CMPE 472 – Computer Networks

Programming Assignment 1

Mustafa Vanlıoğlu

35462114736

Section: 01

Submission date

23.03.2025

Table of Contents

1. Introduction

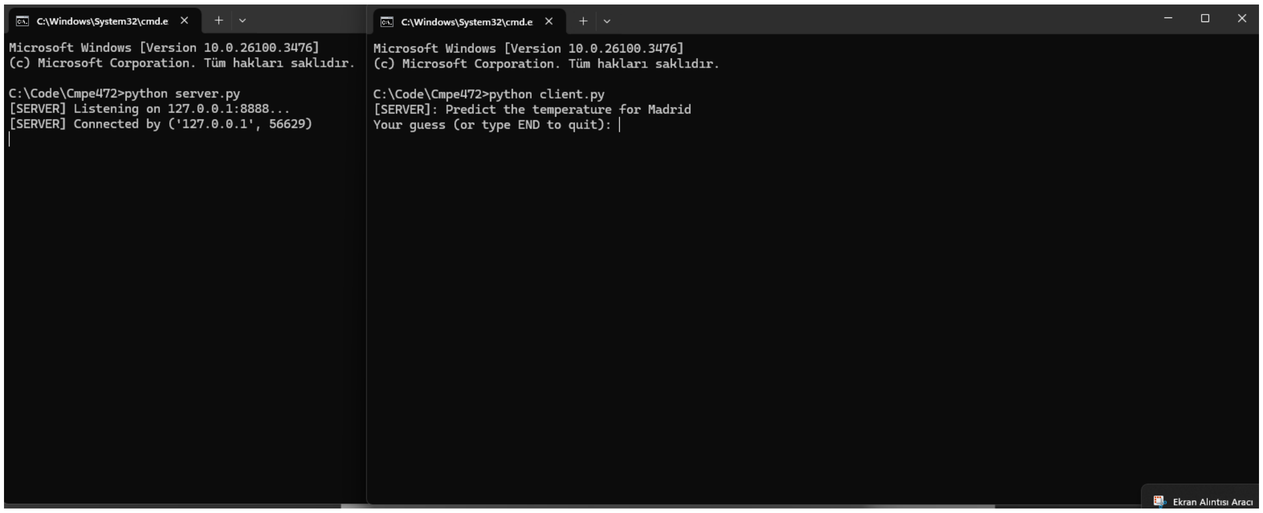
In this programming assignment, a basic client-server based Weather Prediction System was developed using socket programming in Python. The goal of the project is to establish communication between a server and a client running on the same machine (localhost), and simulate a simple weather guessing game.

The server loads weather data from an Excel file containing city names and their corresponding temperatures. It randomly selects a city and asks the client to predict the temperature of that city. The client sends a guess, and the server evaluates the accuracy of the prediction within a specified tolerance range. The system continues until the guess is correct, the maximum number of attempts is reached, or the client explicitly ends the game.

This application demonstrates the use of key networking concepts such as TCP socket communication, request-response cycles, and connection management. Additionally, it incorporates basic file handling and error-checking mechanisms. The implementation is split into two main components:

- server.py: Handles data loading, city selection, and client communication.
- client.py: Connects to the server, interacts with the user, and sends predictions.

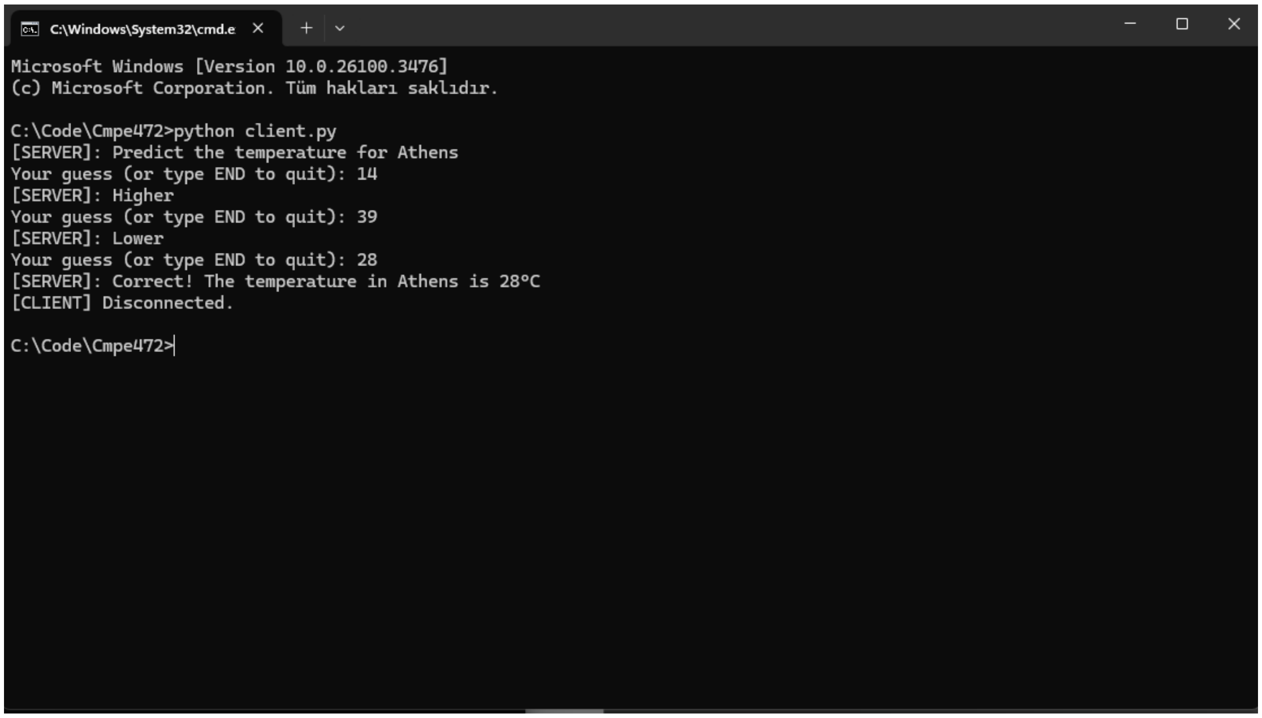
This report presents the design, implementation, and test results of the project, including code explanations and screenshots of sample runs.



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Connected by ('127.0.0.1', 56629)
```

```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Madrid
Your guess (or type END to quit):
```

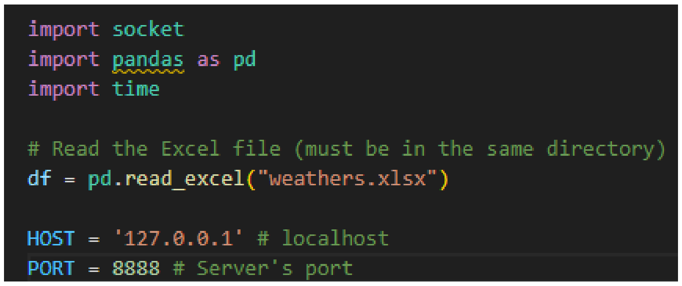
(Figure 1.1 – Connection of client and server)



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Athens
Your guess (or type END to quit): 14
[SERVER]: Higher
Your guess (or type END to quit): 39
[SERVER]: Lower
Your guess (or type END to quit): 28
[SERVER]: Correct! The temperature in Athens is 28°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>
```

(Figure 1.2 – Sample running program via terminal)

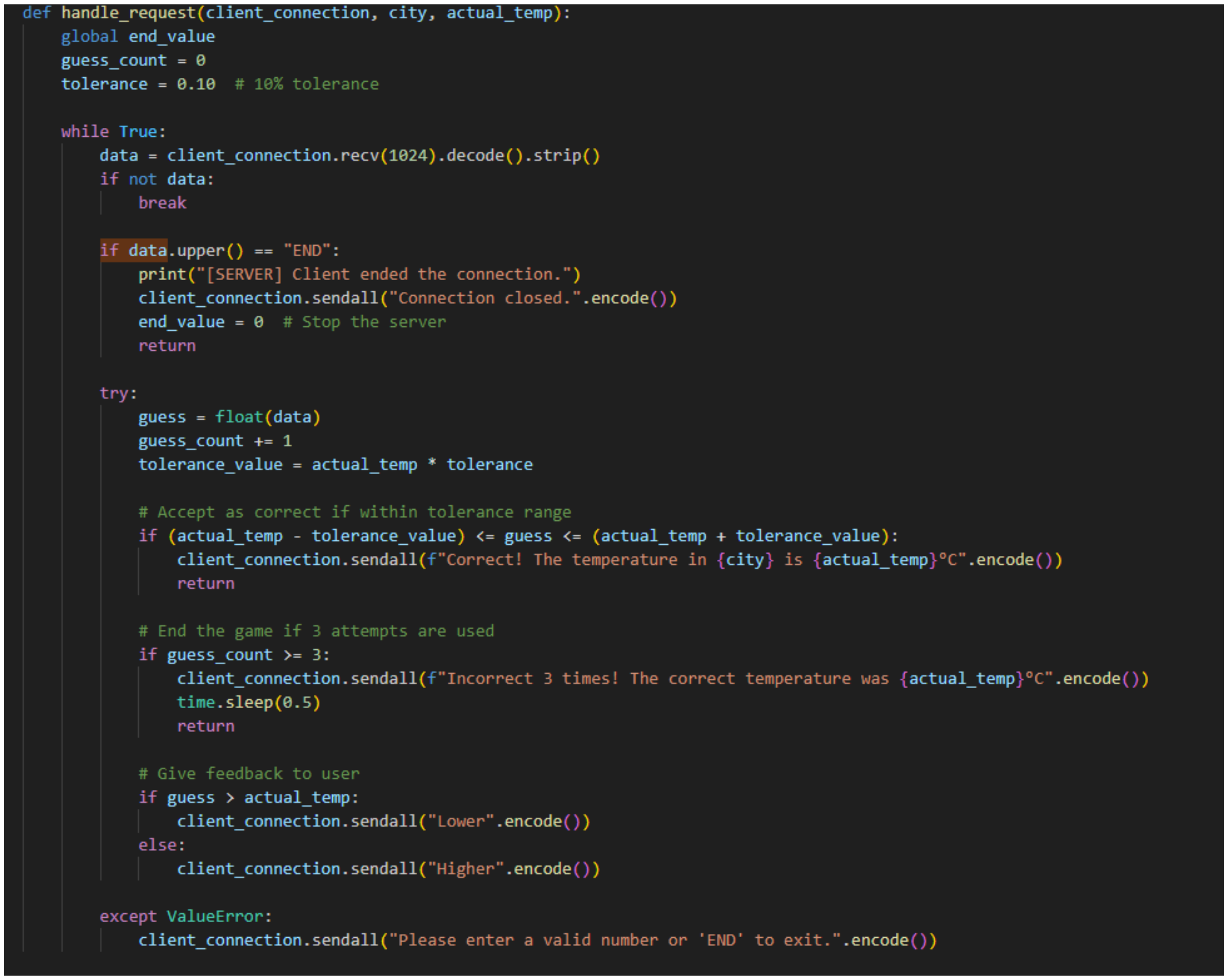
2. Code Explanation



```
import socket
import pandas as pd
import time

# Read the Excel file (must be in the same directory)
df = pd.read_excel("weathers.xlsx")

HOST = '127.0.0.1' # localhost
PORT = 8888 # Server's port
```

(Figure 2.1 – Initial imports and configuration settings in server.py)

- **import socket**: Imports the socket module to establish network communication between the server and the client.
- **import pandas as pd**: Imports the pandas library and aliases it as pd; used to read and handle data from Excel files.
- **import time**: Imports the time module to perform time-based operations like delays.
- **df = pd.read_excel("weathers.xlsx")**: Reads the weathers.xlsx file and loads it into a pandas DataFrame. This file contains city-temperature data and must be located in the same directory as the script.
- **HOST = '127.0.0.1'**: Sets the server's IP address to localhost, meaning it will only accept connections from the same machine.
- **PORT = 8888**: Sets the port number the server will use to listen for incoming client connections.

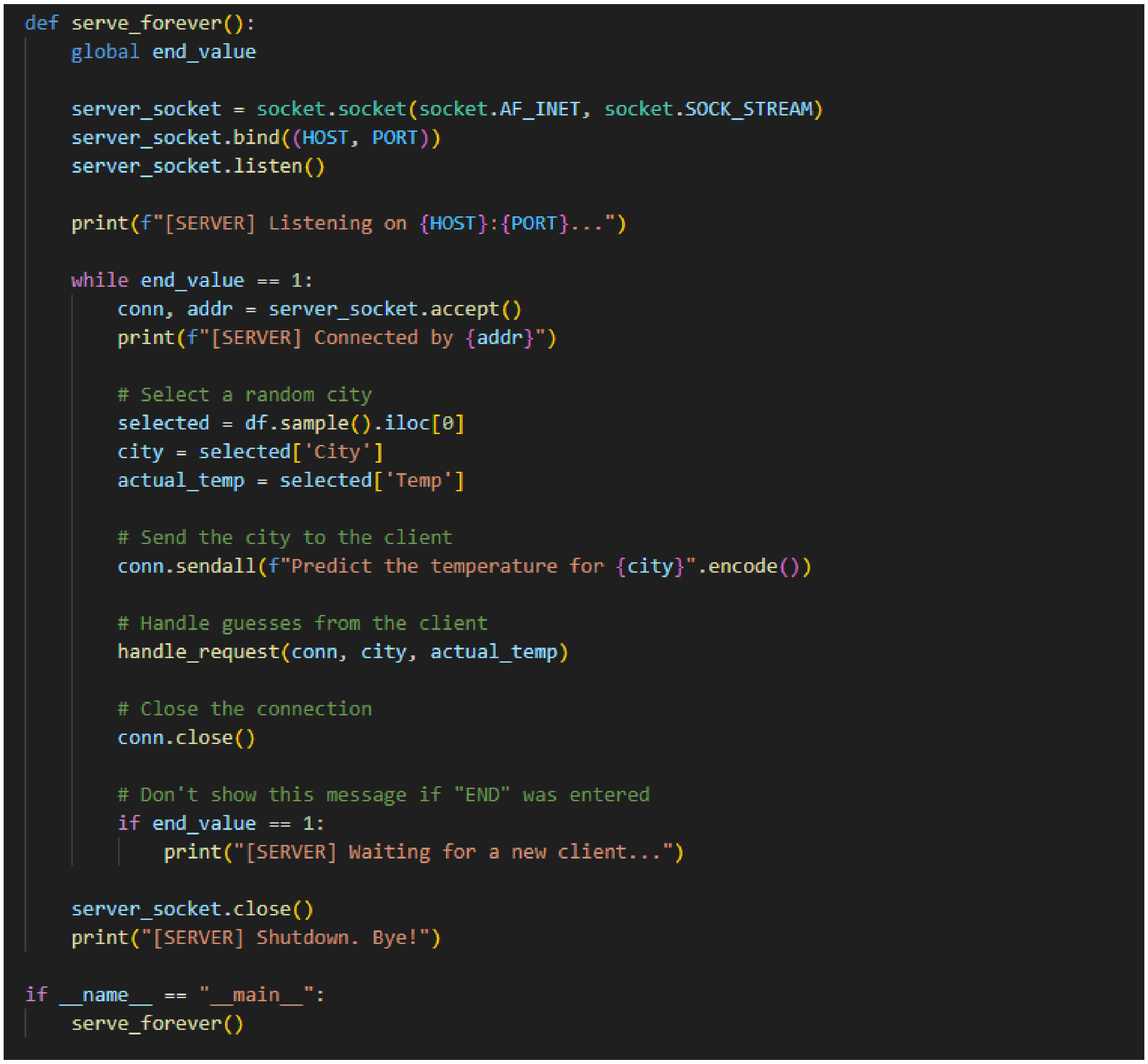


```python
def handle_request(client_connection, city, actual_temp):
    global end_value
    guess_count = 0
    tolerance = 0.10  # 10% tolerance

    while True:
        data = client_connection.recv(1024).decode().strip()
        if not data:
            break

        if data.upper() == "END":
            print("[SERVER] Client ended the connection.")
            client_connection.sendall("Connection closed.".encode())
            end_value = 0  # Stop the server
            return

        try:
            guess = float(data)
            guess_count += 1
            tolerance_value = actual_temp * tolerance

            # Accept as correct if within tolerance range
            if (actual_temp - tolerance_value) <= guess <= (actual_temp + tolerance_value):
                client_connection.sendall(f"Correct! The temperature in {city} is {actual_temp}°C".encode())
                return

            # End the game if 3 attempts are used
            if guess_count >= 3:
                client_connection.sendall(f"Incorrect 3 times! The correct temperature was {actual_temp}°C".encode())
                time.sleep(0.5)
                return

            # Give feedback to user
            if guess > actual_temp:
                client_connection.sendall("Lower".encode())
            else:
                client_connection.sendall("Higher".encode())

        except ValueError:
            client_connection.sendall("Please enter a valid number or 'END' to exit.".encode())
```

(Figure 2.2 – handle_request Function (server.py))

The handle_request function is responsible for managing the interaction between the server and the connected client during the temperature guessing game. It begins by tracking the number of guesses and sets a 10% tolerance range, allowing slightly off guesses to still be considered correct. The function continuously waits for input from the client. If the client enters "END", the server prints a message, sends a confirmation back to the client, updates the end_value to stop the server loop, and exits the function.

When a valid number is received, it is checked against the actual temperature using the defined tolerance. If the guess falls within the acceptable range, the client is informed of their correct answer and the session ends. If the guess is incorrect, the server gives directional feedback such as "Higher" or "Lower". After three incorrect attempts, the correct answer is sent to the client, and the session is terminated.

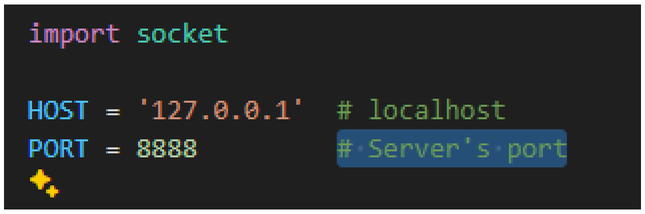

```python
def serve_forever():
    global end_value

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((HOST, PORT))
    server_socket.listen()

    print(f"[SERVER] Listening on {HOST}:{PORT}...")

    while end_value == 1:
        conn, addr = server_socket.accept()
        print(f"[SERVER] Connected by {addr}")

        # Select a random city
        selected = df.sample().iloc[0]
        city = selected['City']
        actual_temp = selected['Temp']

        # Send the city to the client
        conn.sendall(f"Predict the temperature for {city}".encode())

        # Handle guesses from the client
        handle_request(conn, city, actual_temp)

        # Close the connection
        conn.close()

        # Don't show this message if "END" was entered
        if end_value == 1:
            print("[SERVER] Waiting for a new client...")

    server_socket.close()
    print("[SERVER] Shutdown. Bye!")

if __name__ == "__main__":
    serve_forever()
```

(Figure 2.3 – serve_forever Function (server.py))

The serve_forever function is the main server loop that listens for client connections and manages the temperature guessing sessions. It starts by creating a socket object, binding it to a specific IP address and port, and then begins listening for incoming connections. As long as end_value is equal to 1, indicating the server is active, the loop accepts new client connections.

Once a connection is established, the server randomly selects a city and its corresponding temperature from the Excel dataset and sends the city name to the client. Then, it delegates the interaction to the handle_request function, which manages the guessing process. After the interaction ends—either by a correct guess, three failed attempts, or the "END" command—the connection is closed.

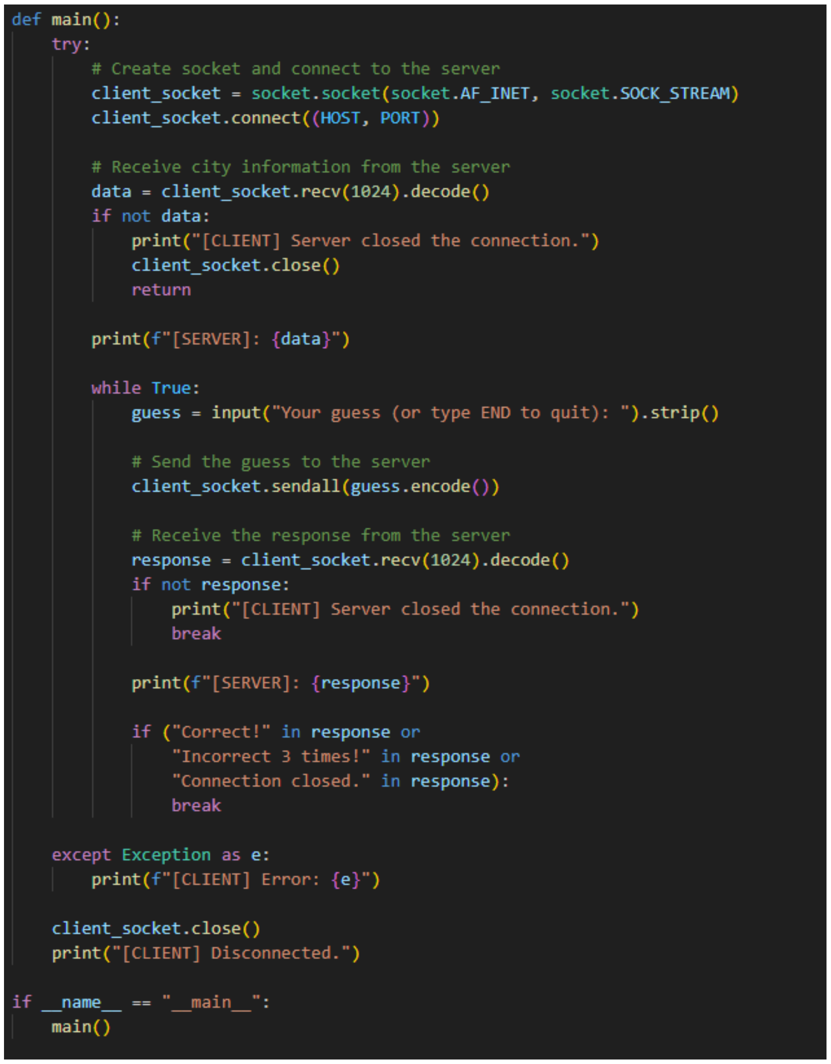
If the game ends normally (not by "END"), the server prints a message indicating that it is ready for a new client. If "END" was received, end_value is set to 0, the loop exits, the server socket is closed, and a shutdown message is printed.



```python
import socket

HOST = '127.0.0.1'  # localhost
PORT = 8888         # Server's port
```

(Figure 2.4 – Initial imports and configuration settings in client.py)

- import socket: Imports the socket module, which allows the program to establish TCP/IP connections between the server and the client.
- HOST = '127.0.0.1': Sets the server's host address to localhost, meaning it will only accept connections from the same machine.
- PORT = 8888: Defines the port number that the server will use to listen for incoming client connections.



```python
def main():
    try:
        # Create socket and connect to the server
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((HOST, PORT))

        # Receive city information from the server
        data = client_socket.recv(1024).decode()
        if not data:
            print("[CLIENT] Server closed the connection.")
            client_socket.close()
            return

        print(f"[SERVER]: {data}")

        while True:
            guess = input("Your guess (or type END to quit): ").strip()

            # Send the guess to the server
            client_socket.sendall(guess.encode())

            # Receive the response from the server
            response = client_socket.recv(1024).decode()
            if not response:
                print("[CLIENT] Server closed the connection.")
                break

            print(f"[SERVER]: {response}")

            if ("Correct!" in response or
                "Incorrect 3 times!" in response or
                "Connection closed." in response):
                break

    except Exception as e:
        print(f"[CLIENT] Error: {e}")

    client_socket.close()
    print("[CLIENT] Disconnected.")

if __name__ == "__main__":
    main()
```

(Figure 2.5 – main Function (client.py))

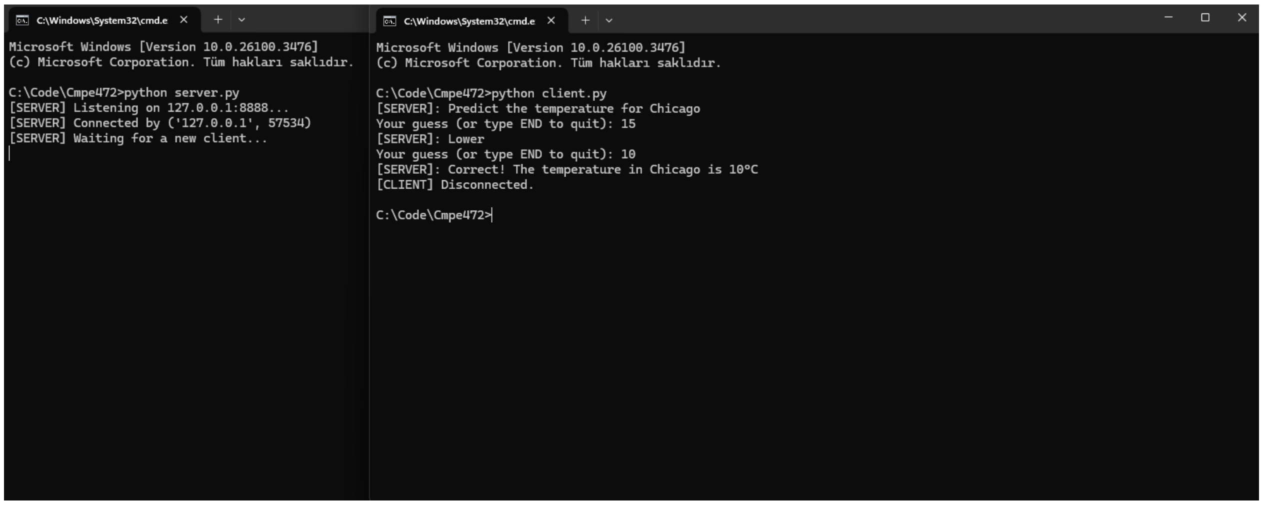
The main() function in the client program is responsible for establishing a connection with the server and handling the user interaction during the temperature guessing game. It first creates a socket and connects to the server using the predefined HOST and PORT values.

Once connected, the client waits to receive an initial message from the server, which contains the name of the city for which the user is expected to guess the temperature. If the server sends no data, the client prints a message and closes the connection.

Inside a while True loop, the user is prompted to enter a guess or type "END" to exit the game. This guess is sent to the server. After sending, the client waits for a response. If the server closes the connection or sends no response, the loop is terminated.

The server's response is displayed, and if the response includes "Correct!", "Incorrect 3 times!", or "Connection closed.", the game ends and the loop breaks. The function includes exception handling to catch and display any runtime errors. Finally, regardless of how the game ends, the socket is closed and a disconnection message is printed.

3. Output and Results

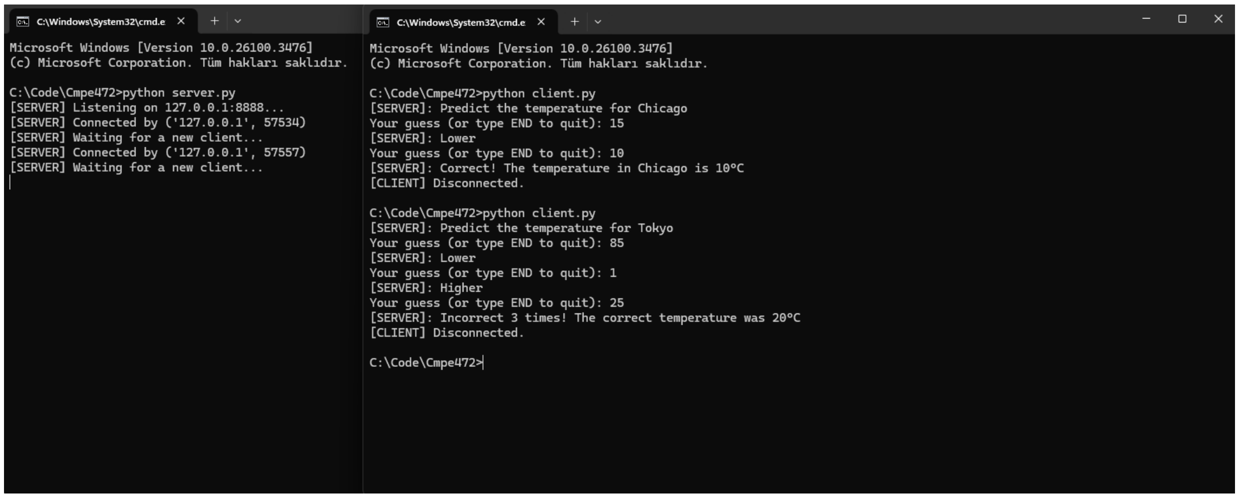


```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Connected by ('127.0.0.1', 57534)
[SERVER] Waiting for a new client...
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Chicago
Your guess (or type END to quit): 15
[SERVER]: Lower
Your guess (or type END to quit): 10
[SERVER]: Correct! The temperature in Chicago is 10°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>
```

(Figure 3.1 – Successful temperature guess and client-server interaction)

The client connects to the server, receives the city name (Chicago), submits guesses, and receives feedback until the correct temperature is guessed and the session ends.

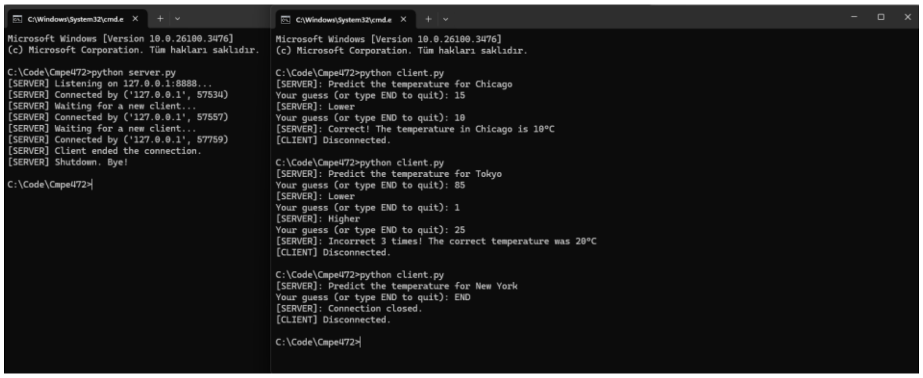


```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Connected by ('127.0.0.1', 57534)
[SERVER] Waiting for a new client...
[SERVER] Connected by ('127.0.0.1', 57557)
[SERVER] Waiting for a new client...
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Chicago
Your guess (or type END to quit): 15
[SERVER]: Lower
Your guess (or type END to quit): 10
[SERVER]: Correct! The temperature in Chicago is 10°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Tokyo
Your guess (or type END to quit): 85
[SERVER]: Lower
Your guess (or type END to quit): 1
[SERVER]: Higher
Your guess (or type END to quit): 25
[SERVER]: Incorrect 3 times! The correct temperature was 20°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>
```

(Figure 3.2 – Unsuccessful temperature guess and client-server interaction)

The client attempts to guess the temperature but fails within three tries, prompting the server to reveal the correct answer and terminate the session.

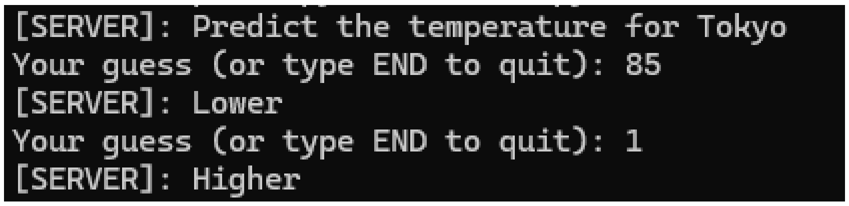


```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Connected by ('127.0.0.1', 57534)
[SERVER] Waiting for a new client...
[SERVER] Connected by ('127.0.0.1', 57557)
[SERVER] Waiting for a new client...
[SERVER] Connected by ('127.0.0.1', 57759)
[SERVER] Client ended the connection.
[SERVER] Shutdown. Bye!

C:\Code\Cmpe472>
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Chicago
Your guess (or type END to quit): 15
[SERVER]: Lower
Your guess (or type END to quit): 10
[SERVER]: Correct! The temperature in Chicago is 10°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Tokyo
Your guess (or type END to quit): 85
[SERVER]: Lower
Your guess (or type END to quit): 1
[SERVER]: Higher
Your guess (or type END to quit): 25
[SERVER]: Incorrect 3 times! The correct temperature was 20°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for New York
Your guess (or type END to quit): END
[SERVER]: Connection closed.
[CLIENT] Disconnected.

C:\Code\Cmpe472>
```

(Figure 3.3 – Client termination using the END command)

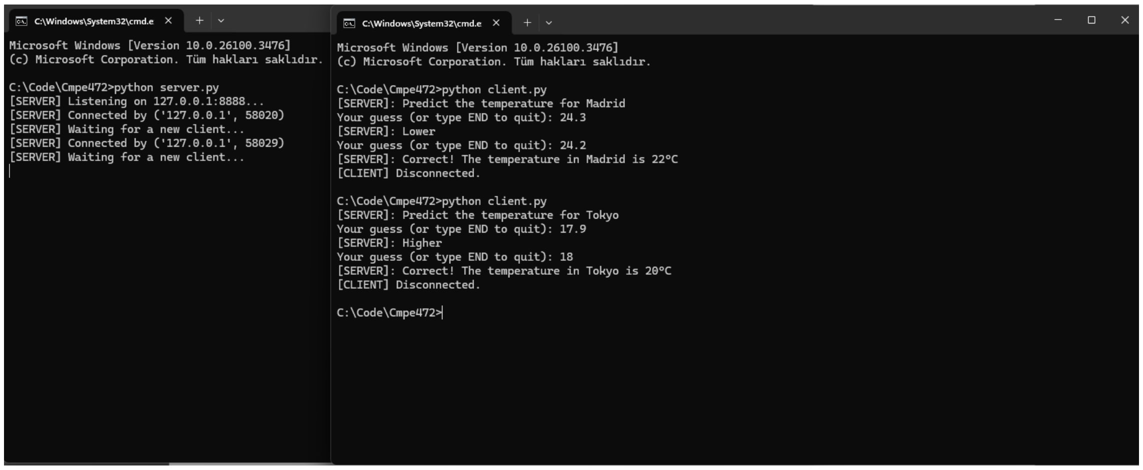
The client enters "END" to exit the game, which triggers a clean disconnection and causes the server to shut down gracefully.



```
[SERVER]: Predict the temperature for Tokyo
Your guess (or type END to quit): 85
[SERVER]: Lower
Your guess (or type END to quit): 1
[SERVER]: Higher
```

(Figure 3.4 – Server feedback on incorrect guesses)

The server provides directional feedback ("Lower" or "Higher") after each incorrect guess to help guide the client toward the correct temperature.



```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python server.py
[SERVER] Listening on 127.0.0.1:8888...
[SERVER] Connected by ('127.0.0.1', 58020)
[SERVER] Waiting for a new client...
[SERVER] Connected by ('127.0.0.1', 58029)
[SERVER] Waiting for a new client...
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Madrid
Your guess (or type END to quit): 24.3
[SERVER]: Lower
Your guess (or type END to quit): 24.2
[SERVER]: Correct! The temperature in Madrid is 22°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>python client.py
[SERVER]: Predict the temperature for Tokyo
Your guess (or type END to quit): 17.9
[SERVER]: Higher
Your guess (or type END to quit): 18
[SERVER]: Correct! The temperature in Tokyo is 20°C
[CLIENT] Disconnected.

C:\Code\Cmpe472>
```

(Figure 3.5 – Correct temperature guesses within tolerance range)

The client successfully guesses the temperatures for Madrid and Tokyo within the allowed 10% tolerance, demonstrating accurate input handling and successful session completion.