python assignment> ▌
```

FIGURE 1 PROOF OF TLS TRANSFER SERVER

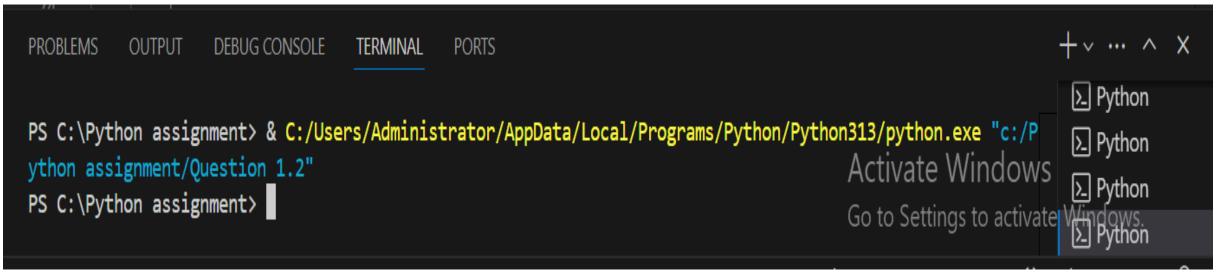
(Sean, 2023)

1.2.

```
import socket
import ssl
import os

SERVER_HOST = "your.server.ip"  # Change this to your server's IP
SERVER_PORT = 8443
CERT_FILE = "certificate.pem"

def send_file(file_path):
    """ Sends a file securely to the server """
    file_name = os.path.basename(file_path)
    file_size = os.path.getsize(file_path)

    with socket.create_connection((SERVER_HOST, SERVER_PORT)) as sock:
```



```
        context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
        context.load_verify_locations(CERT_FILE)
        with context.wrap_socket(sock, server_hostname=SERVER_HOST) as 
tls_sock:
            # Send file name length and file name
            tls_sock.send(len(file_name).to_bytes(2, 'big'))
            tls_sock.send(file_name.encode())

            # Send file size
            tls_sock.send(file_size.to_bytes(8, 'big'))

            # Send file data
            with open(file_path, 'rb') as f:
                while chunk := f.read(4096):
                    tls_sock.send(chunk)

    print(f"File '{file_name}' sent successfully.")

if "_name_" == "_main_":
    send_file("example.txt")  # Change to the file you want to send
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/P
ython assignment/Question 1.2"
PS C:\Python assignment> 
```

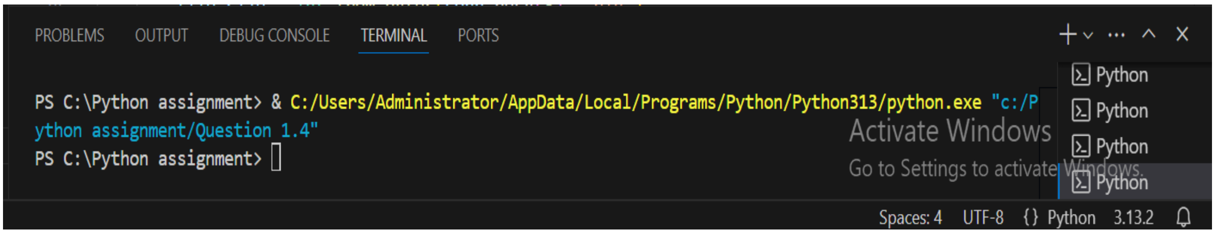
FIGURE 2 PROOF OF CLIENT IMPLEMENTATION

1.3. TLS encrypts the data to guarantee that the file content is private and uninvited parties cannot intercept it. It uses a hard-to-crack cryptographic algorithm that makes sure that no one other than the web server and web client can read or modify transmitted data. TLS guarantees that the data delivered and received are the same, preventing tampering. In order to prevent man-in-the-middle (MITM) attacks, TLS uses certificates to authenticate the client and server. TLS assists financial institutions in meeting regulatory standards for secure communication when handling sensitive data.

(Ubah, 2022)

1.4.

```python
import socket
import ssl
import os

# Server configuration
HOST = '0.0.0.0'
PORT = 8443
SERVER_CERT = "server.crt"
SERVER_KEY = "server.key"
CA_CERT = "ca.crt"
SAVE_DIR = "received_files"

# Ensure save directory exists
os.makedirs(SAVE_DIR, exist_ok=True)

def handle_client(conn):
    """Handles incoming file transfer from the client."""
    try:
        # Receive file name length and name
        file_name_len = int.from_bytes(conn.recv(2), 'big')
        file_name = conn.recv(file_name_len).decode()

        # Receive file size
        file_size = int.from_bytes(conn.recv(8), 'big')

        # Save file
        file_path = os.path.join(SAVE_DIR, file_name)
        with open(file_path, 'wb') as f:
            received = 0
            while received < file_size:
                chunk = conn.recv(min(4096, file_size - received))
                if not chunk:
                    break
                f.write(chunk)
                received += len(chunk)

        print(f"File '{file_name}' received successfully.")
    except Exception as e:
        print(f"Error during file transfer: {e}")
    finally:
        conn.close()

def start_tls_server():
    """Starts the TLS server with mutual authentication."""
    context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain(certfile=SERVER_CERT, keyfile=SERVER_KEY)
    context.load_verify_locations(CA_CERT)
    context.verify_mode = ssl.CERT_REQUIRED  # Enforce client authentication

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((HOST, PORT))
        server_socket.listen(5)
        print(f"[*] Secure file transfer server with mTLS listening on {HOST}:{PORT}...")

        while True:
            client_socket, addr = server_socket.accept()
            print(f"[+] Connection from {addr}")

            try:
                with context.wrap_socket(client_socket, server_side=True) as tls_conn:
                    print(f"[✓] Client authenticated: {tls_conn.getpeercert()}")
                    handle_client(tls_conn)
            except ssl.SSLError as e:
                print(f"[X] SSL Error: {e}")

if "_name_" == "_main_":
    start_tls_server()
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment/Question 1.4"
PS C:\Python assignment> 
```

FIGURE 3 SERVER THAT REQUIRES MUTUAL AUTHENTICATION

Question 2

2.1.

```
import asyncio
import dns.resolver
import time

# List of DNS servers to query
```

```
dns_servers = [
    "1.1.1.1",     # Cloudflare DNS
    "8.8.8.8",     # Google DNS
    "9.9.9.9",     # Quad9 DNS
    "208.67.222.222" # OpenDNS
]
```

```
domain = "example.com"  # Replace with the domain you want to query
```

```
async def query_dns(server, domain):
    resolver = dns.resolver.Resolver()
    resolver.nameservers = [server]
    start_time = time.time()
    try:
        answer = resolver.resolve(domain)
        response_time = time.time() - start_time
        return server, response_time, answer
    except Exception as e:
        return server, float('inf'), None
```

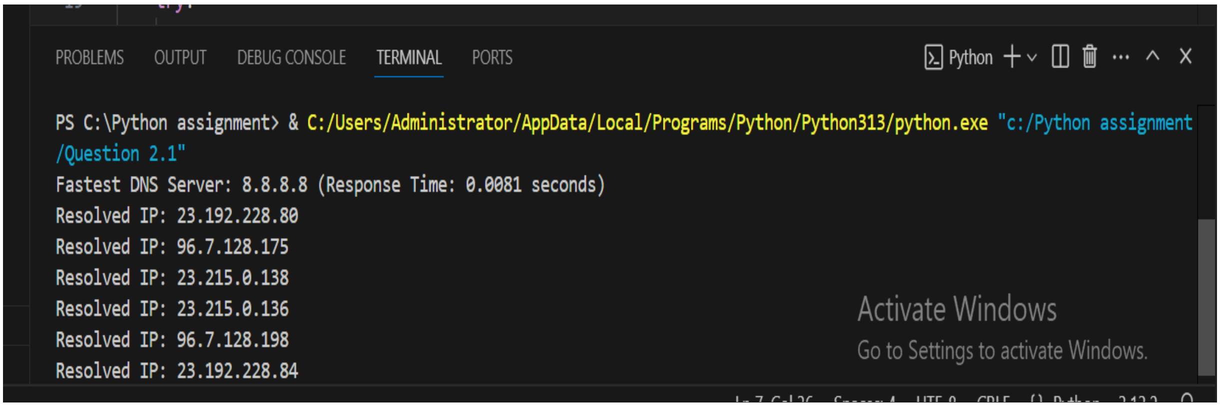
```
async def main():
    tasks = [query_dns(server, domain) for server in dns_servers]
    responses = await asyncio.gather(*tasks)
```

```
    # Choose the fastest response
    fastest = min(responses, key=lambda x: x[1])
```

```
    if fastest[2]:
            print(f"Resolved IP: {ip}")
    else:
        print("No successful response received.")
```

```
# Run the script
```

```
asyncio.run(main())
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment/Question 2.1"
Fastest DNS Server: 8.8.8.8 (Response Time: 0.0081 seconds)
Resolved IP: 23.192.228.80
Resolved IP: 96.7.128.175
Resolved IP: 23.215.0.138
Resolved IP: 23.215.0.136
Resolved IP: 96.7.128.198
Resolved IP: 23.192.228.84
```

FIGURE 4 FASTEST DNS SERVER

(Mosh, 2025)

2.2.

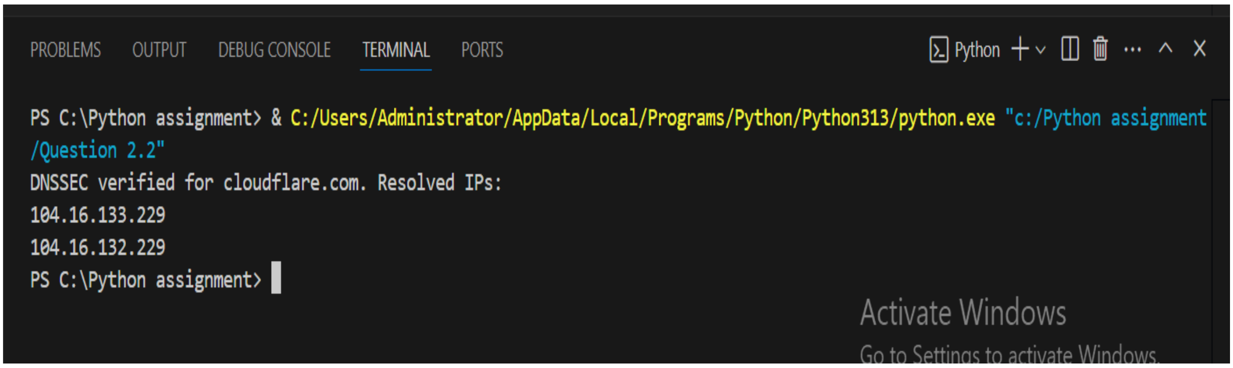
```
import dns.resolver

import dns.dnssec

import dns.name


def resolve_dnssec(domain):

    try:

        resolver = dns.resolver.Resolver()

        #Use Cloudflare DNS

        resolver.nameservers = ["1.1.1.1"]

        #Set maximum query duration to 30 seconds

        resolver.lifetime = 30

        #Set indinidual query timeout to 10 seconds

        resolver.timeout = 10

        # Query A record

        response = resolver.resolve(domain, "A")
```

```
        dnssec_response = resolver.resolve(domain, "DNSKEY")
```

```
        if dnssec_response:
            print(f"DNSSEC verified for {domain}. Resolved IPs:")
            for ip in response:
                print(ip)
        else:
            print(f"DNSSEC validation failed for {domain}. Possible security risk!")
```

```
    except Exception as e:
        print(f"DNS resolution error: {e}")
```

```
# Test with a DNSSEC-enabled domain
resolve_dnssec("cloudflare.com")
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment/Question 2.2"
DNSSEC verified for cloudflare.com. Resolved IPs:
104.16.133.229
104.16.132.229
PS C:\Python assignment>
```

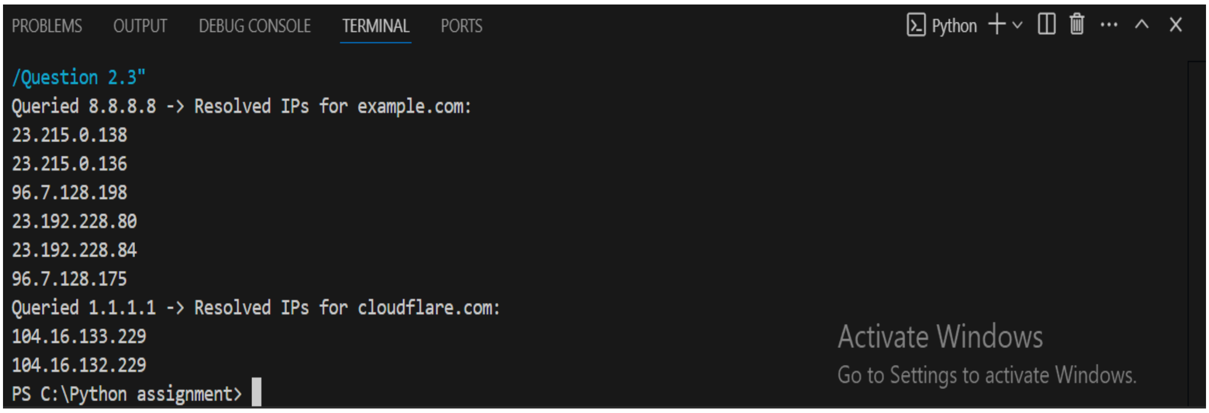
Figure 5 DNS resolution system

2.3.

```
import dns.resolver
import itertools


# List of DNS servers
dns_servers = itertools.cycle([
    "8.8.8.8", "1.1.1.1", "9.9.9.9", "208.67.222.222"
])
```

```
def resolve_with_load_balancing(domain):
    server = next(dns_servers)
    # Get the next DNS server in round-robin order
    resolver = dns.resolver.Resolver()
    resolver.nameservers = [server]

    try:
        response = resolver.resolve(domain, "A")
        print(f"Queried {server} -> Resolved IPs for {domain}:")
        for ip in response:
            print(ip)
    except Exception as e:
        print(f"Error querying {server}: {e}")
```

```
# Test resolution
resolve_with_load_balancing("example.com")
resolve_with_load_balancing("cloudflare.com")
```



```
/Question 2.3"
Queried 8.8.8.8 -> Resolved IPs for example.com:
23.215.0.138
23.215.0.136
96.7.128.198
23.192.228.80
23.192.228.84
96.7.128.175
Queried 1.1.1.1 -> Resolved IPs for cloudflare.com:
104.16.133.229
104.16.132.229
PS C:\Python assignment>
```

FIGURE 6 LOAD-BALANCING MECHANISM

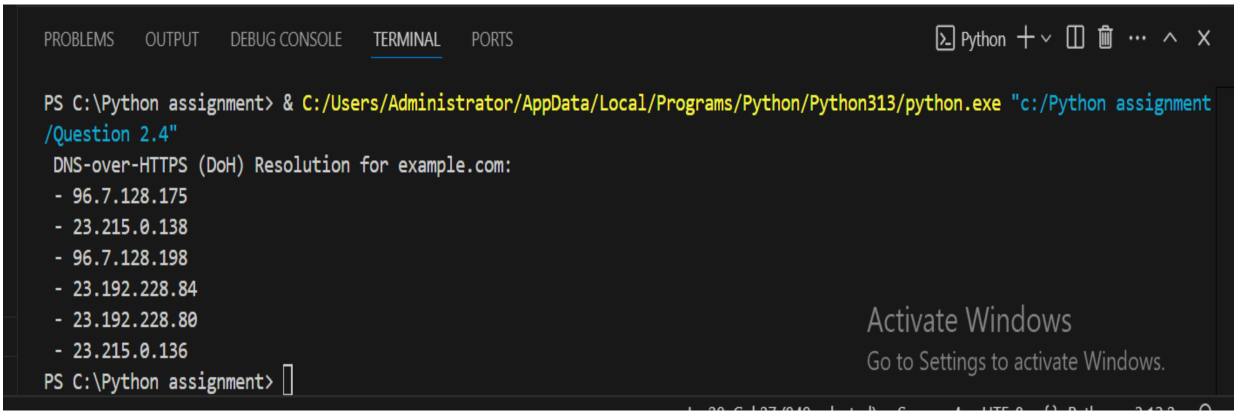
2.4.

```
import requests

def doh_resolve(domain):
    # Cloudflare DoH Server
    url = "https://cloudflare-dns.com/dns-query"
    headers = {"accept": "application/dns-json"}
    # Query A record
    params = {"name": domain, "type": "A"}
```

```
    try:
        response = requests.get(url, headers=headers, params=params, timeout=10)
        response.raise_for_status()

        data = response.json()

        if "Answer" in data:
            print(f" DNS-over-HTTPS (DoH) Resolution for {domain}:")
            for answer in data["Answer"]:
                print(f" - {answer['data']}")
        else:
            print(f" No DNS records found for {domain}.")
```

```
    except requests.exceptions.Timeout:
        print(f" Error: DoH request timed out for {domain}.")
    except requests.exceptions.RequestException as e:
        print(f" DoH resolution error: {e}")
```

```
# Test with a domain
doh_resolve("example.com")
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment
/Question 2.4"
 DNS-over-HTTPS (DoH) Resolution for example.com:
 - 96.7.128.175
 - 23.215.0.138
 - 96.7.128.198
 - 23.192.228.84
 - 23.192.228.80
 - 23.215.0.136
PS C:\Python assignment> 
```

FIGURE 7 DNS-OVER-HTTPS RESOLUTION

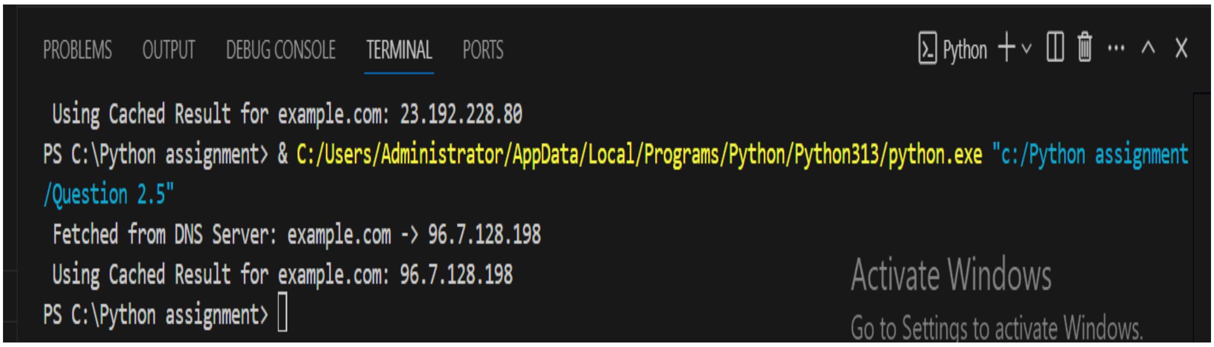
2.5.

```python
import dns.resolver
import time

# Dictionary to store cached DNS results (with expiration)
dns_cache = {}

# Cache timeout (seconds)
CACHE_EXPIRATION = 60

def resolve_dns(domain):
    current_time = time.time()

    # Check if the domain is already cached and still valid
    if domain in dns_cache:
        cached_entry = dns_cache[domain]
        if current_time - cached_entry["timestamp"] < CACHE_EXPIRATION:
            print(f" Using Cached Result for {domain}: {cached_entry['ip']}")
            return cached_entry["ip"]
        else:
            print(f" Cache Expired for {domain}, re-querying...")

    try:
        resolver = dns.resolver.Resolver()
        # Use Google's DNS
        resolver.nameservers = ["8.8.8.8"]
        # Query A record
        answer = resolver.resolve(domain, "A")

        # Get the first IP from the answer
        ip_address = answer[0].to_text()
        # Cache the result with a timestamp
        dns_cache[domain] = {"ip": ip_address, "timestamp": current_time}
```

```
        print(f" Fetched from DNS Server: {domain} -> {ip_address}")
        return ip_address
```

```
    except dns.resolver.NoAnswer:
        print(f" No DNS records found for {domain}.")
    except dns.resolver.NXDOMAIN:
        print(f" Error: Domain {domain} does not exist.")
    except dns.resolver.Timeout:
        print(f" Error: DNS query for {domain} timed out.")
    except Exception as e:
        print(f" DNS resolution error: {e}")
```

```
# Example Usage
resolve_dns("example.com")
time.sleep(2)  # Simulate some delay
resolve_dns("example.com")
```



```
 Using Cached Result for example.com: 23.192.228.80
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment
/Question 2.5"
 Fetched from DNS Server: example.com -> 96.7.128.198
 Using Cached Result for example.com: 96.7.128.198
PS C:\Python assignment>
```

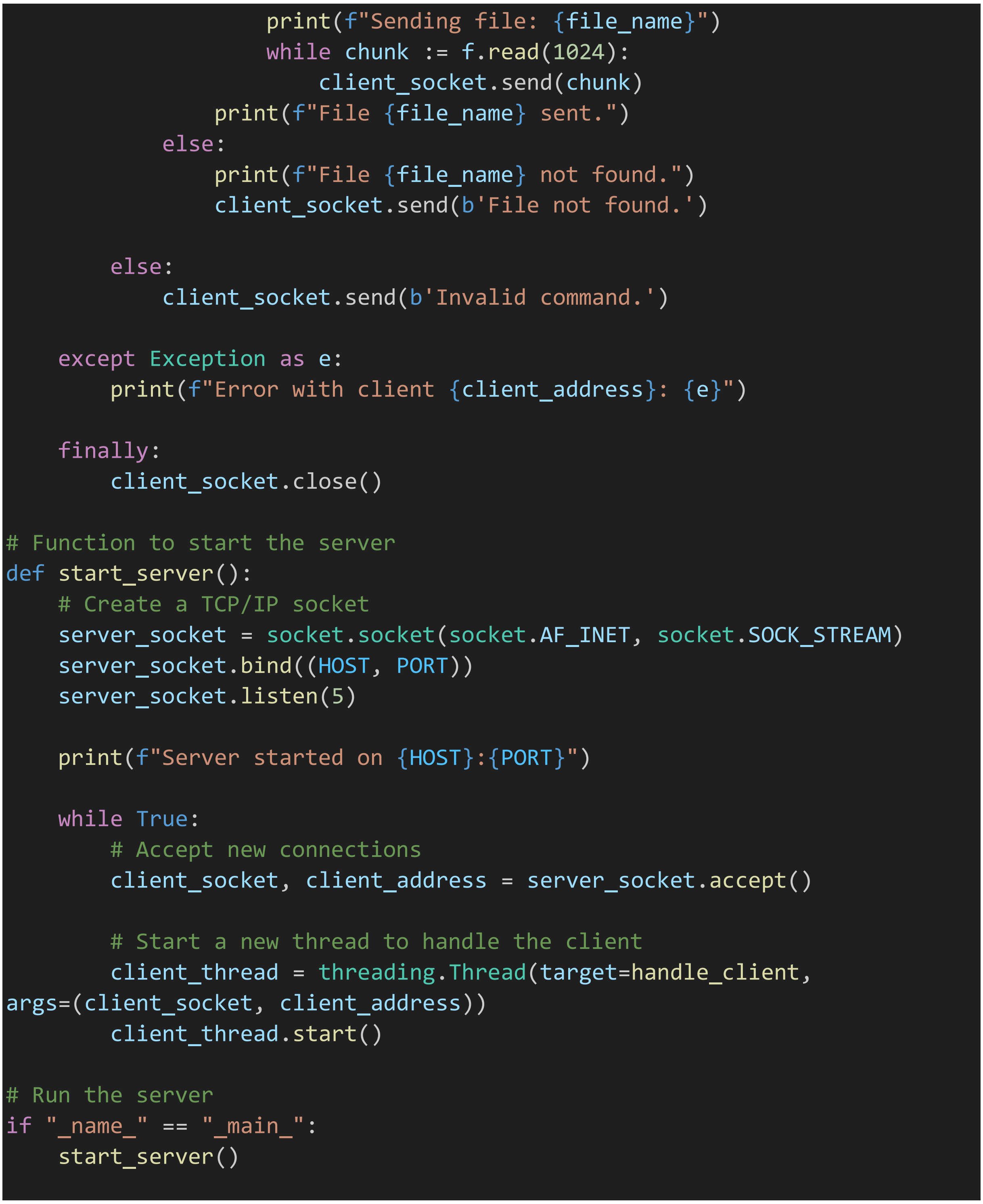
Figure 8 DNS Query Caching

(Insider, 2025)

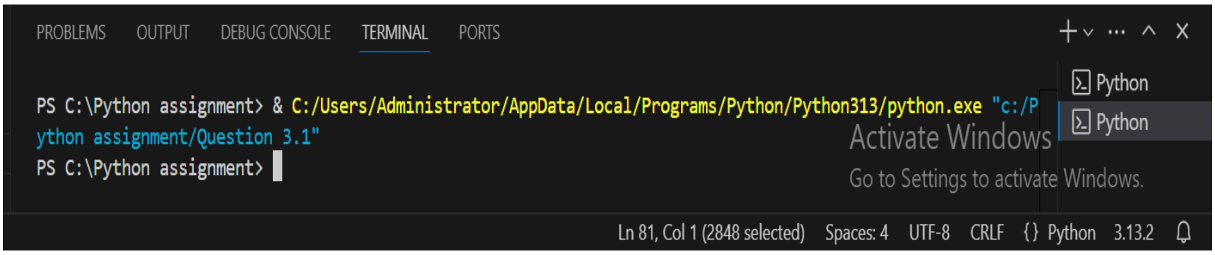
Question 3

3.1.

```
import socket
import threading
import os

# Define the host and port
HOST = '127.0.0.1'  # Localhost
PORT = 12345        # Port to bind the server to

# Function to handle client communication
def handle_client(client_socket, client_address):
    print(f"New connection from {client_address}")

    try:
        # Receiving the command from the client (either 'send' or
'request')
        command = client_socket.recv(1024).decode('utf-8')

        if command == 'send':
            # Receive file name
            file_name = client_socket.recv(1024).decode('utf-8')
            file_path = os.path.join('received_files', file_name)

            # Create directory to store received files
            if not os.path.exists('received_files'):
                os.mkdir('received_files')

            # Open the file to write the data received
            with open(file_path, 'wb') as f:
                print(f"Receiving file: {file_name}")
                while True:
                    data = client_socket.recv(1024)
                    if not data:
                        break
                    f.write(data)
                print(f"File {file_name} received and saved.")

        elif command == 'request':
            # Receive the requested file name
            file_name = client_socket.recv(1024).decode('utf-8')
            file_path = os.path.join('received_files', file_name)

            if os.path.exists(file_path):
                # Send the file to the client
                with open(file_path, 'rb') as f:
```



```
                    print(f"Sending file: {file_name}")
                    while chunk := f.read(1024):
                        client_socket.send(chunk)
                print(f"File {file_name} sent.")
            else:
                print(f"File {file_name} not found.")
                client_socket.send(b'File not found.')

        else:
            client_socket.send(b'Invalid command.')

    except Exception as e:
        print(f"Error with client {client_address}: {e}")

    finally:
        client_socket.close()

# Function to start the server
def start_server():
    # Create a TCP/IP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((HOST, PORT))
    server_socket.listen(5)

    print(f"Server started on {HOST}:{PORT}")

    while True:
        # Accept new connections
        client_socket, client_address = server_socket.accept()

        # Start a new thread to handle the client
        client_thread = threading.Thread(target=handle_client,
args=(client_socket, client_address))
        client_thread.start()

# Run the server
if "_name_" == "_main_":
    start_server()
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/P
ython assignment/Question 3.1"
PS C:\Python assignment> █
```

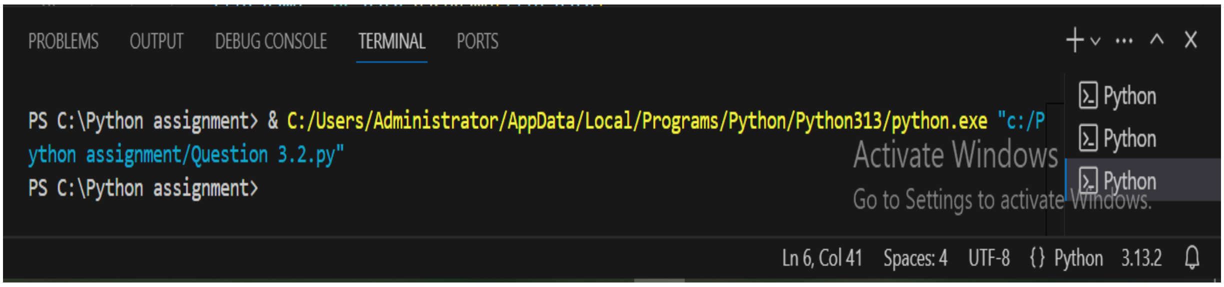
FIGURE 9 TCP SERVER USING MULTITHREADING

(Editor, 2024)

3.2.

```
import socket
import os

# Define the server address and port
SERVER_HOST = '127.0.0.1'  # Server address (localhost)
SERVER_PORT = 12345        # Server port

# Function to upload a file to the server
def upload_file(file_path):
    try:
        # Check if the file exists
        if not os.path.isfile(file_path):
            print(f"File '{file_path}' does not exist.")
            return

        # Connect to the server
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
client_socket:
            client_socket.connect((SERVER_HOST, SERVER_PORT))

            # Send 'send' command to notify the server we are sending a
file
            client_socket.send(b'send')

            # Send the file name
            file_name = os.path.basename(file_path)
            client_socket.send(file_name.encode('utf-8'))

            # Send the file content in chunks
            with open(file_path, 'rb') as file:
                while chunk := file.read(1024):
                    client_socket.send(chunk)

            print(f"File '{file_name}' uploaded successfully.")

    except Exception as e:
        print(f"An error occurred while uploading the file: {e}")

# Function to request a file from the server
def request_file(file_name):
    try:
        # Connect to the server
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
client_socket:
```

```
            client_socket.connect((SERVER_HOST, SERVER_PORT))

            # Send 'request' command to notify the server we want to
request a file
            client_socket.send(b'request')

            # Send the file name to request
            client_socket.send(file_name.encode('utf-8'))

            # Receive the file from the server and save it
            with open(f'received_{file_name}', 'wb') as file:
                while chunk := client_socket.recv(1024):
                    if chunk == b'File not found.':
                        print(f"File '{file_name}' not found on the
server.")
                        return
                    file.write(chunk)

            print(f"File '{file_name}' downloaded successfully.")

    except Exception as e:
        print(f"An error occurred while requesting the file: {e}")

# Main function for user interaction
def main():
    while True:
        print("\nOptions:")
        print("1. Upload a file to the server")
        print("2. Request a file from the server")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ").strip()

        if choice == '1':
            file_path = input("Enter the file path to upload: ").strip()
            upload_file(file_path)
        elif choice == '2':
            file_name = input("Enter the file name to request: ").strip()
            request_file(file_name)
        elif choice == '3':
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please try again.")

# Run the client
if "_name_" == "_main_":
    main()
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/Python assignment/Question 3.2.py"
PS C:\Python assignment>
```

FIGURE 10 TCP CLIENT

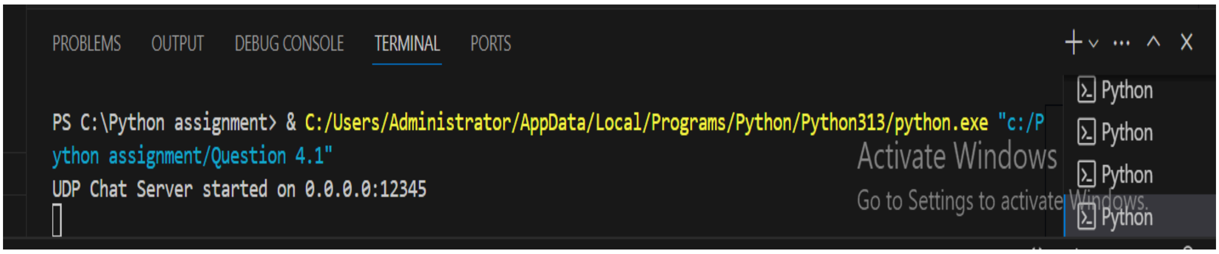
(Labs, 2024)

Question 4

4.1.

```
import socket

# Server Configuration
SERVER_IP = "0.0.0.0"
SERVER_PORT = 12345
BUFFER_SIZE = 1024

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((SERVER_IP, SERVER_PORT))

print(f"UDP Chat Server started on {SERVER_IP}:{SERVER_PORT}")

clients = set()  # Store client addresses

while True:
    message, client_address = server_socket.recvfrom(BUFFER_SIZE)

    if client_address not in clients:
        clients.add(client_address)

    print(f"Received message from {client_address}: {message.decode()}")

    # Broadcast the message to all connected clients
    for client in clients:
        if client != client_address:  # Don't send message back to sender
```

```
            server_socket.sendto(message, client)
```



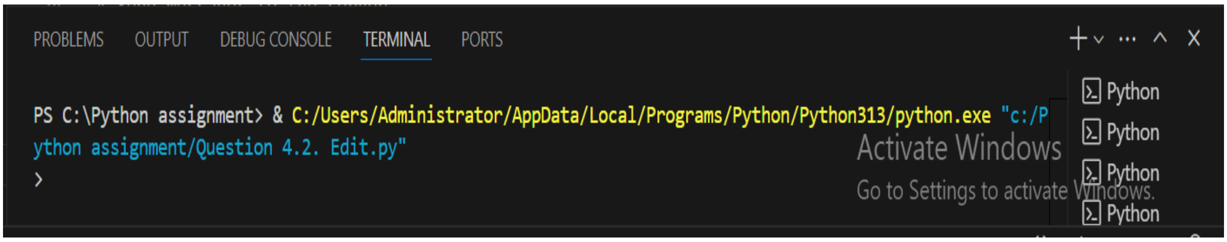
```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/P
ython assignment/Question 4.1"
UDP Chat Server started on 0.0.0.0:12345
```

FIGURE 11 UDP SERVER

(Alsop, 2024)

4.2.

```
import socket
import threading

# Server Configuration
SERVER_IP = "127.0.0.1"
SERVER_PORT = 12345
BUFFER_SIZE = 1024

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Function to receive messages from the server
def receive_messages():
    while True:
        try:
            message, _ = client_socket.recvfrom(BUFFER_SIZE)
            print("\n" + message.decode() + "\n> ", end="")
        except:
            break

# Start a thread for receiving messages
threading.Thread(target=receive_messages, daemon=True).start()

# Send messages to the server
while True:
    message = input("> ")
    if message.lower() == "exit":
        break
    client_socket.sendto(message.encode(), (SERVER_IP, SERVER_PORT))

client_socket.close()
```



```
PS C:\Python assignment> & C:/Users/Administrator/AppData/Local/Programs/Python/Python313/python.exe "c:/P
ython assignment/Question 4.2. Edit.py"
>
```

FIGURE 12 DNS CLIENT

Works Cited

Alsop, R. (2024, June 24). *Information Security Stack Exchange*. Retrieved from Securing HTTP File Transfer: https://security.stackexchange.com

Editor, G. (2024, December 27). *GeeksforGeeks*. Retrieved from https://www.geeksforgeeks.org

Insider, P. (2025, March 12). *Python.org*. Retrieved from https://www.python.org

Labs, P. (2024, October 25). *Programiz*. Retrieved from Python File Operation : https://www.programiz.com

Mosh. (2025, February 12). *Programming with Mosh*. Retrieved from Youtube: https://m.youtube.com/watch?v=K5KVEU3aaeQ&t=614s&pp=2AHmBJACAQ%3D%3D

Sean. (2023, July 23). *Debugbar*. Retrieved from How to implement TLS/SSL in python?: https://www.debugbar.com

Ubah, K. (2022, October 16). *Implementing TLS/SSL in Python*. Retrieved from https://snyk.io