LIVE •

VIDEO STREAMING

WITH PYTHON

GROUP MEMBERS

- MAYANK SHARMA
- NISHANT SHARMA
- RISHI RAJ SINGH
- SAI RAM REDDY

Live Video Streaming using Socket Programming with Python

Live Streaming

Streaming is the continuous transmission of audio or video files from a server to a client.

Socket Programming

It shows how to use socket APIs to connect links between remote and local processes

Why Python?

Python provides 2levels access to network, and have already-made libraries

CN PROJECT

USE OF VIDEO STREAMING

Reaching Highly Targeted Niches.

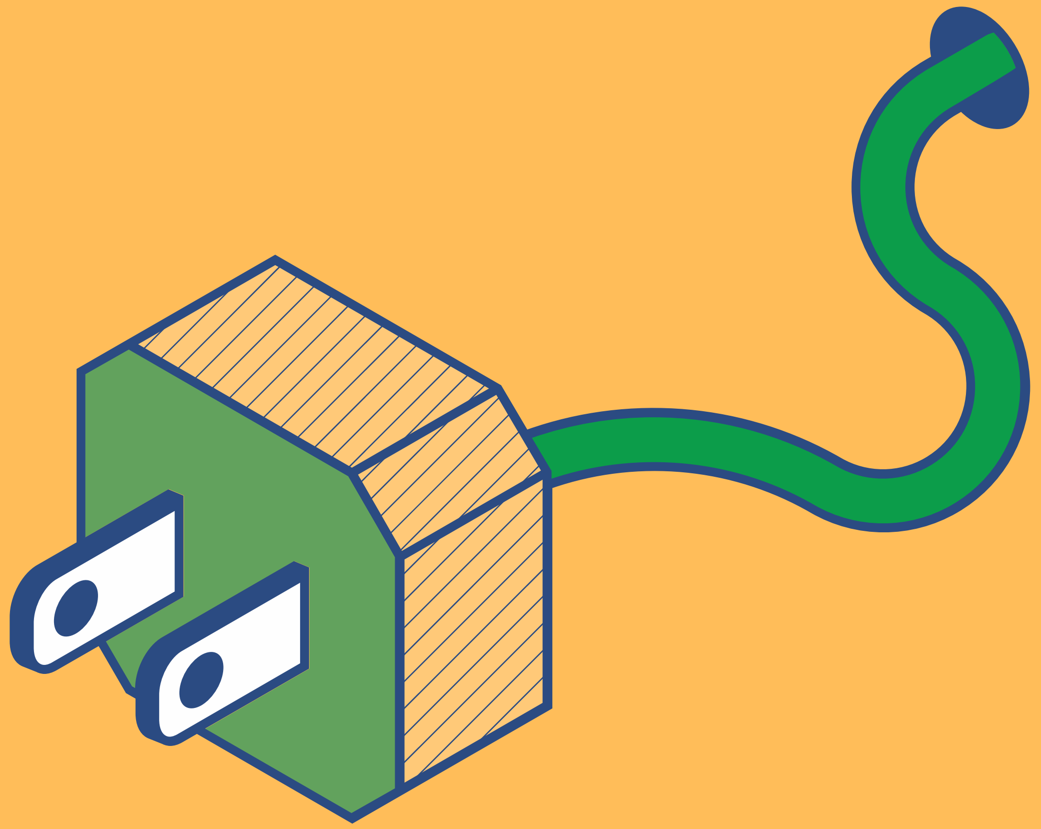
No Need to Download the content

Real Time Engagement

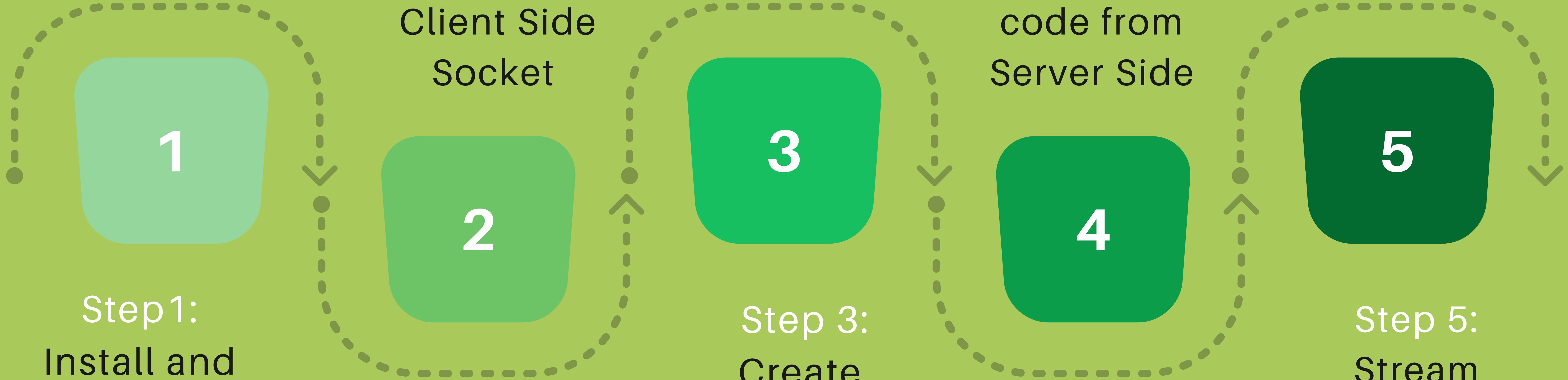
WHAT IS SOCKET PROGRAMMING

Sockets and the socket API are used to send messages across a network. They provide a form of IPC.

The most common type of socket applications are client-server application where one side acts as the server and waits for connections from clients, this is the type of live video streaming application we are going to build.



STEPS FOR **SOURCE** CODE



Step 1:
Install and import the open libraries.

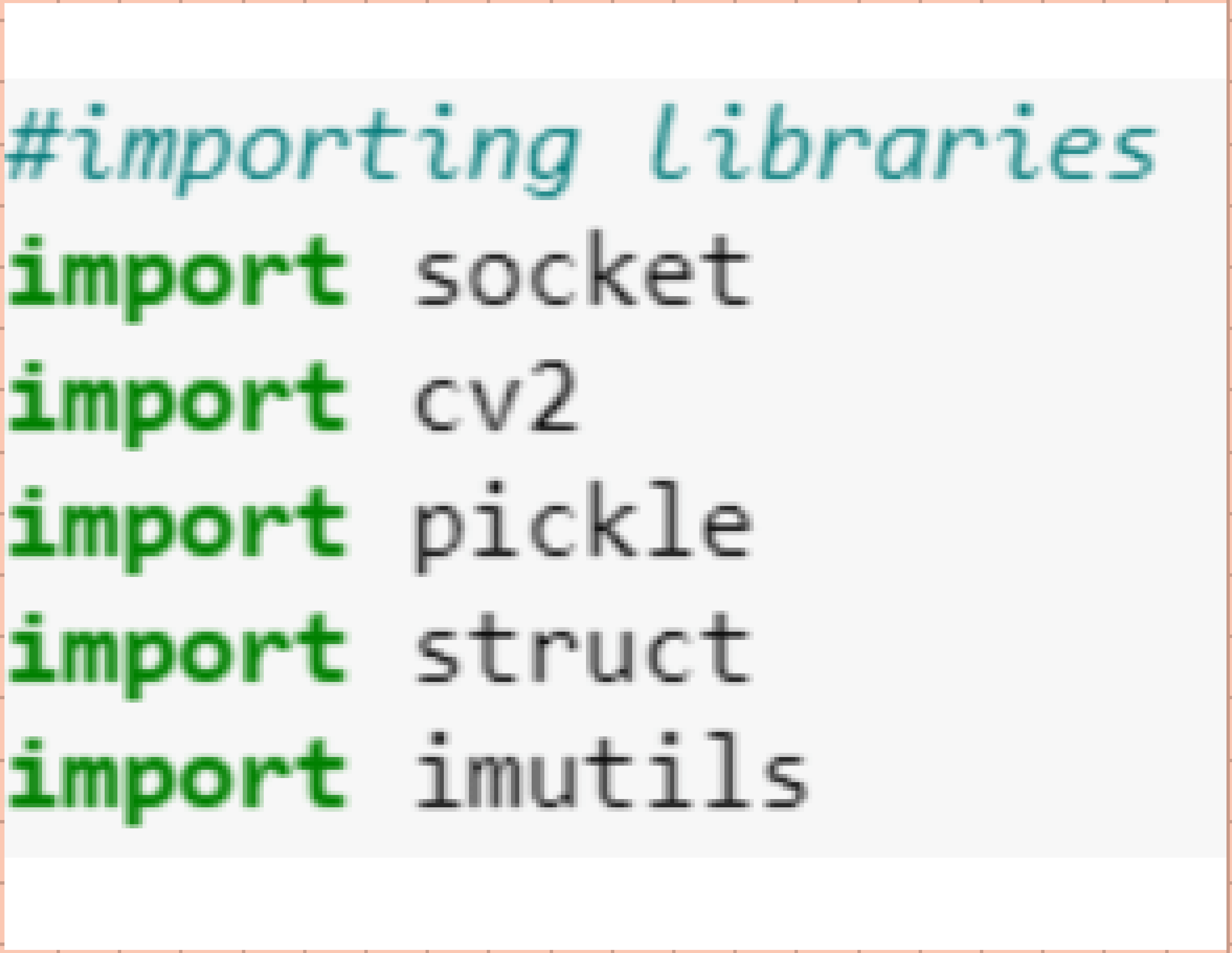
Step 2:
Create Client Side Socket

Step 3:
Create Server Side Socket

Step 4:
Stream video code from Server Side

Step 5:
Stream video code from Client Side

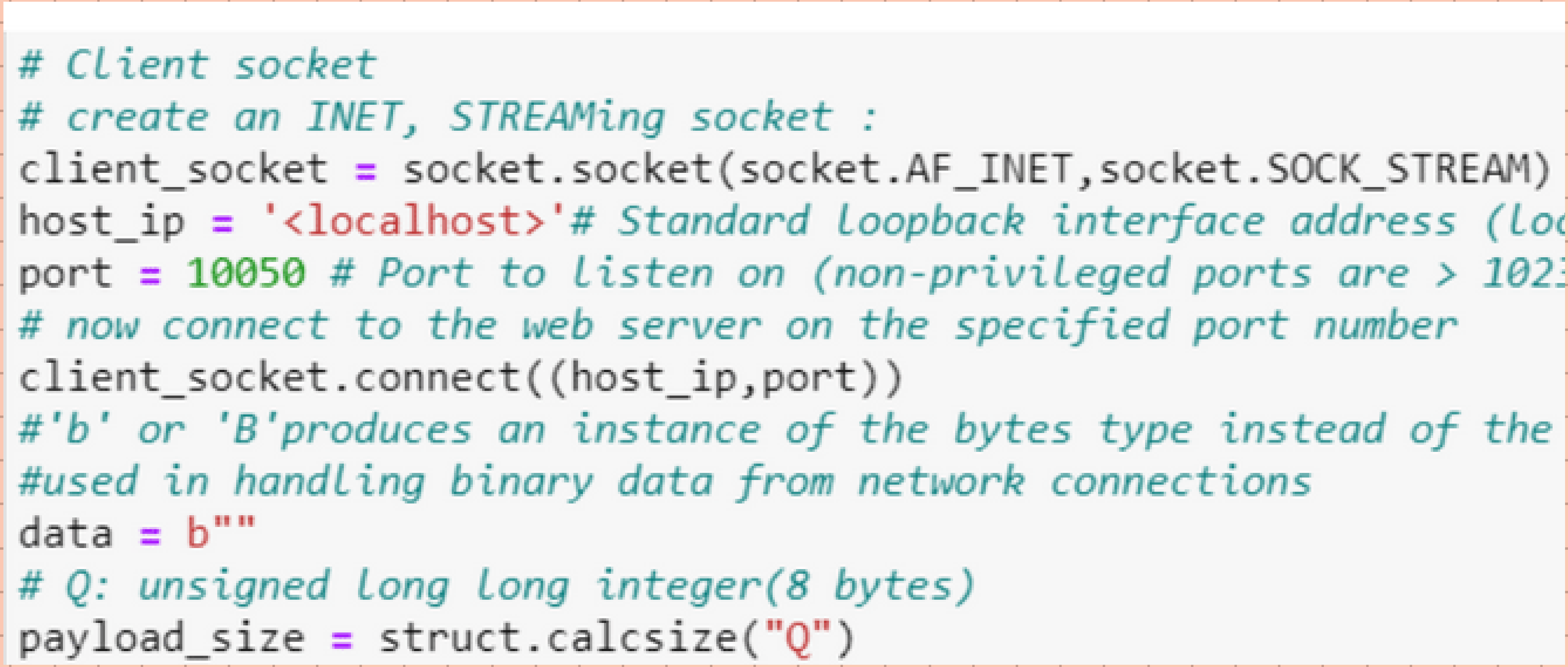
Step1 INSTALL AND IMPORT THE FOLLOWING LIBRARIES IN BOTH THE FILES



```
#importing libraries
import socket
import cv2
import pickle
import struct
import imutils
```

socket: Get socket module from python
cv2: To import Computer Vision module from python
pickle: for serializing and de-serializing python object structures.
struct: to convert native Python data types into a string of bytes
imutils: Image processing operations

Client
CODE



```python
# Client socket
# create an INET, STREAMing socket :
client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_ip = '<localhost>'# Standard loopback interface address (lo
port = 10050 # Port to listen on (non-privileged ports are > 102
# now connect to the web server on the specified port number
client_socket.connect((host_ip,port))
#'b' or 'B'produces an instance of the bytes type instead of the
#used in handling binary data from network connections
data = b""
# Q: unsigned long long integer(8 bytes)
payload_size = struct.calcsize("Q")
```

Client

CODE - EXPLAINATION

client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM))-> we create a object of socket “client_socket”.

AF_INET means it is a Ipv4 address.

SOCK_STREAM means that it is a TCP socket.

host_ip = ‘<local address>’-> specifies host ip address.

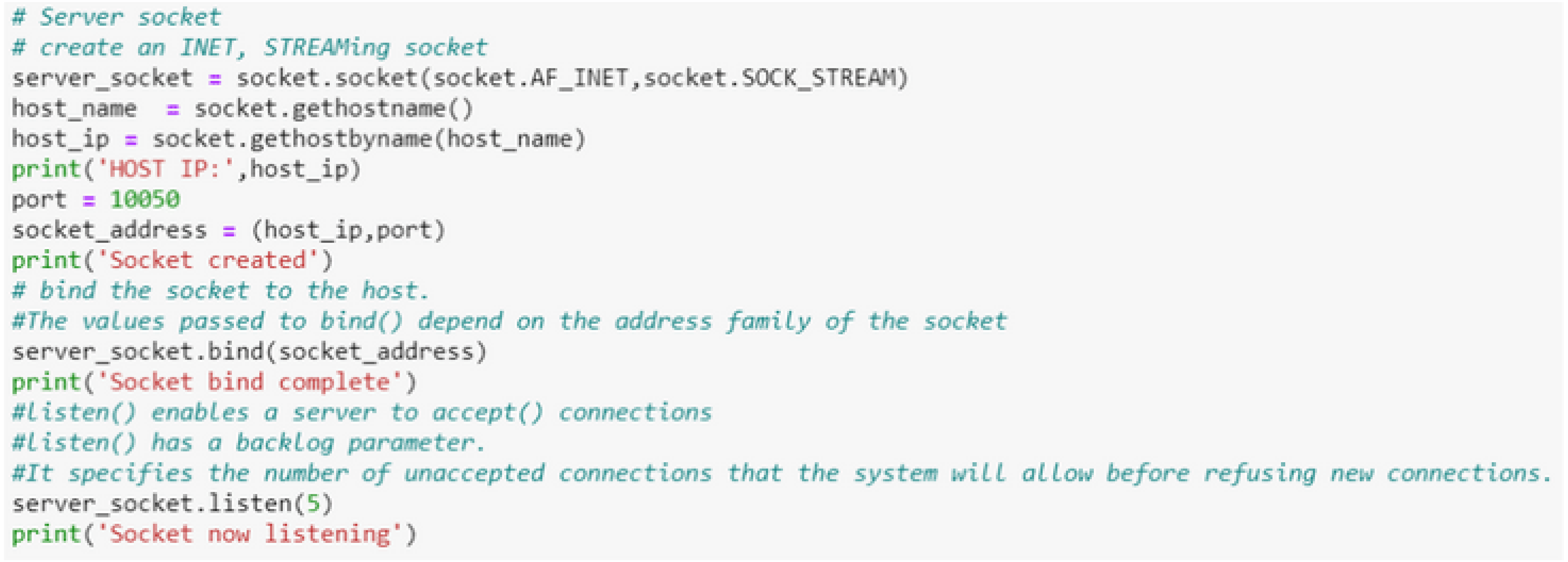
port = 10050-> server port number.

client_socket.connect((host_ip,port))-> for connecting to the remote socket.

data = b""-> instance of variable data of byte string type.

payload_size = struct.calcsize("Q")->payload_size is defined by size of Q. Q is unsigned long long integer.

Source CODE



```
# Server socket
# create an INET, STREAMing socket
server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_name  = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print('HOST IP:',host_ip)
port = 10050
socket_address = (host_ip,port)
print('Socket created')
# bind the socket to the host.
#The values passed to bind() depend on the address family of the socket
server_socket.bind(socket_address)
print('Socket bind complete')
#listen() enables a server to accept() connections
#listen() has a backlog parameter.
#It specifies the number of unaccepted connections that the system will allow before refusing new connections.
server_socket.listen(5)
print('Socket now listening')
```

Source

CODE - EXPLAINATION

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

-> with this line we create a object of socket “server_socket”.

AF_INET means it is a Ipv4 address

SOCK_STREAM means that it is a TCP socket.

host_name = socket.gethostname()

host_ip = socket.gethostbyname(host_name)

these two lines is use for getting hostname and IP address of a local machine.

port = 10050 -> for specifying port no.

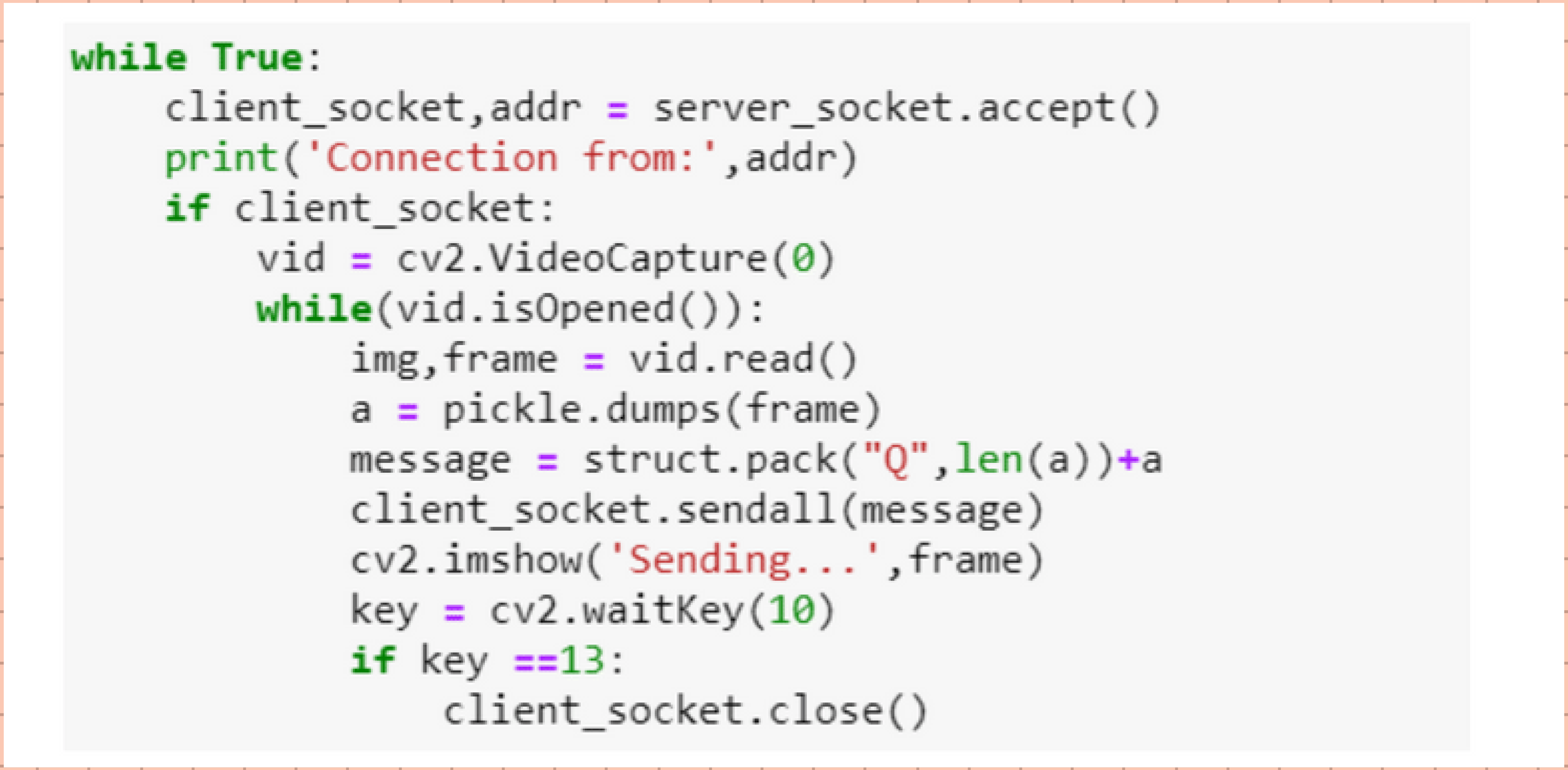
socket_address = (host_ip,port)

server_socket.bind(socket_address)

Assigns a socket to an address.

Stream Video

CODE
SERVER SIDE



```python
while True:
    client_socket,addr = server_socket.accept()
    print('Connection from:',addr)
    if client_socket:
        vid = cv2.VideoCapture(0)
        while(vid.isOpened()):
            img,frame = vid.read()
            a = pickle.dumps(frame)
            message = struct.pack("Q",len(a))+a
            client_socket.sendall(message)
            cv2.imshow('Sending...',frame)
            key = cv2.waitKey(10)
            if key ==13:
                client_socket.close()
```

Stream Video

CODE EXPLANATION

SERVER SIDE

Server Side (i) Listening for any client request, client request +address.

(ii) If Client request –

(a) Start Video Capture

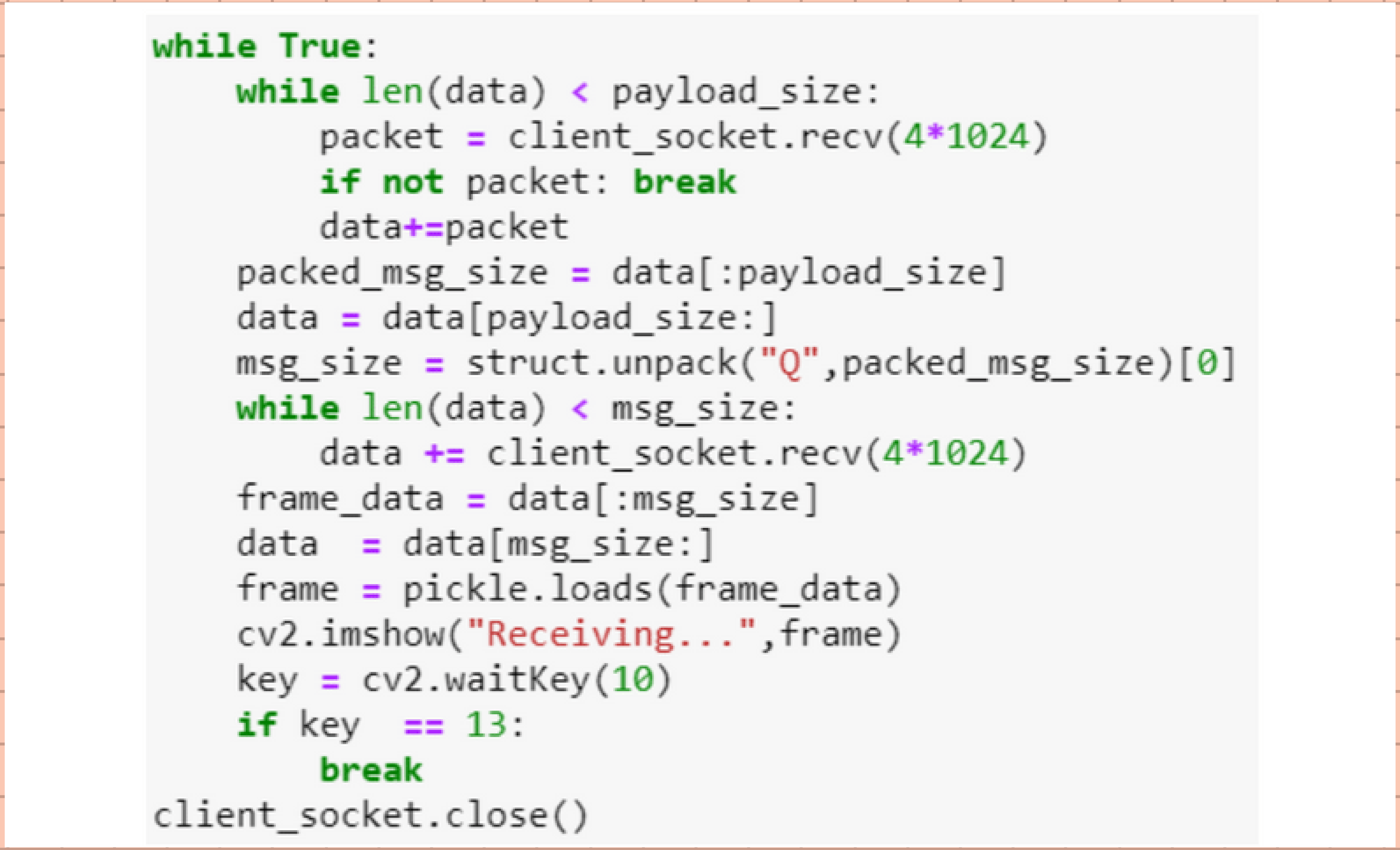
(b) While Loop (Checking if vdo looking or not)

1. Divide vdo in img and frame (important is frame).
2. Dump the frame (Using pickle).
3. Creating its message.
4. Send it to client socket .
5. cv.imshow – it will show which frame is transmitted.
6. Waitkey (press any key, waits 10 sec).
7. If key pressed = 13, close client.

Stream Video

CODE

CLIENT SIDE



```python
while True:
    while len(data) < payload_size:
        packet = client_socket.recv(4*1024)
        if not packet: break
        data+=packet
    packed_msg_size = data[:payload_size]
    data = data[payload_size:]
    msg_size = struct.unpack("Q",packed_msg_size)[0]
    while len(data) < msg_size:
        data += client_socket.recv(4*1024)
    frame_data = data[:msg_size]
    data  = data[msg_size:]
    frame = pickle.loads(frame_data)
    cv2.imshow("Receiving...",frame)
    key = cv2.waitKey(10)
    if key  == 13:
        break
client_socket.close()
```

Stream Video

CODE EXPLANATION
CLIENT SIDE

Client Side

(i) Check if data < overall data size (takes data and append data)

(ii) Client_socket.rec(4*1024), here (4*1024) is data received in bytes.

(iii) Take all data.

(iv) Unnpack data.

(v) Check len and receive packets.

(vi) Load frame.

(vii) Show loaded frame.

Expected Result

Conclusion :

Hence, in this way we can create a live video streaming application without audio which can be used for purposes like surveillance, or may be to group study without disturbance etc.

ELEMENT IN SOURCE CODE

SOCKET

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

SERVER

In the server side we use accept() as it blocks and waits. When a client connects, it returns a new socket object representing the connection and a tuple. The tuple will contain (host, port) for IPv4 connections or (host, port, flowinfo, scopeid) for IPv6.

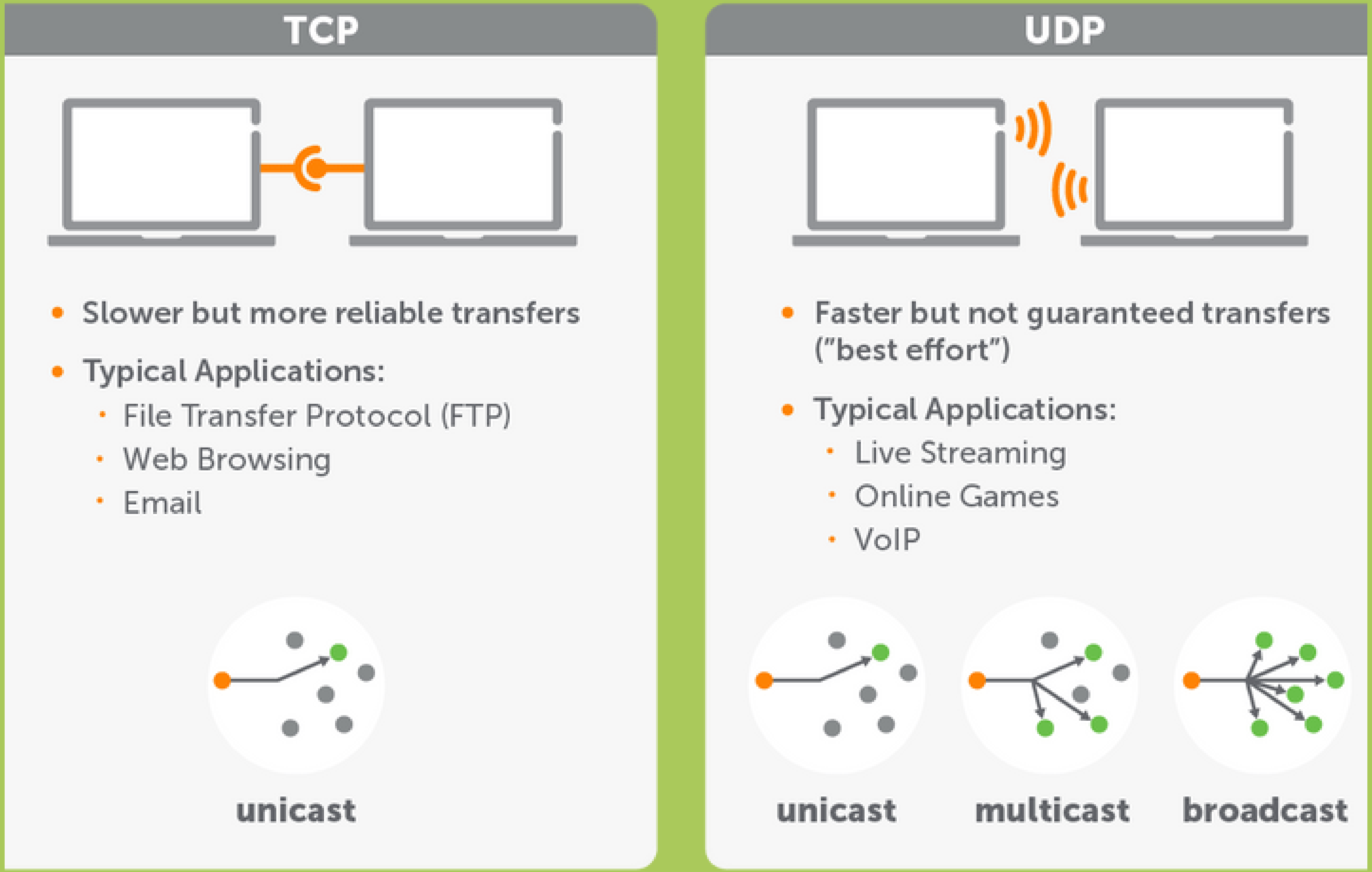
CLIENT

The client creates a socket object, connects to the server and calls s.recv() to read the server's reply . In the client side we receive data frames for the video streaming but in form of data packets

TCP

Transmission Control Protocol (TCP/IP) is a connection-oriented protocol. It is an intermediate layer of the application layer and internet protocol layer in the OSI model.

Streaming use the (UDP) or (TCP)?



TCP

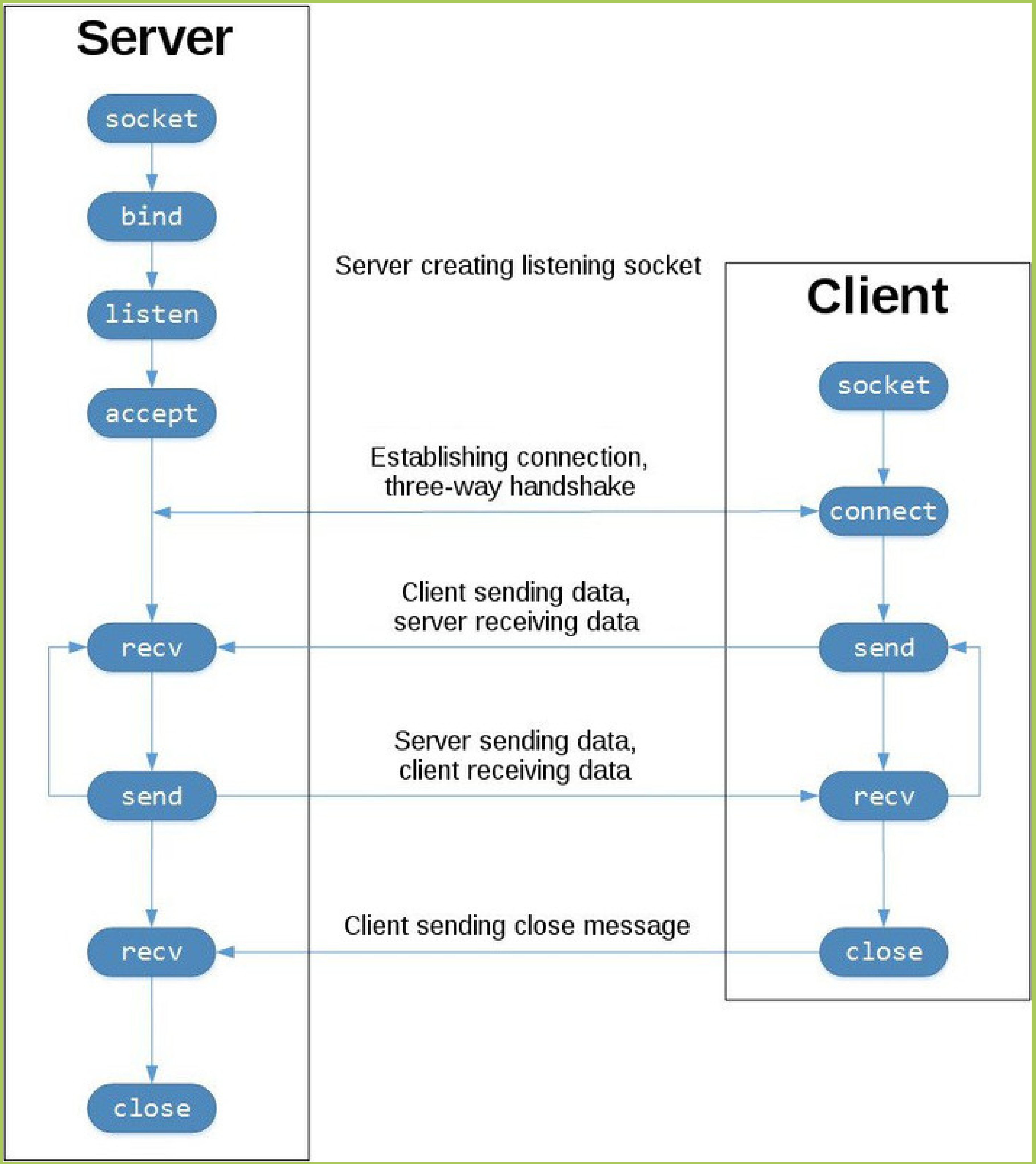
- Slower but more reliable transfers
- Typical Applications:
  - File Transfer Protocol (FTP)
  - Web Browsing
  - Email

UDP

- Faster but not guaranteed transfers ("best effort")
- Typical Applications:
  - Live Streaming
  - Online Games
  - VoIP

UDP is recommended over TCP, but for our objective, TCP is suitable.

1. Is reliable: packets dropped in the network are detected and retransmitted by the sender.
2. Has in-order data delivery.


Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

DATA FLOW FOR TCP:

The left represents the serverand right is the client.

- Socket () function creates a socket on client as well as server side
- Socket () function creates a socket on client as well as server side
- Listen () function is used only in TCP/IP protocol to connection establishment on server side
- Accept () function is called by the TCP/IP server.
- Connect () function is used on client side.
- Read () function is used to both side.
- Close () function is used to terminate the connection on both side.

Additional Feature:

01

Streaming video + Audio

Till now, we are able to create a live video streaming application without audio which can be used for purposes group study without disturbance etc.

We are trying to add audio also.

We are using TCP version socket here.

02

Monetization:

We are trying to add some real life application like collecting funds during live streaming.

Monetization Of The Video Content

- In this process, we can generate income through the videos, that we share online on any platform.
- This is usually achieved through advertising, subscriptions, or direct transactions.
- This is usually achieved through advertising, subscriptions, or direct transaction

One of the ways to monetize the video content is using the pay-per-view model by implementing video pay-walls. Generally, the incentive for the audience to pay for the live streaming content is more than the non-live content. For example, people are willing to pay, to access a live stream of sports events, remote attendance at a conference, and for events like music concerts, etc.

3 Profitable Monetization Models To Sell Videos Online

1. SVOD: Subscription Video On Demand
2. TVOD: Transactional Video On Demand
3. AVOD: Advertising Video On Demand

GITHUB LINK

https://mayank13-01.github.io/livevideostreaming/