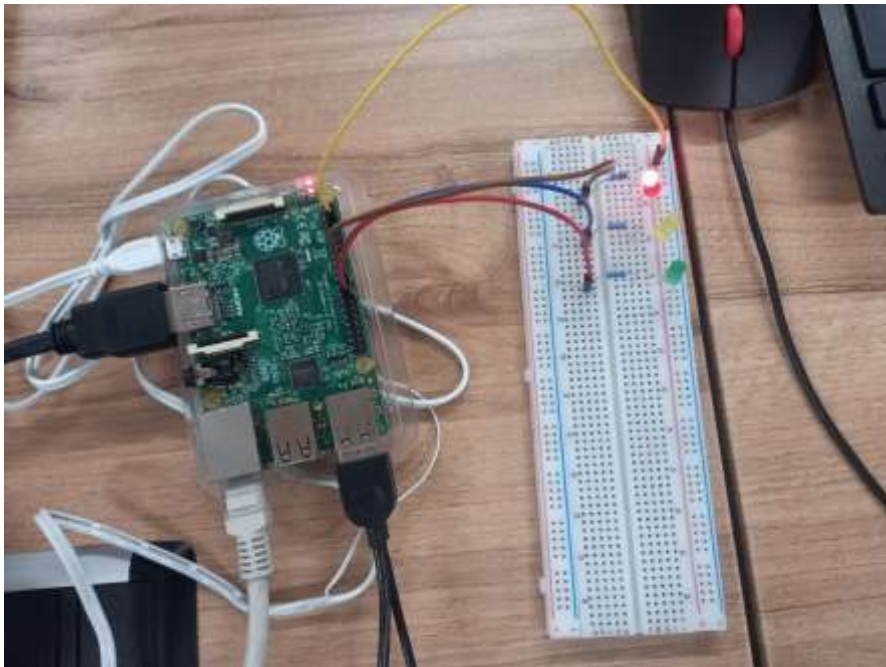


# CSC311 Networking Practical 1

## Assignment Report

Group 10

## Traffic Control System



# Table of Contents

Introduction	PG 3
Hardware and Design	PG 3
System Architecture Diagram	PG 7
Technical Implementation	PG 11
Server script	PG 20
Client script	PG 21
Conclusion	PG 22
References	PG 22
Student Contributions	PG 23

## Introduction

The Traffic Control System project presents a sophisticated solution utilizing Raspberry Pi technology to replicate and manage traffic flow at an intersection. Our endeavor is to enhance safety and efficiency by emulating traditional traffic light functionality while integrating contemporary IoT principles.

Possible use cases:

**Integration with Smart Cities:** This technology is applicable to smart city projects that optimize traffic flow, reduce congestion, and raise the bar for urban mobility.

**Emergency Response Management:** The system's flexibility enables the prioritizing of emergency vehicles in times of crisis, such as natural disasters or accidents, guaranteeing prompt and secure transit.

**Event Traffic Management:** Our technology reduces disruptions and improves attendee experience by streamlining traffic around venues for everything from large-scale gatherings to sporting events.

**Programs for Pedestrian Safety:** With its pedestrian crossing capabilities, the system supports efforts to ensure urban safety, especially in places with high population density.

## Hardware and Design

This project simulates a traffic light system at an intersection using a Raspberry Pi.

Objectives:

- Develop a Python program to control traffic light sequencing.
- Simulate real-world traffic light behaviour for vehicles and pedestrians.

Hardware Components:

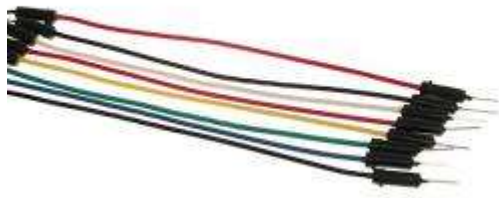
1. Raspberry Pi



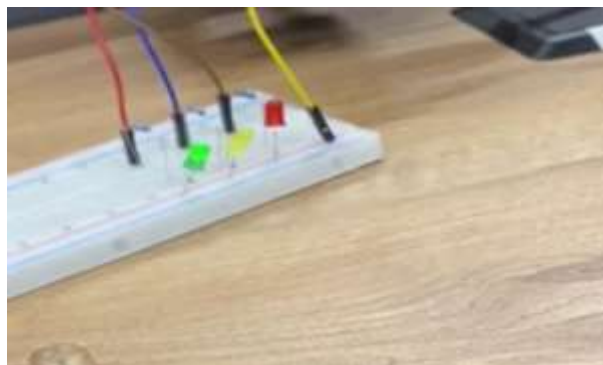
## 2. Breadboard



## 3. Jumper Wires



## 4. 3x LEDs (Red, Yellow, Green)



1. **Raspberry Pi:** The Raspberry Pi acts as the brain of the system, running the Python script that controls the traffic light sequence.
2. **Breadboard:** The breadboard provides a platform for connecting the Raspberry Pi to the LEDs and resistors.
3. **Jumper Wires:** These wires connect the Raspberry Pi's GPIO pins to the LEDs and resistors, allowing the Raspberry Pi to control the LEDs.
4. **LEDs (3x):** Three LEDs represent the traffic light signals: red for stop, yellow for caution, and green for go.

- **Python Script:** The Python script running on the Raspberry Pi controls the timing and sequence of the traffic lights. It can be programmed to simulate different traffic light patterns, such as fixed timings or adaptive timings based on simulated traffic density.
1. The LEDs are connected to the Raspberry Pi's GPIO pins using jumper wires.
  2. The resistors are placed in series with each LED to control the current.
  3. The Python script is written to control the GPIO pins, turning the LEDs on and off according to the desired traffic light sequence.
- The Python script can be designed to accept user input to modify traffic light timings.
  - Error handling can be implemented to detect and respond to potential issues, such as hardware malfunctions.

### **System Modules:**

**Traffic Light Manager:** This Python module controls the overall traffic light sequence. It defines the different light states (red, yellow, green) and their durations. It also manages the transitions between states based on timing or external triggers (if implemented).

**GPIO Interface:** This module handles communication between the Raspberry Pi and the physical LEDs. It translates the desired light states (red, yellow, green) from the Traffic Light Manager into specific GPIO pin operations (on/off) for the LEDs.

### **System Interactions:**

The Traffic Light Manager defines the traffic light sequence, including durations for each light state (red, yellow, green) and the order of transitions.

The Traffic Light Manager calls the GPIO Interface module whenever a state change is required.

The GPIO Interface module receives the desired light state (red, yellow, green) from the Traffic Light Manager.

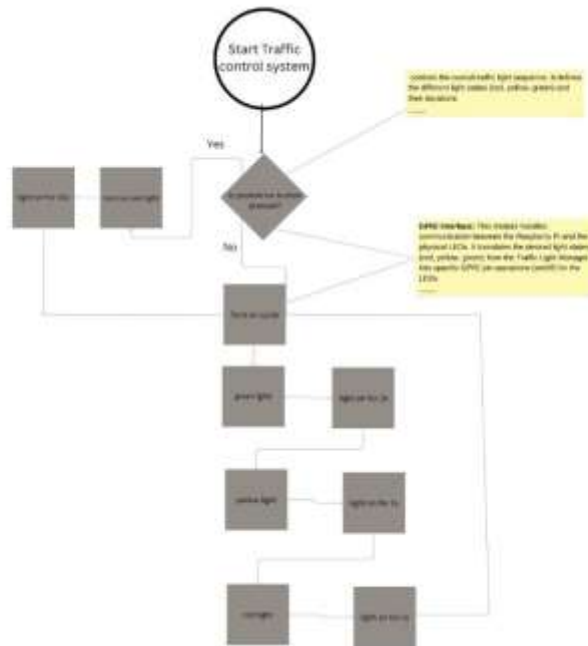
Based on the received state, the GPIO Interface module activates/deactivates the corresponding GPIO pins connected to the LEDs.

(Optional) The External Input Module (if implemented) reads data from external sensors and communicates with the Traffic Light Manager to potentially adjust light timings dynamically.

### **Flowchart - Traffic Light Cycle:**

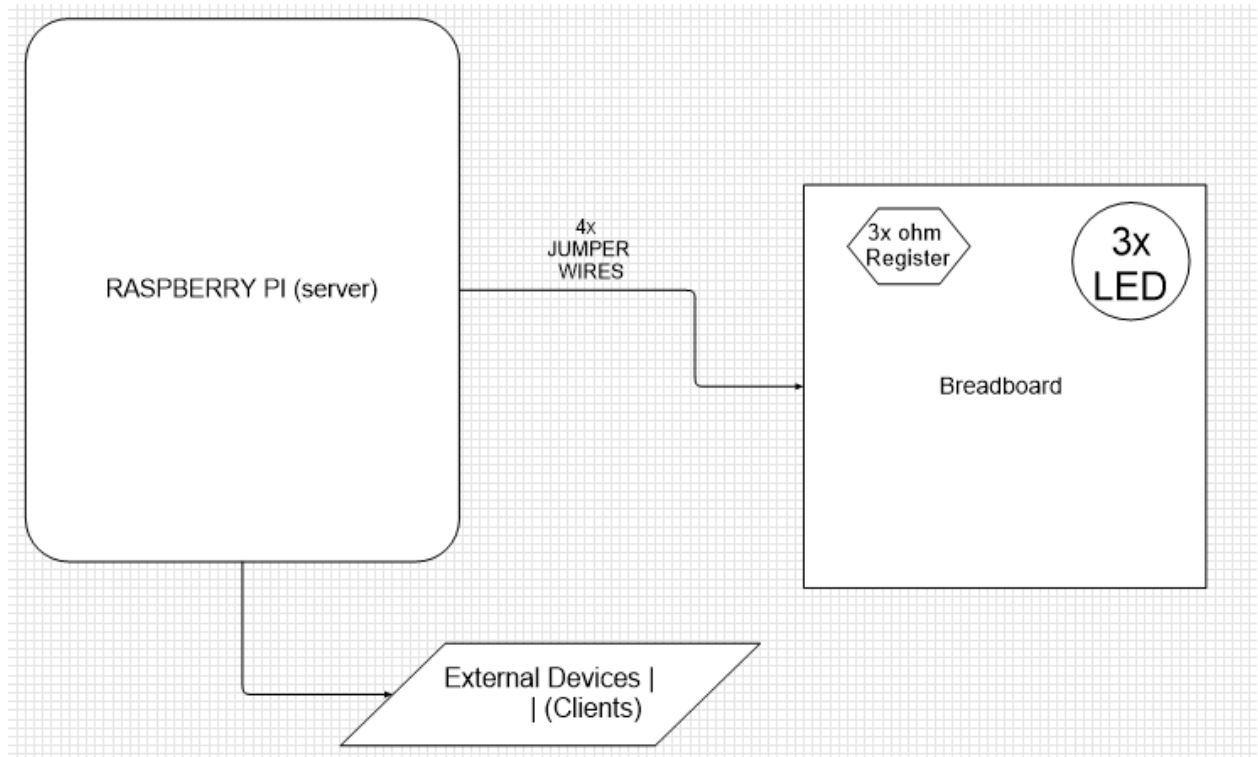
1. **Start**
2. **Traffic Light Manager:**
  - Define light state durations (Red, Yellow, Green)
  - Define state transitions
3. **Traffic Light Manager calls GPIO Interface:**

- Provide desired light state (Red, Yellow, Green)
  - 4. **GPIO Interface:**
    - Activate/Deactivate corresponding GPIO pin based on received state
    - Control LED (on/off)
  - 5. **Delay for current light state duration**
- Loop back to Step 2** (for continuous traffic light cycle)



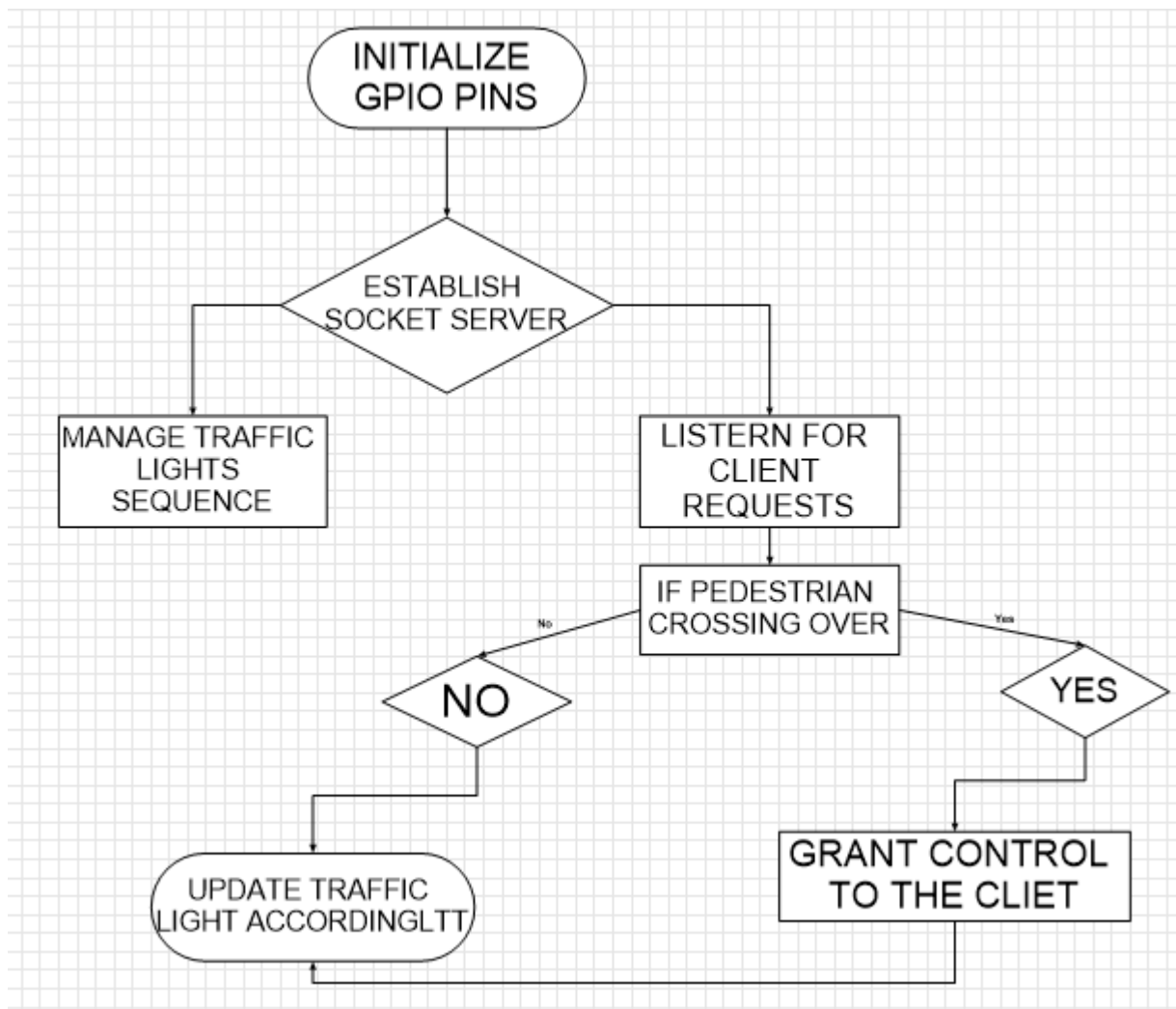
**System Architecture Diagram:**

The Traffic Control System architecture consists of two main components: the Raspberry Pi acting as the server and external devices (clients). Below is an overview of the system architecture:



### Flowchart - Server Script:

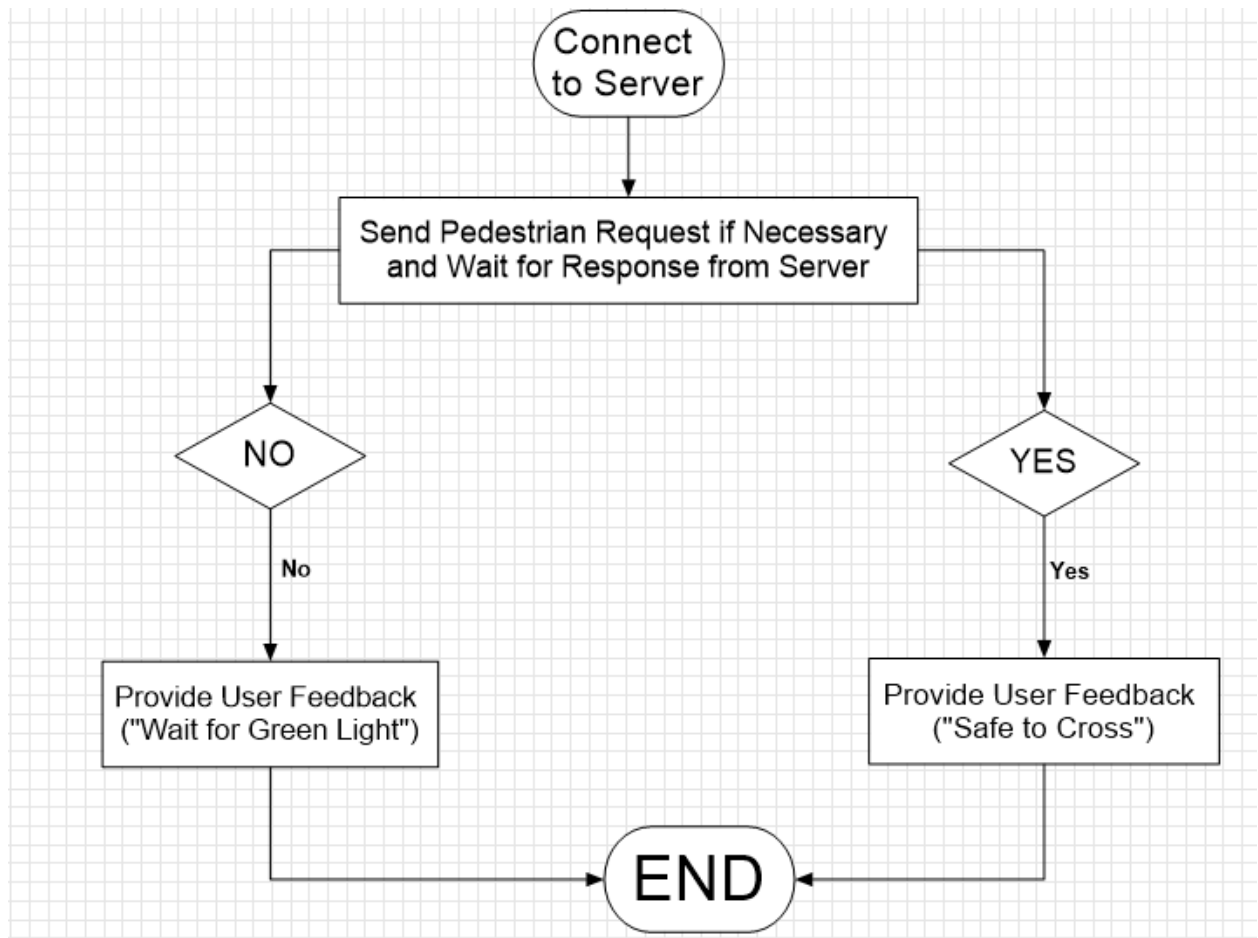
The server script manages traffic lights and handles pedestrian crossing requests. Below is a simplified flowchart illustrating its operation:





### Flowchart - Client Script:

The client script facilitates communication with the server and sends pedestrian crossing requests. Below is a simplified flowchart illustrating its operation:



### Design Decisions:

1. Programming Language: Python was chosen for its simplicity, readability, and availability of libraries for GPIO control and socket communication.
2. Communication Protocol: TCP/IP was selected for its reliability and support for bidirectional communication between the server and clients.
3. GPIO Pin Mappings: Specific GPIO pins (9, 15 and 21) were mapped to control the LEDs representing traffic lights based on the Raspberry Pi model and pin available, ensuring compatibility and functionality with the LED components.

### Design Patterns/Principles:

1. Client-Server Architecture-Implemented to separate concerns and facilitate communication between the Raspberry Pi server and external devices.
2. Observer Pattern: Used to handle pedestrian crossing requests, where the server notifies the traffic lights to update based on client requests.

3. Single Responsibility Principle (SRP): Each script (server and client) is designed to perform specific tasks, promoting code maintainability and reusability.
4. Separation of Concerns: Ensured that server and client scripts handle distinct responsibilities, promoting modularity and maintainability.
5. Error Handling: Implemented robust error handling mechanisms to gracefully manage unexpected events during communication and GPIO control.

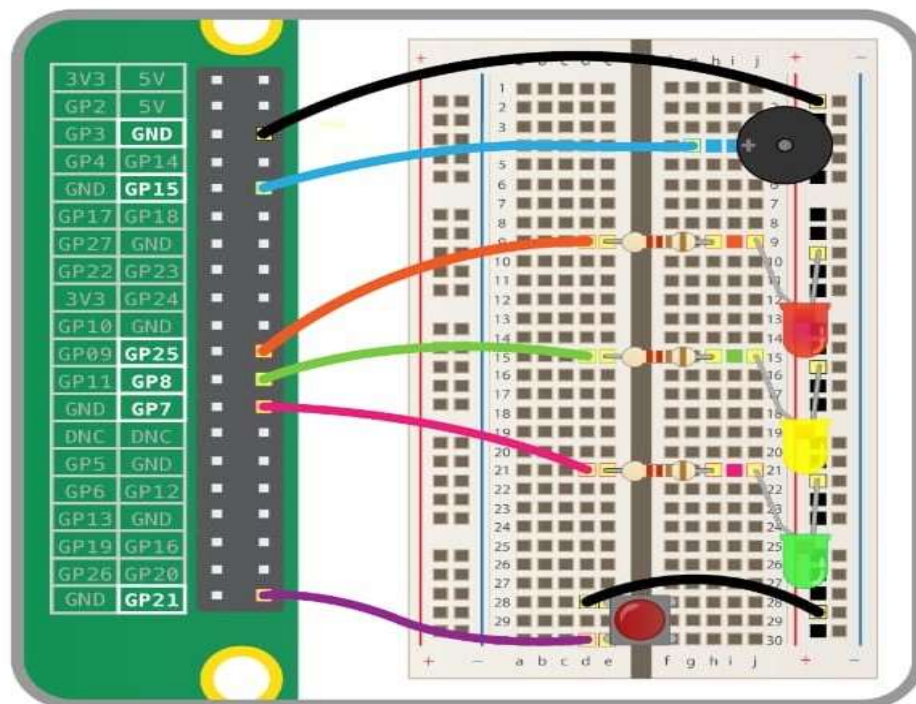
## Technical Implementation

The provided image illustrates the physical setup of the Traffic Control System, focusing on the connection between the GPIO pins on the Raspberry Pi and the components on the breadboard. Below is a breakdown of the key elements depicted in the picture:

The Raspberry Pi board prominently features a set of GPIO (General Purpose Input/Output) pins, which serve as the interface between the Raspberry Pi and external hardware components. These pins are strategically located along the edge of the board, facilitating easy access for connecting various peripherals.

The GPIO pins from the Raspberry Pi are connected to the breadboard, a versatile prototyping tool used to create electronic circuits. In the image, the GPIO pins are carefully wired to specific rows and columns on the breadboard, enabling seamless integration with the hardware components of the Traffic Control System.

Surrounding the GPIO pins and breadboard, various hardware components are positioned, including LEDs, resistors, and jumper wires. Each component plays a crucial role in the functionality of the Traffic Control System, such as indicating traffic light status, initiating pedestrian crossings, and regulating electrical currents.



This is the design of the Raspberry Pi setup, illustrating the connection of jumper wires from the breadboard to specific GPIO pins.

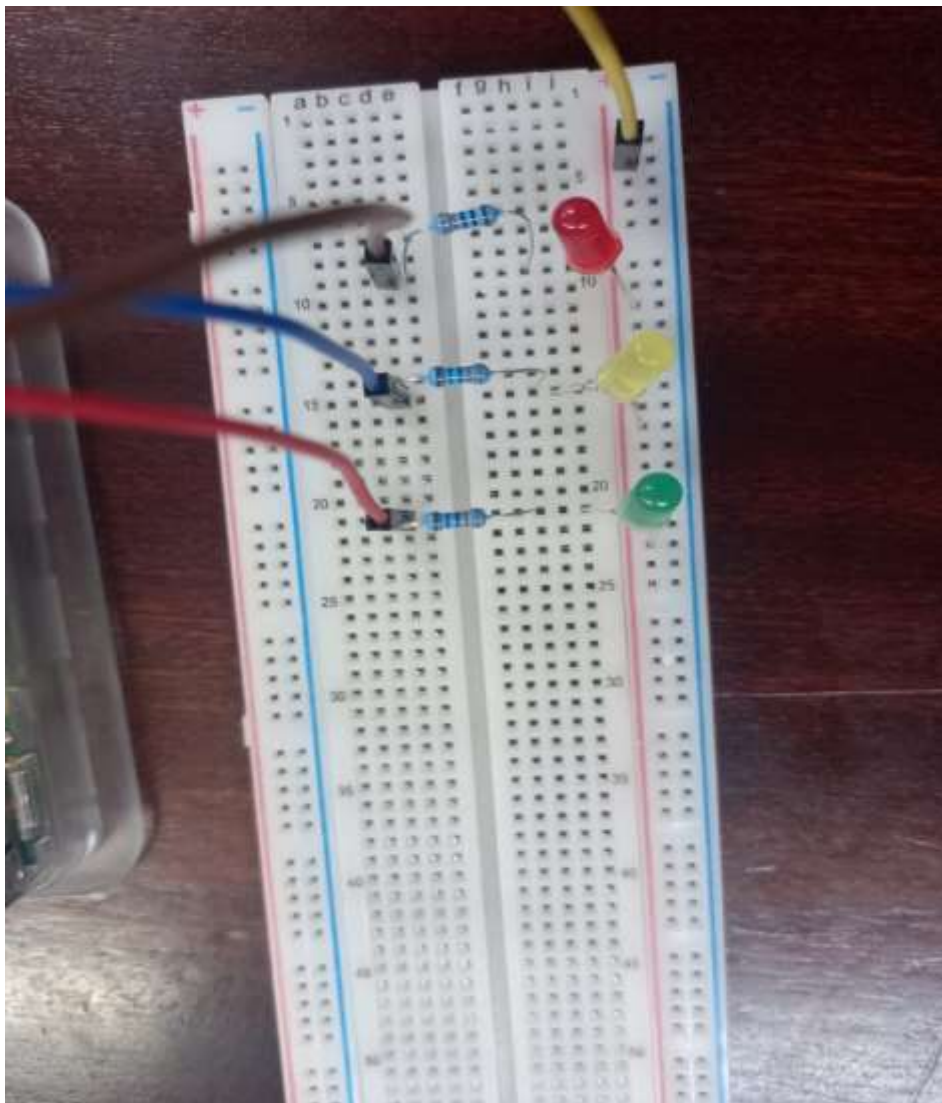
The image below showcases a breadboard setup with a series of jumper wires interconnecting various components, facilitating the electrical connections necessary for the

Traffic Control System project. The breadboard serves as a central platform for organizing and connecting the hardware components, providing a convenient and modular layout for prototyping and experimentation.

The jumper wires, depicted in various colours, are meticulously arranged to establish connections between the Raspberry Pi and other peripheral components, such as LEDs and resistors. Each jumper wire is carefully inserted into the designated holes on the breadboard, ensuring secure and reliable electrical connections.

The LEDs, represented by small cylindrical components with coloured caps, are strategically positioned on the breadboard, with each LED corresponding to a specific function within the Traffic Control System. These LEDs serve as visual indicators for the traffic lights, emitting distinct colours such as red, yellow, and green to signal vehicular and pedestrian traffic.

Accompanying the LEDs are resistors, denoted by small cylindrical components with coloured bands, which are essential for regulating the flow of electrical current through the LEDs. The resistors are integrated into the circuitry to prevent overloading and ensure the longevity of the LEDs, maintaining consistent and reliable performance over time.



Jumper wires connected with resistors and LEDs: Ensuring precise control and illumination in the Traffic Control System.



Connecting the wires to the specified GPIO pins

3V3	5V
GPIO2	5V
GPIO3	GND
GPIO4	GPIO14
GND	GPIO15
GPIO17	GPIO18
GPIO27	GND
GPIO22	GPIO23
3V3	GPIO24
GPIO10	GND
GPIO9	GPIO25
GPIO11	GPIO8
GND	GPIO7
ADV	ADV
GPIO5	GND
GPIO6	GPIO12
GPIO13	GND
GPIO19	GPIO16
GPIO26	GPIO20
GND	GPIO21

GPIO pins diagram. 3 jumper cables attached to GPIO7, GPIO8 and GPIO25

To configure the GPIO pins on the Raspberry Pi for interfacing with the hardware components, we first identified the purpose of each GPIO pin used in the project. Here's a breakdown of the GPIO pin configuration and initialization process:

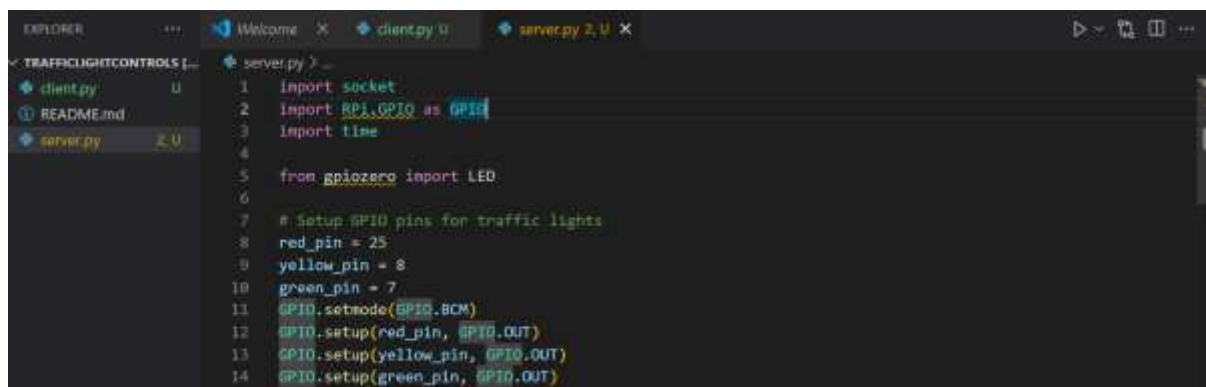
### Purpose of GPIO Pins:

**GPIO Pin 25: Controls the red LED for vehicular traffic stop signal.**

**GPIO Pin 8: Controls the yellow LED for vehicular traffic caution signal.**

**GPIO Pin 7: Controls the green LED for vehicular traffic go signal.**

**Initialization in Python Code:** We used the RPi.GPIO library in Python to initialize and control the GPIO pins. Here's how each GPIO pin is initialized in the Python code:

A screenshot of a code editor window with a dark theme. The left sidebar shows a file explorer with a project named 'TRAFFICLIGHTCONTROLS' containing files 'client.py', 'README.md', and 'server.py'. The 'server.py' file is selected and open in the main editor. The code in 'server.py' is as follows:

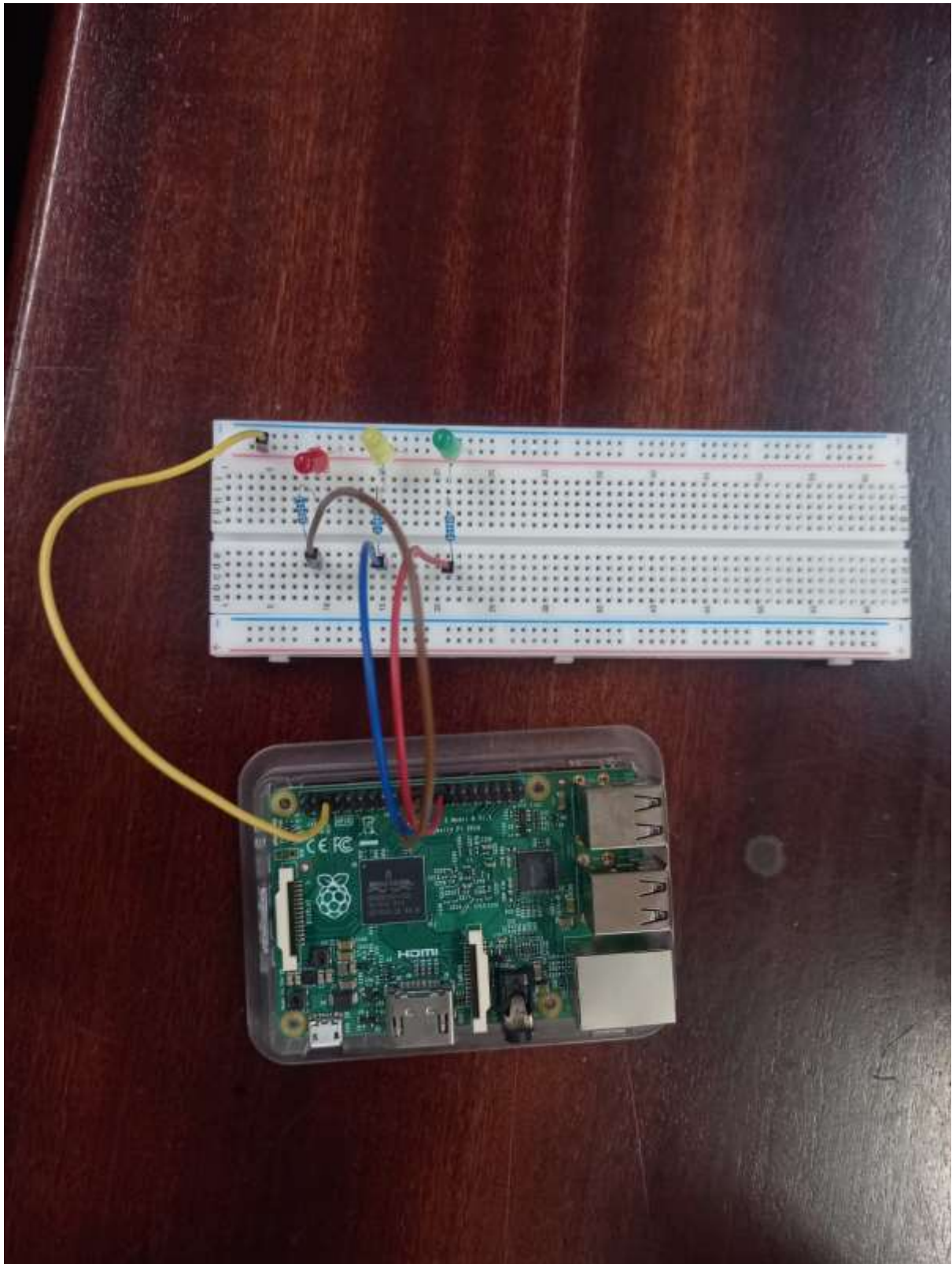
```
1 import socket
2 import RPi.GPIO as GPIO
3 import time
4
5 from gpiozero import LED
6
7 # Setup GPIO pins for traffic lights
8 red_pin = 25
9 yellow_pin = 8
10 green_pin = 7
11 GPIO.setmode(GPIO.BCM)
12 GPIO.setup(red_pin, GPIO.OUT)
13 GPIO.setup(yellow_pin, GPIO.OUT)
14 GPIO.setup(green_pin, GPIO.OUT)
```

The GPIO (General Purpose Input/Output) pins on the Raspberry Pi are meticulously connected with resistors and LEDs to ensure precise control and illumination. This careful arrangement of electronic components plays a pivotal role in orchestrating the behavior of the traffic lights, enabling them to function with accuracy and reliability.

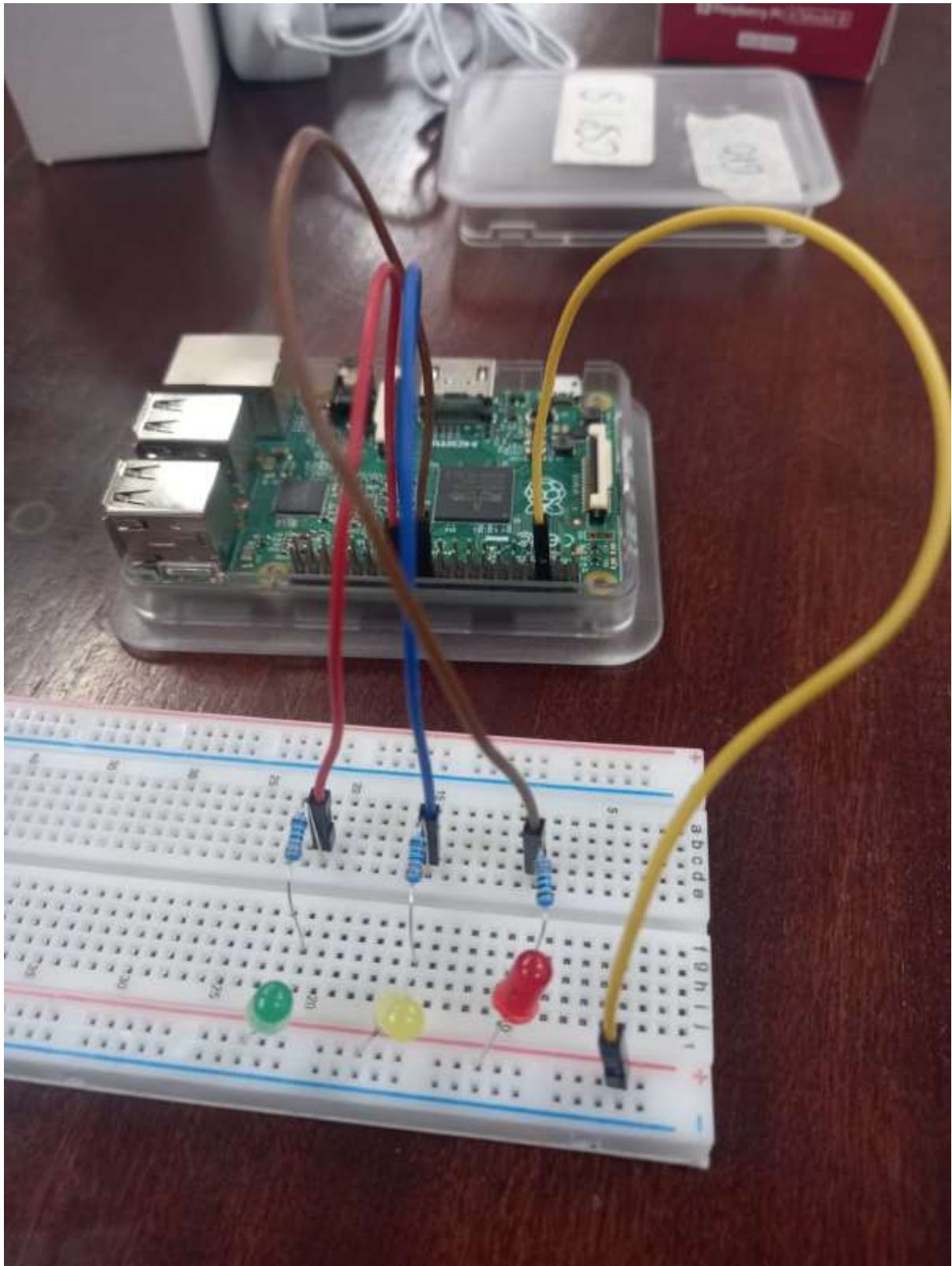
The inclusion of resistors in the circuitry serves to regulate the flow of electrical current, preventing excessive power consumption and potential damage to the LEDs. By strategically integrating resistors into the design, the system achieves optimal performance while safeguarding the longevity of its components.

The LEDs, coupled with the GPIO pins and resistors, facilitate illumination of the traffic lights, delivering signals to motorists and pedestrians. Through the utilization of these components, the Traffic Control System not only ensures effective traffic management but also enhances safety and visibility at intersections, contributing to the overall efficiency and functionality of urban transportation systems.



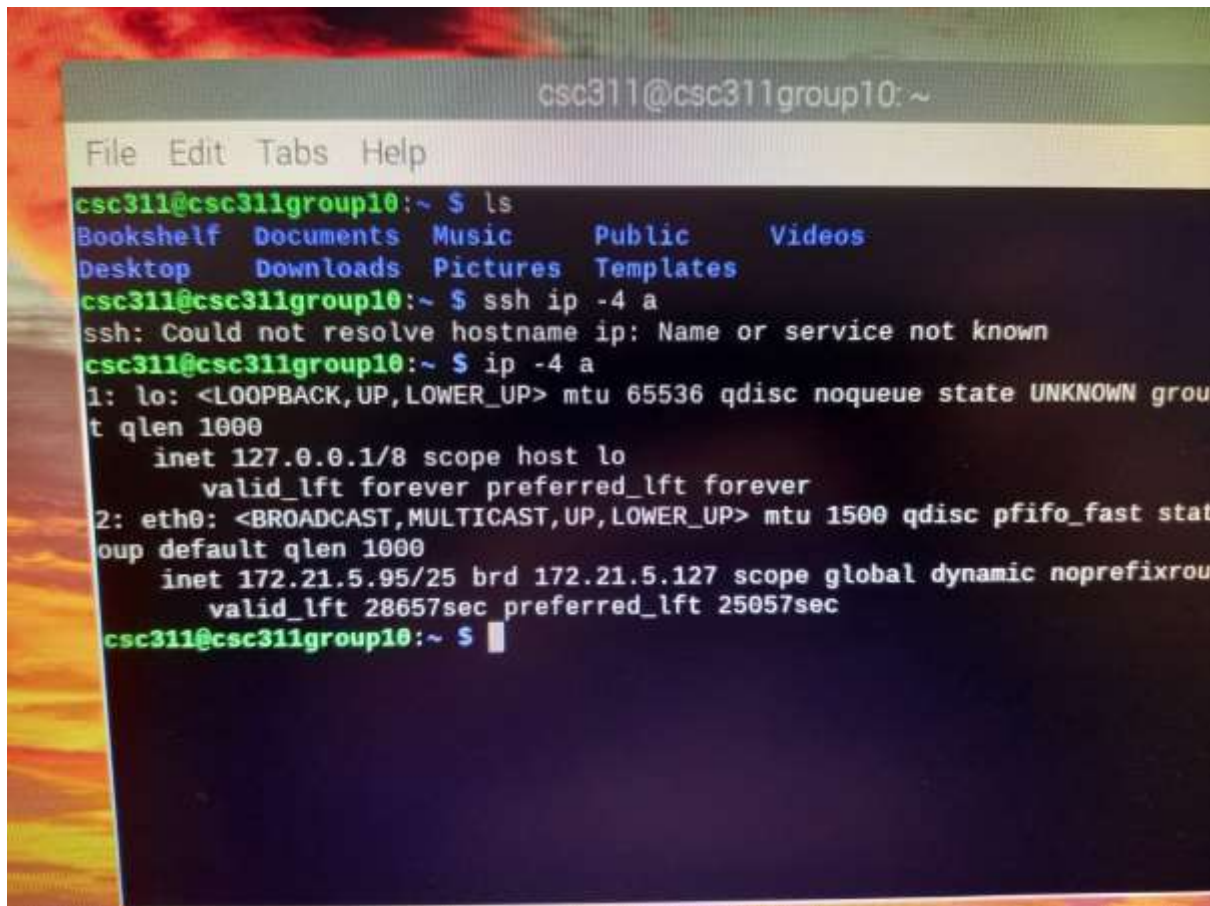


Final result of the traffic control system



Close-ups of the traffic control system



A terminal window titled 'csc311@csc311group10:~' with a menu bar 'File Edit Tabs Help'. The terminal shows the following commands and output:

```
csc311@csc311group10:~ $ ls
Bookshelf  Documents  Music      Public     Videos
Desktop    Downloads  Pictures    Templates

csc311@csc311group10:~ $ ssh ip -4 a
ssh: Could not resolve hostname ip: Name or service not known

csc311@csc311group10:~ $ ip -4 a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
    t qlen 1000
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
    oup default qlen 1000
    inet 172.21.5.95/25 brd 172.21.5.127 scope global dynamic noprefixrou
        valid_lft 28657sec preferred_lft 25057sec

csc311@csc311group10:~ $
```

Searching for IP Address

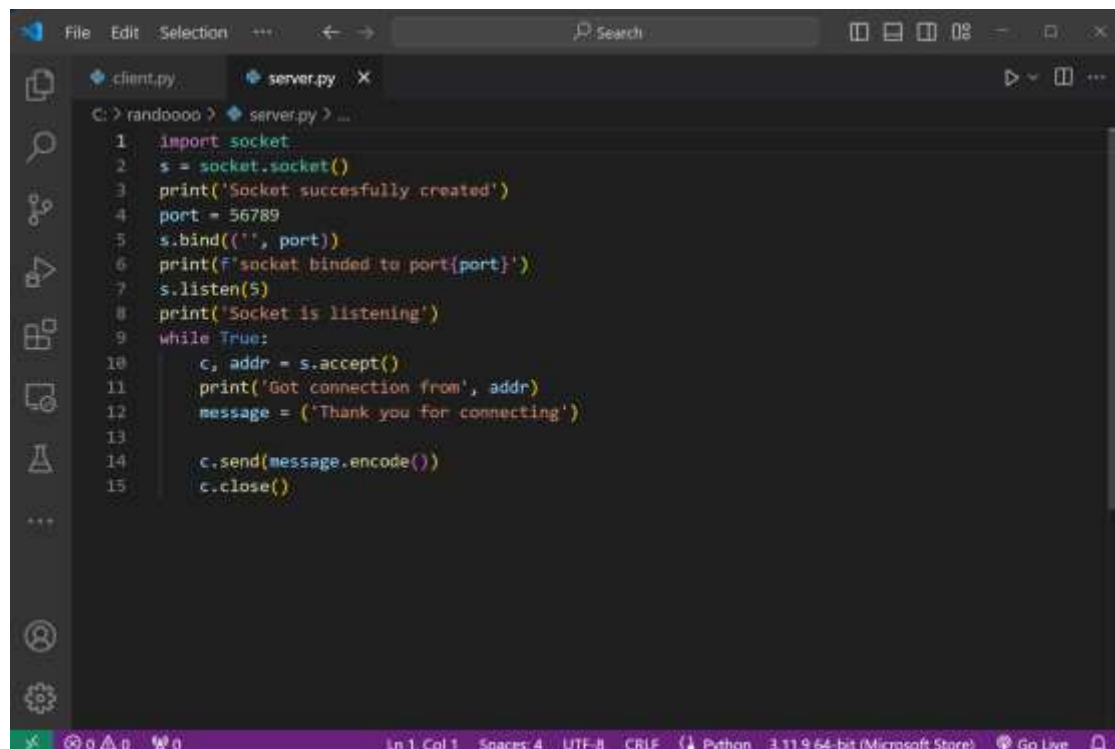
**Below is the first code written to establish a client-server relationship:**

**Socket Setup:** Both the server and client use sockets for communication. Sockets provide a communication channel between processes running on different devices. In our system, the server creates a socket and binds it to a specific port, while the client creates a socket to connect to the server's IP address and port.

**Server Waiting for Connections:** After creating and binding the socket, the server enters a listening state, waiting for incoming connections from clients. The `listen()` method allows the server to accept a specified number of incoming connections.

**Client Connection:** The client initiates a connection to the server by creating a socket and specifying the server's IP address and port. The `connect()` method establishes a connection to the server.

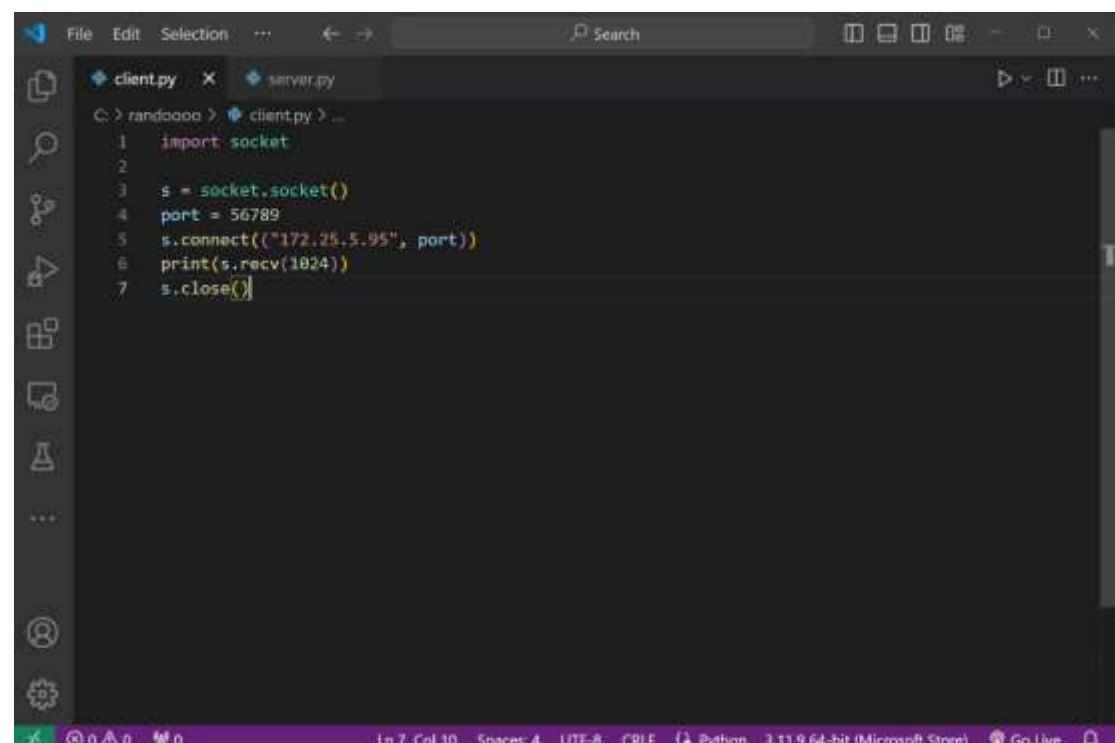
**Accepting Connections:** When a client attempts to connect, the server accepts the connection using the `accept()` method. This method returns a new socket object representing the connection and a tuple containing the client's address.



```
File Edit Selection ... Search
client.py server.py X
C:\> randomoo > server.py > ...
1 import socket
2 s = socket.socket()
3 print('Socket succesfully created')
4 port = 56789
5 s.bind(('', port))
6 print(f'socket binded to port{port}')
7 s.listen(5)
8 print('Socket is listening')
9 while True:
10     c, addr = s.accept()
11     print('Got connection from', addr)
12     message = ('Thank you for connecting')
13
14     c.send(message.encode())
15     c.close()
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store) Go Live

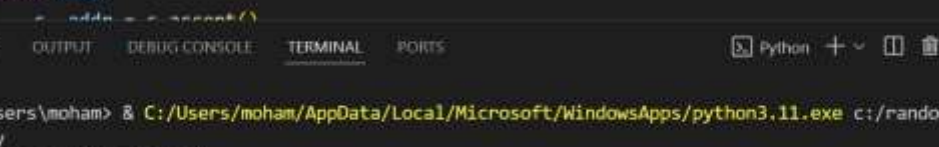
Server.py code used to test the relationship



```
File Edit Selection ... Search
client.py X server.py
C:\> randomoo > client.py > ...
1 import socket
2
3 s = socket.socket()
4 port = 56789
5 s.connect(("172.25.5.95", port))
6 print(s.recv(1024))
7 s.close()
```

Ln 7, Col 10 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store) Go Live

Client.py code used to test the relationship



The screenshot shows a VS Code editor with a Python script in the background and a terminal window in the foreground. The script defines a simple TCP server that listens on port 56789 and prints the received data. The terminal shows the command to run the script and its successful execution.

```
8 print('Socket is listening')
9 while True:
10     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     s.bind(('localhost', 56789))
12     s.listen(5)
13     print('Server is listening on port 56789')
14     client, address = s.accept()
15     data = client.recv(1024)
16     print('Received data from %s: %s' % address, data)
17     client.send('Hello, %s!' % address)
18     client.close()
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + - [ ] [X] ... ^ X

PS C:\Users\moham> & C:/Users/moham/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/randooooo/s  
erver.py  
Socket succesfully created  
socket binded to port56789  
Socket is listening

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.9 64-bit (Microsoft Store) Go Live

Test completed successfully! Client-server model is running.

The image displays two Windows PowerShell windows side-by-side, illustrating a simple network communication setup.

**Left Window (Client):**

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! ht
tps://aka.ms/PSWindows

PS C:\Users\moham> cd ..
PS C:\Users> cd ..
PS C:\> cd randoooo
PS C:\randoooo> ls

Directory: C:\randoooo

Mode                LastWriteTime         Length Name
----                -
-a-----         2024/04/12      15:47             118 client.py
-a-----         2024/04/11      01:05             361 server.py
-a-----         2024/04/12      15:27      134525 serversnippet.png
-a-----         2024/04/12      15:26       89226 snippet.png

PS C:\randoooo> python client.py
b'Thank you for connecting'
PS C:\randoooo>
  
```

**Right Window (Server):**

```

Windows PowerShell

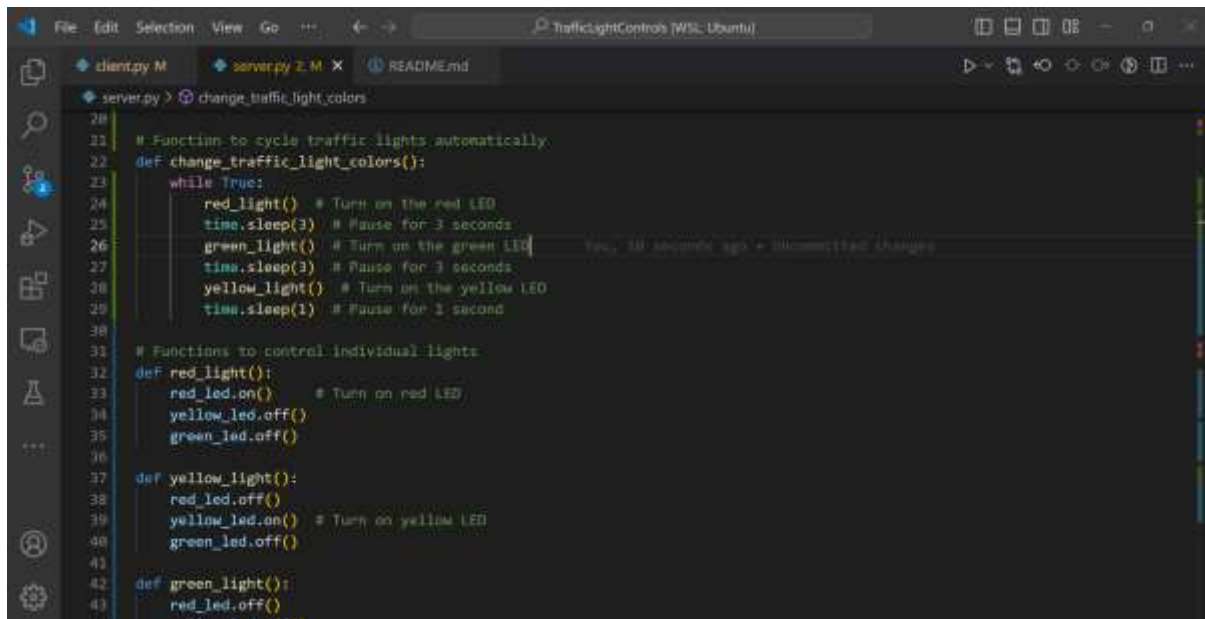
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo>
PS C:\randoooo> ls

Directory: C:\randoooo

Mode                LastWriteTime         Length Name
----                -
-a-----         2024/04/12      15:47             118 client.py
-a-----         2024/04/11      01:05             361 server.py
-a-----         2024/04/12      15:27      134525 serversnippet.png
-a-----         2024/04/12      15:26       89226 snippet.png

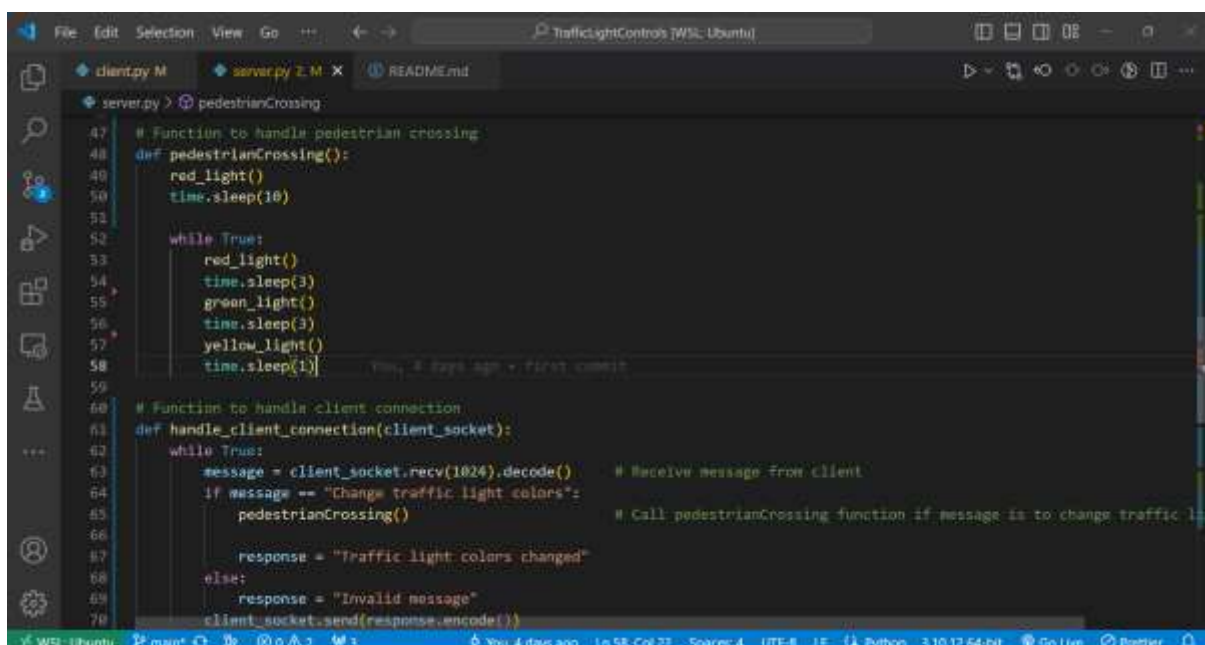
PS C:\randoooo> python server.py
Socket successfully created
socket binded to port 56789
Socket is listening
Got connection from ('10.0.0.111', 58886)
  
```

## SERVER.py SCRIPT



```
20
21 # Function to cycle traffic lights automatically.
22 def change_traffic_light_colors():
23     while True:
24         red_light() # Turn on the red LED
25         time.sleep(3) # Pause for 3 seconds
26         green_light() # Turn on the green LED
27         time.sleep(3) # Pause for 3 seconds
28         yellow_light() # Turn on the yellow LED
29         time.sleep(1) # Pause for 1 second
30
31 # Functions to control individual lights
32 def red_light():
33     red_led.on() # Turn on red LED
34     yellow_led.off()
35     green_led.off()
36
37 def yellow_light():
38     red_led.off()
39     yellow_led.on() # Turn on yellow LED
40     green_led.off()
41
42 def green_light():
43     red_led.off()
```

The above code snippet defines functions to control traffic lights. The `change_traffic_light_colors()` function cycles through red, green, and yellow lights with specified time intervals using `time.sleep()`. The `red_light()`, `yellow_light()`, and `green_light()` functions control individual LEDs by turning them on or off accordingly.



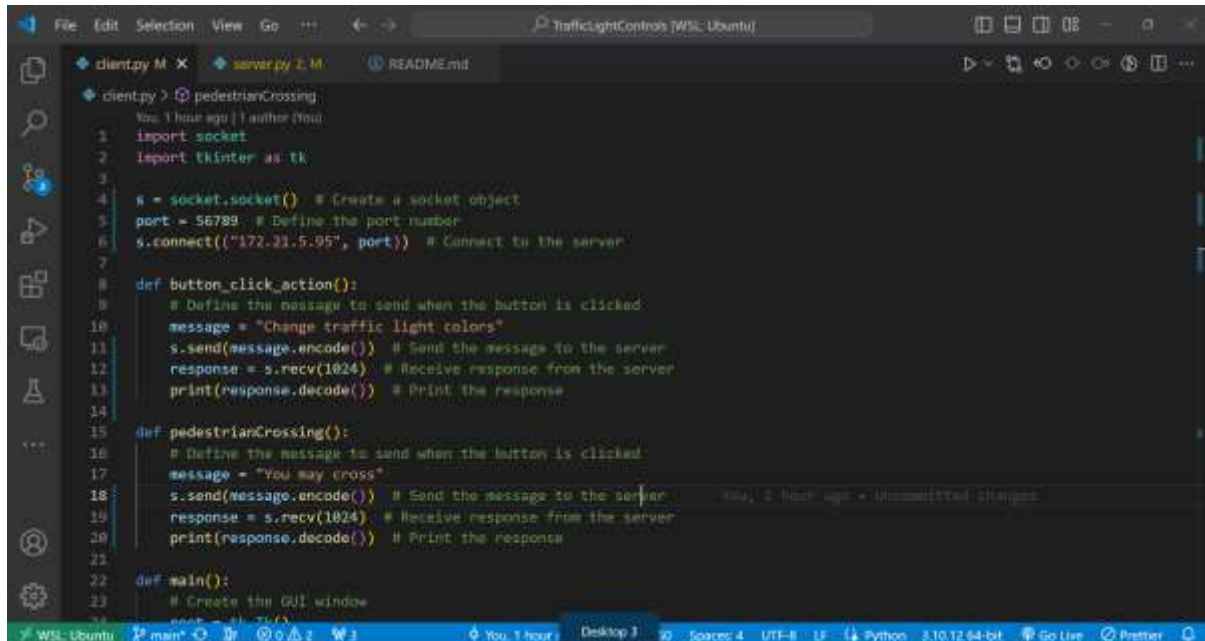
```
47 # Function to handle pedestrian crossing
48 def pedestrianCrossing():
49     red_light()
50     time.sleep(10)
51
52     while True:
53         red_light()
54         time.sleep(3)
55         green_light()
56         time.sleep(3)
57         yellow_light()
58         time.sleep(1)
59
60 # Function to handle client connection
61 def handle_client_connection(client_socket):
62     while True:
63         message = client_socket.recv(1024).decode() # Receive message from client
64         if message == "Change traffic light colors":
65             pedestrianCrossing() # Call pedestrianCrossing function if message is to change traffic light colors
66
67             response = "Traffic light colors changed"
68         else:
69             response = "Invalid message"
70         client_socket.send(response.encode())
```

The above code snippet includes two functions.

The `pedestrianCrossing()` function initiates a pedestrian crossing sequence. Initially, it sets the traffic light to red, waits for 10 seconds, then enters a loop where it alternates between red, green, and yellow lights with specified time intervals.

The `handle_client_connection(client_socket)` function continuously listens for messages from a client. If the received message is "Change traffic light colors," it initiates the pedestrian crossing sequence by calling the `pedestrianCrossing()` function and sends back a response indicating that the traffic light colors have changed. If the message is not recognized, it sends back a response indicating that the message is invalid.

## CLIENT.py SCRIPT



```
client.py M X server.py 2 M README.md
client.py > pedestrianCrossing
You, 1 hour ago {} author (You)
1 import socket
2 import tkinter as tk
3
4 s = socket.socket() # Create a socket object
5 port = 56789 # Define the port number
6 s.connect(("172.21.5.95", port)) # Connect to the server
7
8 def button_click_action():
9     # Define the message to send when the button is clicked
10    message = "Change traffic light colors"
11    s.send(message.encode()) # Send the message to the server
12    response = s.recv(1024) # Receive response from the server
13    print(response.decode()) # Print the response
14
15 def pedestrianCrossing():
16     # Define the message to send when the button is clicked
17     message = "You may cross"
18     s.send(message.encode()) # Send the message to the server
19     response = s.recv(1024) # Receive response from the server
20     print(response.decode()) # Print the response
21
22 def main():
23     # Create the GUI window
24     root = tk.Tk()
25     root.title("Traffic Light Controls")
26     root.geometry("300x100")
27     button = tk.Button(root, text="Change Traffic Light Colors", command=button_click_action)
28     button.pack()
29     button = tk.Button(root, text="Pedestrian Crossing", command=pedestrianCrossing)
30     button.pack()
31     root.mainloop()
32
33 if __name__ == '__main__':
34     main()
```

The above code snippet establishes a client connection to a server using sockets and Tkinter for GUI.

It creates a socket object `s`.

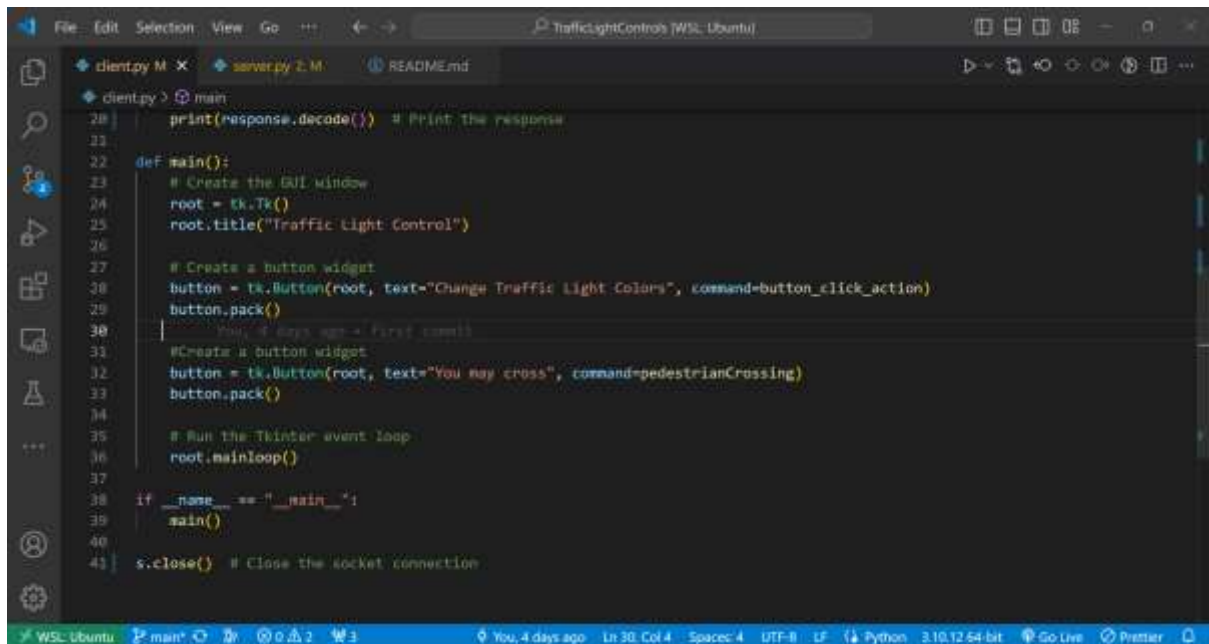
Defines the port number to connect to.

Connects to the server at the specified IP address and port.

The `button_click_action()` function is called when a button is clicked. It sends the message "Change traffic light colors" to the server, receives a response, and prints it.

The `pedestrianCrossing()` function sends the message "You may cross" to the server when called and prints the response received.



A screenshot of a code editor window titled 'TrafficLightControls [WSL: Ubuntu]'. The editor shows a Python file named 'client.py' with a 'main' function. The code includes comments and function calls for creating a GUI window, adding buttons, and starting the Tkinter event loop. The status bar at the bottom indicates the file is 'main.py', it was edited '4 days ago', and is using 'Python 3.10.12 64-bit'.

```
20 | print(response.decode()) # Print the response
21 |
22 | def main():
23 |     # Create the GUI window
24 |     root = tk.Tk()
25 |     root.title("Traffic Light Control")
26 |
27 |     # Create a button widget
28 |     button = tk.Button(root, text="Change Traffic Light Colors", command=button_click_action)
29 |     button.pack()
30 |     # You, 4 days ago - First commit
31 |     # Create a button widget
32 |     button = tk.Button(root, text="You may cross", command=pedestrianCrossing)
33 |     button.pack()
34 |
35 |     # Run the Tkinter event loop
36 |     root.mainloop()
37 |
38 | if __name__ == "__main__":
39 |     main()
40 |
41 | s.close() # Close the socket connection
```

The above code snippet defines a `main()` function to create a GUI window using Tkinter. It sets up two buttons: one for changing traffic light colors and another for indicating pedestrian crossing. The buttons are linked to their respective functions, `button_click_action()` and `pedestrianCrossing()`, using the `command` parameter. Finally, it starts the Tkinter event loop to handle user interactions with the GUI.

## Conclusion

Throughout the course of this project, our team encountered a myriad of challenges and triumphs that enriched our understanding of IoT systems, Python programming, and traffic management principles. Navigating through the intricacies of traffic management and hardware configuration provided invaluable hands-on experience. Moving forward, potential improvements could include integrating sensors for real-time traffic detection, refining traffic algorithms for enhanced efficiency, and exploring options for remote monitoring and control. Despite the challenges, this project has been a rewarding journey, fueling our enthusiasm for innovation and inspiring future endeavors in the realm of smart city technologies.

**References:**

Real Python,(n.d).Python on Raspberry Pi. Available at: <https://realpython.com/python-raspberry-pi/> .(Accessed : 09 April 2024)

Raspberry Pi Foundation.(n.d). Physical computing with python: LED brightness control. Available: <https://projects.raspberrypi.org/en/projects/physicalcomputing/9> .(Accessed: 09 April 2024)

<https://docs.python.org/3/library/threading.html>

<https://www.geeksforgeeks.org/multithreading-python-set-1/>

<https://realpython.com/python-raspberry-pi/>

<https://realpython.com/python-sockets/>

<https://sites.google.com/site/namrataprojects/home/traffic-monitoring-system-using-raspberry-pi-and-thingsboard>

[https://www.youtube.com/watch?v=loGWUgnRIHs&ab\\_channel=NaveenMayantha](https://www.youtube.com/watch?v=loGWUgnRIHs&ab_channel=NaveenMayantha)

[https://www.youtube.com/watch?v=KeXSfX8Pgo0&t=4s&ab\\_channel=voidloopRobotech%26Automation](https://www.youtube.com/watch?v=KeXSfX8Pgo0&t=4s&ab_channel=voidloopRobotech%26Automation)

[https://www.youtube.com/watch?v=sN0r6Jz9dvl&ab\\_channel=BekBrace](https://www.youtube.com/watch?v=sN0r6Jz9dvl&ab_channel=BekBrace)

[https://www.youtube.com/watch?v=DeFST8vtul&ab\\_channel=NeuralNine](https://www.youtube.com/watch?v=DeFST8vtul&ab_channel=NeuralNine)

[https://www.youtube.com/watch?v=B8VX0C48Y9w&ab\\_channel=CytronTechnologies](https://www.youtube.com/watch?v=B8VX0C48Y9w&ab_channel=CytronTechnologies)

[https://www.youtube.com/watch?v=xfQIPWFISgQ&t=14s&ab\\_channel=AlexanderBaran-Harper](https://www.youtube.com/watch?v=xfQIPWFISgQ&t=14s&ab_channel=AlexanderBaran-Harper)

[https://www.youtube.com/watch?v=64GYKAp8a-0&ab\\_channel=SSTecTutorials](https://www.youtube.com/watch?v=64GYKAp8a-0&ab_channel=SSTecTutorials)

**Group Members:**

Mohamed Asad Bandarkar(Group Leader) 4271451

Omphile Thipe 4148120

Phidzaglima Ntanganedzeni 4270626

Jaydin Morrison 4260345

Junior Mukiza 4265028

Oluphi Vukaphi 4037881

Maluleke Vukosi Jayson 4279123

Netshishivhe Aluwani 4135656

Phumela Cenge 4135664

Mahdi Galant 4252863