# **Name: Abdurrahman Qureshi**

# **Roll No: 242466**

Practical No: 8

**1) Write a python program to send details form server to client.**

CLIENT CODE:

import socket

import pickle

s = socket.socket()

try:

    s.connect(("localhost",3000))

    data = s.recv(1024)

    data\_dict = pickle.loads(data)

    print(data\_dict)

    s.close()

except:

    print("Something went wrong")

SERVER CODE:

import socket

import pickle

details = {

        "device\_name": "GhouledGadget",

        "device\_model": "Dell Inspiron 15",

        "ram": "24GB",

        "ssd": "512GB"

    }

serialized\_details = pickle.dumps(details)

s = socket.socket()

try:

    s.bind(("localhost",3000))

    print("Socket binded to port 3000")

    s.listen(5)

    response, address = s.accept()

    print(f"Socket connection established with address {address}")

    response.sendall(serialized\_details)

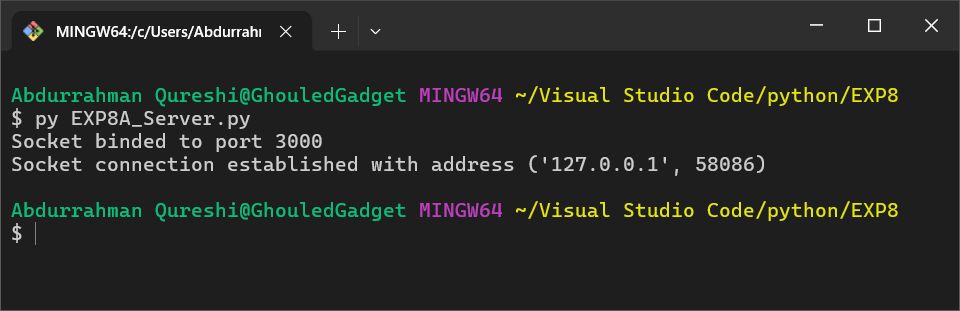
    response.close()

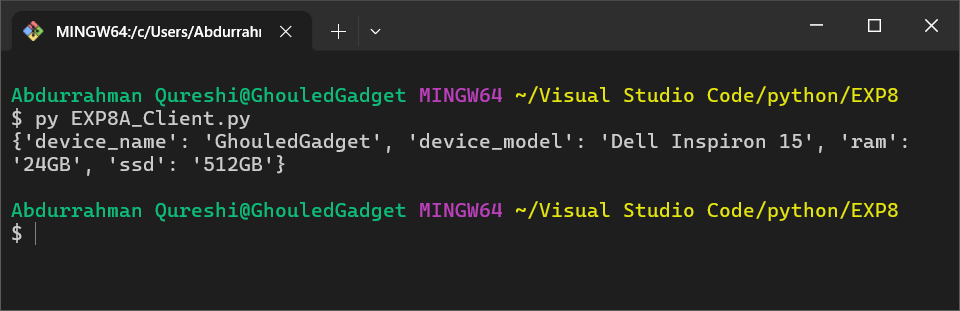
    s.close()

except:

    print("Something went wrong")

OUTPUT:





**2) Write a python program to have a two-way communication**

CLIENT CODE:

import socket

import pickle

s = socket.socket()

try:

    s.connect(("localhost", 3000))

    data = s.recv(1024)

    data\_dict = pickle.loads(data)

    print("Received device details from server:", data\_dict)

    response\_message = "Device details received successfully!"

    s.sendall(response\_message.encode())

    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

SERVER CODE:

import socket

import pickle

details = {

        "device\_name": "GhouledGadget",

        "device\_model": "Dell Inspiron 15",

        "ram": "24GB",

        "ssd": "512GB"

    }

serialized\_details = pickle.dumps(details)

s = socket.socket()

try:

    s.bind(("localhost",3000))

    print("Socket binded to port 3000")

    s.listen(5)

    response, address = s.accept()

    print(f"Socket connection established with address {address}")

    response.sendall(serialized\_details)

    print("Device details sent to the client")

    client\_response = response.recv(1024)

    print(f"Received response from client: {client\_response.decode()}")

    response.close()

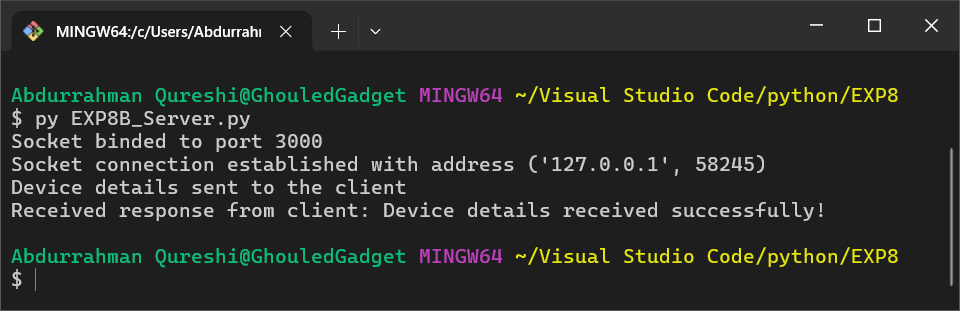
    s.close()

except:

    print("Something went wrong")

OUTPUT:





**3) Write a python program to chat between server and client and end the chat when wither says “over”**

CLIENT CODE:

import socket

s = socket.socket()

try:

    s.connect(("localhost", 3000))

    while True:

        server\_msg = s.recv(1024).decode()

        if server\_msg.lower() == "over":

            print("Server has ended the conversation.")

            break

        print(f"Server says: {server\_msg}")

        client\_msg = input("Client, enter your message: ")

        s.send(client\_msg.encode())

        if client\_msg.lower() == "over":

            print("Client is closing the connection.")

            break

    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

SERVER CODE:

import socket

s = socket.socket()

try:

    s.bind(("localhost", 3000))

    print("Socket binded to port 3000")

    s.listen(5)

    response, address = s.accept()

    print(f"Socket connection established with address {address}")

    while True:

        server\_msg = input("Server, enter your message: ")

        response.send(server\_msg.encode())

        if server\_msg.lower() == "over":

            print("Server is closing the connection.")

            break

        client\_msg = response.recv(1024).decode()

        if client\_msg.lower() == "over":

            print("Client has ended the conversation.")

            break

        print(f"Client says: {client\_msg}")

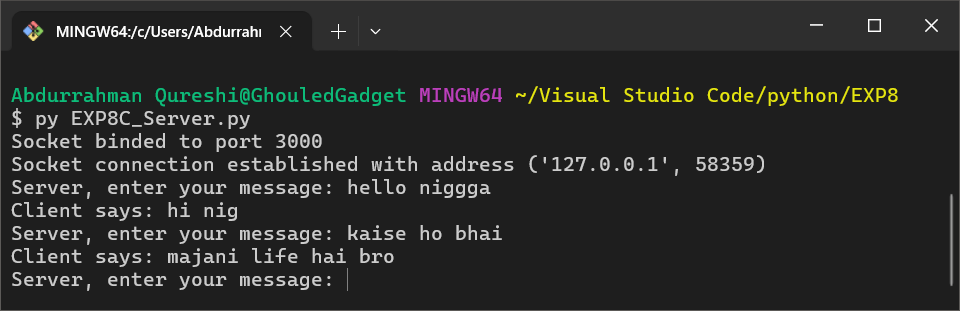
    response.close()

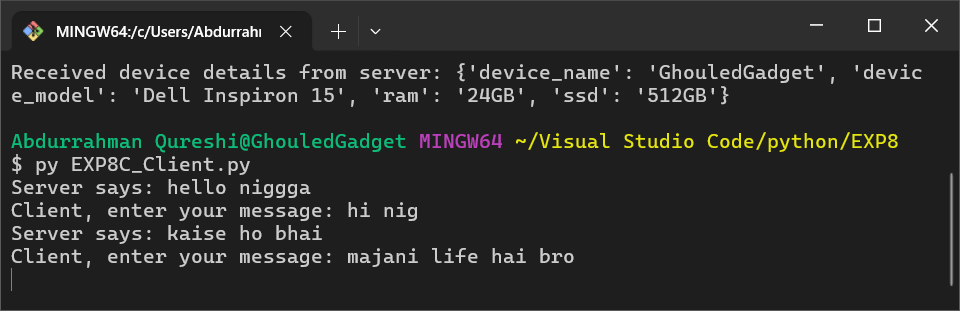
    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

OUTPUT:





**4) Write a python program to send file to sever from client.**

CLIENT CODE:

import socket

s = socket.socket()

try:

    s.connect(("localhost", 3000))

    img = open("Lady D.png", "rb")

    img.close()

    s.sendall(img)

    print("Image sent to the server")

    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

SERVER CODE:

import socket

s = socket.socket()

try:

    s.bind(("localhost", 3000))

    print("Socket binded to port 3000")

    s.listen(5)

    response, address = s.accept()

    print(f"Socket connection established with address {address}")

    client\_response = response.recv(4096)

    print(f"Received response from client:")

    print(client\_response)

    response.close()

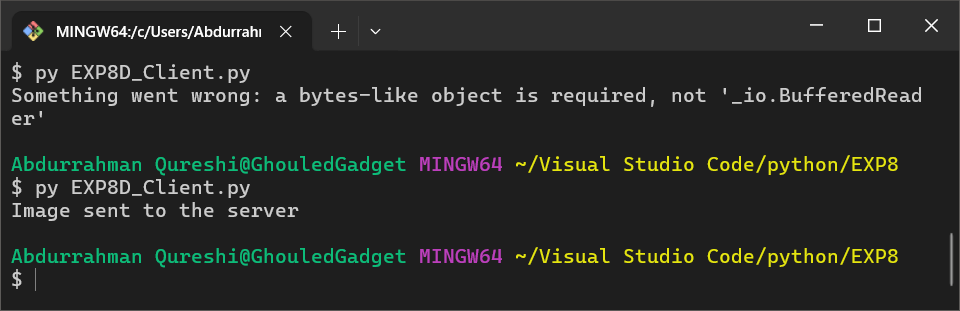
    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

OUTPUT:





**5) Write a python program to secure our sever-client chat.**

CLIENT CODE:

import socket

s = socket.socket()

ciphered = {

*#* UPPERCASE LETTERS

    "A": "Xq7", "B": "Lm3", "C": "Pn9", "D": "Rk2", "E": "Yj5", "F": "Tb8", "G": "Hw1", "H": "Uv4", "I": "Zs6", "J": "Oc0",

    "K": "Qa9", "L": "Wd3", "M": "Ei7", "N": "Fy2", "O": "Gx5", "P": "Ju8", "Q": "Kl1", "R": "Vn4", "S": "Bm6", "T": "Cp0",

    "U": "Az9", "V": "Sr3", "W": "Dt7", "X": "Mh2", "Y": "Nf5", "Z": "Ig8",

*#* LOWERCASE LETTERS

    "a": "7qX", "b": "3mL", "c": "9nP", "d": "2kR", "e": "5jY", "f": "8bT", "g": "1wH", "h": "4vU", "i": "6sZ", "j": "0cO",

    "k": "9aQ", "l": "3dW", "m": "7iE", "n": "2yF", "o": "5xG", "p": "8uJ", "q": "1lK", "r": "4nV", "s": "6mB", "t": "0pC",

    "u": "9zA", "v": "3rS", "w": "7tD", "x": "2hM", "y": "5fN", "z": "8gI",

*#* NUMBERS

    "0": "Xq7", "1": "Lm3", "2": "Pn9", "3": "Rk2", "4": "Yj5", "5": "Tb8", "6": "Hw1", "7": "Uv4", "8": "Zs6", "9": "Oc0",

*#* SPECIAL CHARACTERS

    " ": "Xq7", ",": "Lm3", ".": "Pn9", "!": "Rk2", "?": "Yj5", "'": "Tb8", ":": "Hw1", ";": "Uv4", "/": "Zs6", "-": "Oc0",

}

deciphered = {

*#* UPPERCASE LETTERS

    "Xq7": "A", "Lm3": "B", "Pn9": "C", "Rk2": "D", "Yj5": "E", "Tb8": "F", "Hw1": "G", "Uv4": "H", "Zs6": "I", "Oc0": "J",

    "Qa9": "K", "Wd3": "L", "Ei7": "M", "Fy2": "N", "Gx5": "O", "Ju8": "P", "Kl1": "Q", "Vn4": "R", "Bm6": "S", "Cp0": "T",

    "Az9": "U", "Sr3": "V", "Dt7": "W", "Mh2": "X", "Nf5": "Y", "Ig8": "Z",

*#* LOWERCASE LETTERS

    "7qX": "a", "3mL": "b", "9nP": "c", "2kR": "d", "5jY": "e", "8bT": "f", "1wH": "g", "4vU": "h", "6sZ": "i", "0cO": "j",

    "9aQ": "k", "3dW": "l", "7iE": "m", "2yF": "n", "5xG": "o", "8uJ": "p", "1lK": "q", "4nV": "r", "6mB": "s", "0pC": "t",

    "9zA": "u", "3rS": "v", "7tD": "w", "2hM": "x", "5fN": "y", "8gI": "z",

*#* NUMBERS

    "Xq7": "0", "Lm3": "1", "Pn9": "2", "Rk2": "3", "Yj5": "4", "Tb8": "5", "Hw1": "6", "Uv4": "7", "Zs6": "8", "Oc0": "9",

*#* SPECIAL CHARACTERS

    "Xq7": " ", "Lm3": ",", "Pn9": ".", "Rk2": "!", "Yj5": "?", "'": "'", ":": ":", "/": "/", "-": "\_",

    "Qa9": "+", "=": "[", "]": "]", "{": "{", "}": "}", "<": "<", ">": ">", "|": "|"  *#* The ciphered characters are reversed for deciphering

}

def polyalphabetic\_cipher(input, type):

    result = ""

    if type == True:

        for char in input:

            result += ciphered[char]

    else:

        trinity = [(input[i:i+3]) for i in range(0, len(input), 3)]

        for char in trinity:

            result += deciphered[char]

        print(trinity)

    return result

try:

    s.connect(("localhost", 3000))

    while True:

        server\_msg = s.recv(1024).decode()

        if server\_msg.lower() == "over":

            print("Server has ended the conversation.")

            break

        print(f"Server says: {polyalphabetic\_cipher(server\_msg, False)}")

        print(f"Server says: {server\_msg}")

        client\_msg = input("Client, enter your message: ")

        s.send(polyalphabetic\_cipher(client\_msg,True).encode())

        if client\_msg.lower() == "over":

            print("Client is closing the connection.")

            break

    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

SERVER CODE:

import socket

s = socket.socket()

ciphered = {

*#* UPPERCASE LETTERS

    "A": "Xq7", "B": "Lm3", "C": "Pn9", "D": "Rk2", "E": "Yj5", "F": "Tb8", "G": "Hw1", "H": "Uv4", "I": "Zs6", "J": "Oc0",

    "K": "Qa9", "L": "Wd3", "M": "Ei7", "N": "Fy2", "O": "Gx5", "P": "Ju8", "Q": "Kl1", "R": "Vn4", "S": "Bm6", "T": "Cp0",

    "U": "Az9", "V": "Sr3", "W": "Dt7", "X": "Mh2", "Y": "Nf5", "Z": "Ig8",

*#* LOWERCASE LETTERS

    "a": "7qX", "b": "3mL", "c": "9nP", "d": "2kR", "e": "5jY", "f": "8bT", "g": "1wH", "h": "4vU", "i": "6sZ", "j": "0cO",

    "k": "9aQ", "l": "3dW", "m": "7iE", "n": "2yF", "o": "5xG", "p": "8uJ", "q": "1lK", "r": "4nV", "s": "6mB", "t": "0pC",

    "u": "9zA", "v": "3rS", "w": "7tD", "x": "2hM", "y": "5fN", "z": "8gI",

*#* NUMBERS

    "0": "Xq7", "1": "Lm3", "2": "Pn9", "3": "Rk2", "4": "Yj5", "5": "Tb8", "6": "Hw1", "7": "Uv4", "8": "Zs6", "9": "Oc0",

*#* SPECIAL CHARACTERS

    " ": "Xq7", ",": "Lm3", ".": "Pn9", "!": "Rk2", "?": "Yj5", "'": "Tb8", ":": "Hw1", ";": "Uv4", "/": "Zs6", "-": "Oc0",

}

deciphered = {

*#* UPPERCASE LETTERS

    "Xq7": "A", "Lm3": "B", "Pn9": "C", "Rk2": "D", "Yj5": "E", "Tb8": "F", "Hw1": "G", "Uv4": "H", "Zs6": "I", "Oc0": "J",

    "Qa9": "K", "Wd3": "L", "Ei7": "M", "Fy2": "N", "Gx5": "O", "Ju8": "P", "Kl1": "Q", "Vn4": "R", "Bm6": "S", "Cp0": "T",

    "Az9": "U", "Sr3": "V", "Dt7": "W", "Mh2": "X", "Nf5": "Y", "Ig8": "Z",

*#* LOWERCASE LETTERS

    "7qX": "a", "3mL": "b", "9nP": "c", "2kR": "d", "5jY": "e", "8bT": "f", "1wH": "g", "4vU": "h", "6sZ": "i", "0cO": "j",

    "9aQ": "k", "3dW": "l", "7iE": "m", "2yF": "n", "5xG": "o", "8uJ": "p", "1lK": "q", "4nV": "r", "6mB": "s", "0pC": "t",

    "9zA": "u", "3rS": "v", "7tD": "w", "2hM": "x", "5fN": "y", "8gI": "z",

*#* NUMBERS

    "Xq7": "0", "Lm3": "1", "Pn9": "2", "Rk2": "3", "Yj5": "4", "Tb8": "5", "Hw1": "6", "Uv4": "7", "Zs6": "8", "Oc0": "9",

*#* SPECIAL CHARACTERS

    "Xq7": " ", "Lm3": ",", "Pn9": ".", "Rk2": "!", "Yj5": "?", "'": "'", ":": ":", "/": "/", "-": "\_",

    "Qa9": "+", "=": "[", "]": "]", "{": "{", "}": "}", "<": "<", ">": ">", "|": "|"  *#* The ciphered characters are reversed for deciphering

}

def polyalphabetic\_cipher(input, type):

    result = ""

    if type == True:

        for char in input:

            result += ciphered[char]

    else:

        trinity = [(input[i:i+3]) for i in range(0, len(input), 3)]

        for char in trinity:

            result += deciphered[char]

        print(trinity)

    return result

try:

    s.connect(("localhost", 3000))

    while True:

        server\_msg = s.recv(1024).decode()

        if server\_msg.lower() == "over":

            print("Server has ended the conversation.")

            break

        print(f"Server says: {polyalphabetic\_cipher(server\_msg, False)}")

        print(f"Server says: {server\_msg}")

        client\_msg = input("Client, enter your message: ")

        s.send(polyalphabetic\_cipher(client\_msg,True).encode())

        if client\_msg.lower() == "over":

            print("Client is closing the connection.")

            break

    s.close()

except Exception as e:

    print(f"Something went wrong: {e}")

OUTPUT:

