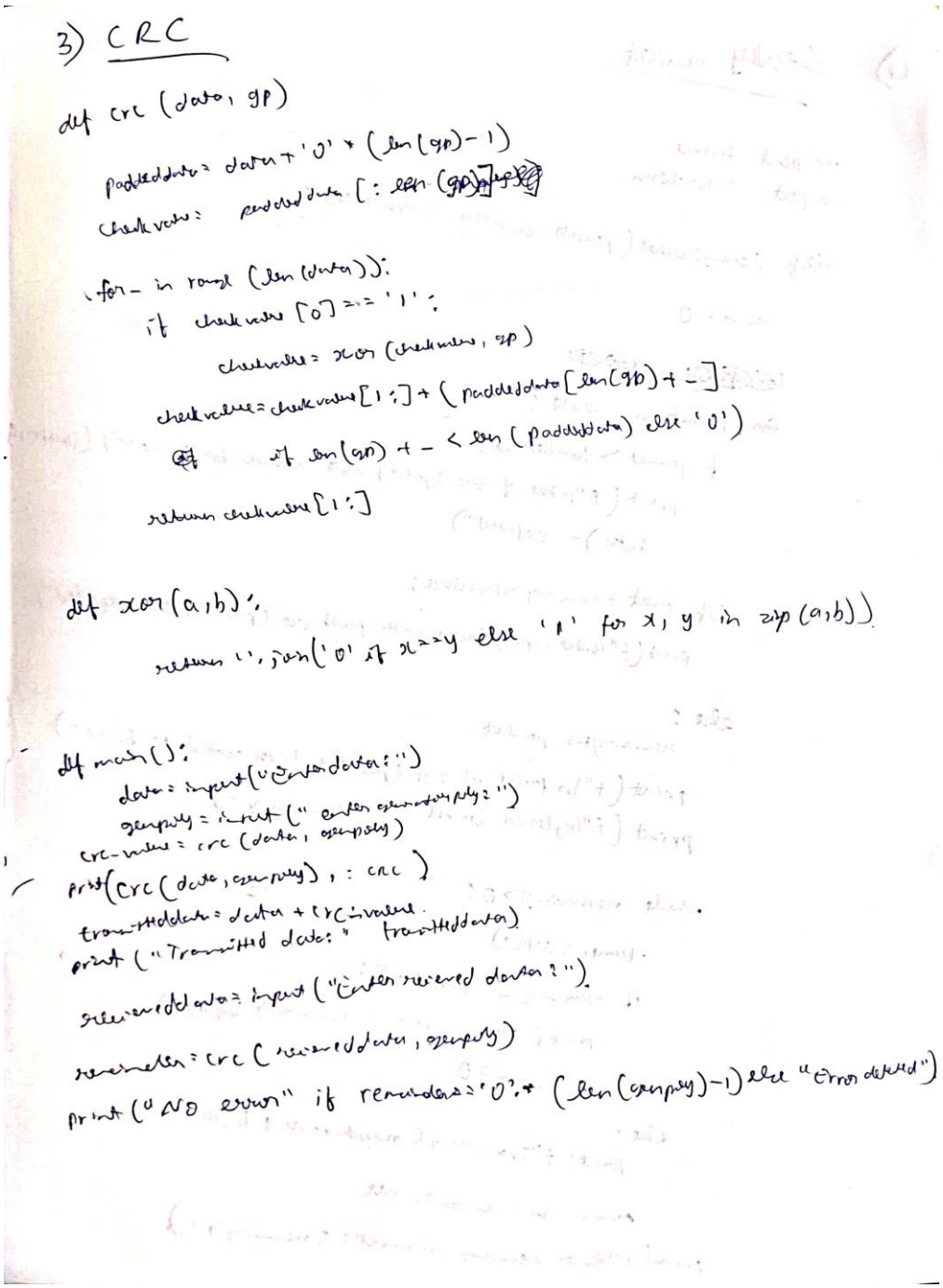
PART-B

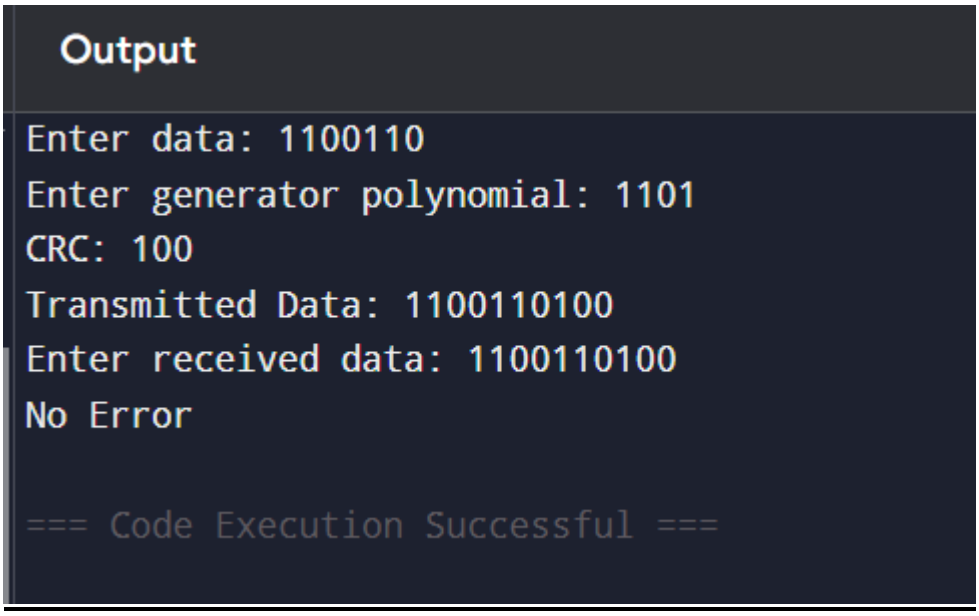
Program 14

Write a program for error detecting code using CRC-CCITT (16-bits).

Code :



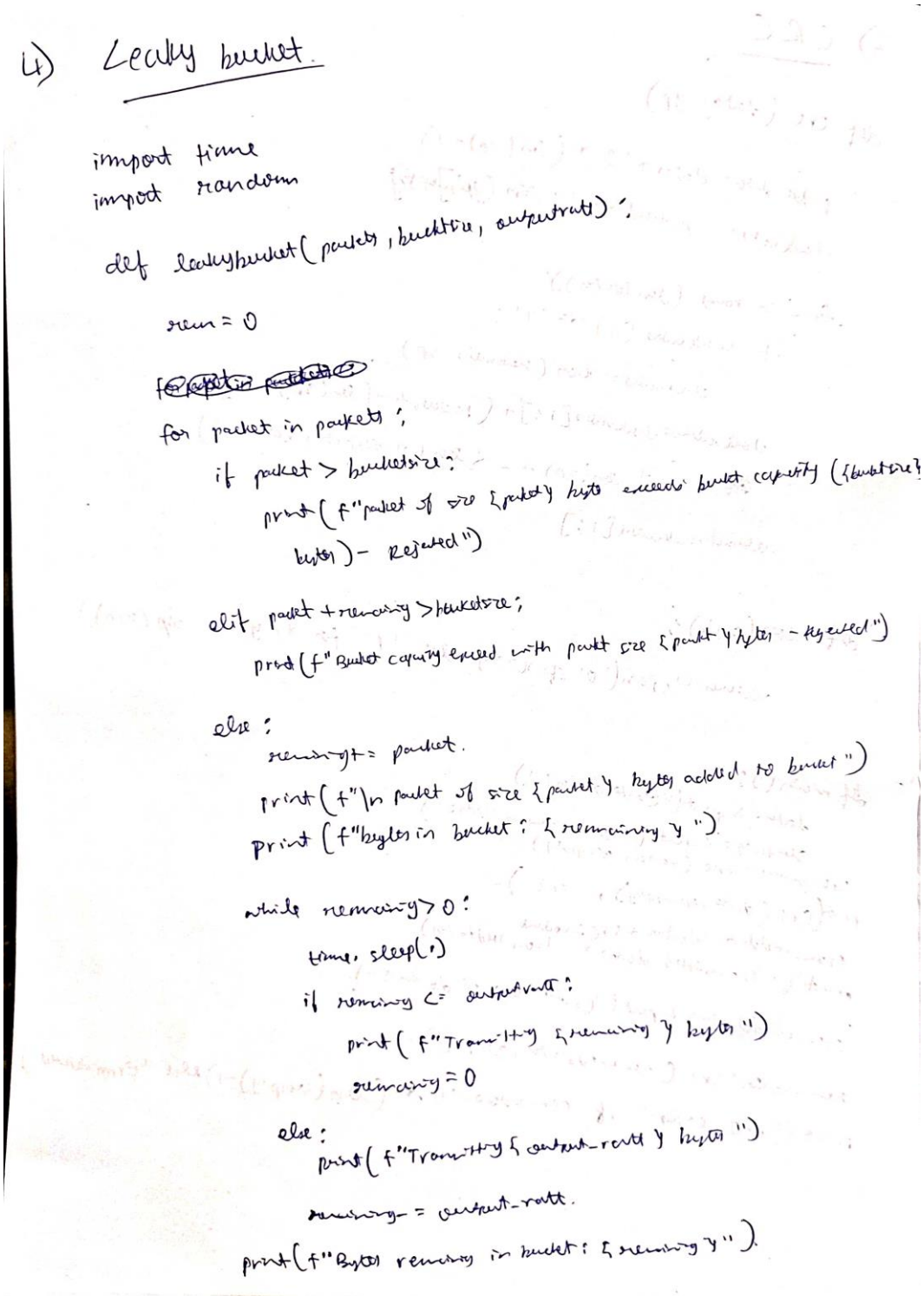
```
3) CRC

def crc (data, gp)
    paddeddata = data + '0' * (len(gp)-1)
    checkvalue = paddeddata [: len(gp)]

    for _ in range (len(data)):
        if checkvalue[0] == '1':
            checkvalue = xor(checkvalue, gp)
        checkvalue = checkvalue[1:] + (paddeddata[len(gp)+_]
            if len(gp)+_ < len(paddeddata) else '0')
    return checkvalue[1:]

def xor(a,b):
    return ''.join('0' if x==y else '1' for x, y in zip(a,b))

def main():
    data = input("Enter data: ")
    gpoly = input("enter generator poly: ")
    crc_value = crc(data, gpoly)
    print(crc(data, gpoly), : crc)
    transmitteddata = data + crc_value
    print("Transmitted data: " transmitteddata)
    receiveddata = input("Enter received data? ")
    remainder = crc(receiveddata, gpoly)
    print("NO error" if remainder == '0'* (len(gpoly)-1) else "Error detected")
```


Output
Enter data: 1100110
Enter generator polynomial: 1101
CRC: 100
Transmitted Data: 1100110100
Enter received data: 1100110100
No Error
=== Code Execution Successful ===

Program 15

Write a program for congestion control using Leaky bucket algorithm.

Code :



```
4) Leaky bucket

import time
import random

def leakybucket(packets, bucketsize, outputrate):
    rem = 0
    for packet in packets:
        if packet > bucketsize:
            print(f"packet of size {packet} bytes exceeds bucket capacity ({bucketsize}
              bytes) - Rejected")
        elif packet + remaining > bucketsize:
            print(f"Bucket capacity exceed with packt size {packet} bytes - Rejected")
        else:
            remaining += packet.
            print(f"\n packet of size {packet} bytes added to bucket")
            print(f"bytes in bucket : {remaining} ")
            while remaining > 0:
                time. sleep(1)
                if remaining <= outputrate:
                    print(f"Transmitting {remaining} bytes")
                    remaining = 0
                else:
                    print(f"Transmitting {outputrate} bytes")
                    remaining -= outputrate.
            print(f"Bytes remaining in bucket: {remaining}").
```

```
def main():
    packets = [random.randint(1,100) for _ in range(5)]
    print(f"Generated packets: {packets}")

    bucket_size = int(input("Enter bucket size: "))
    output_rate = int(input("Enter output rate: "))

    leaky_bucket(packets, bucket_size, output_rate)
```

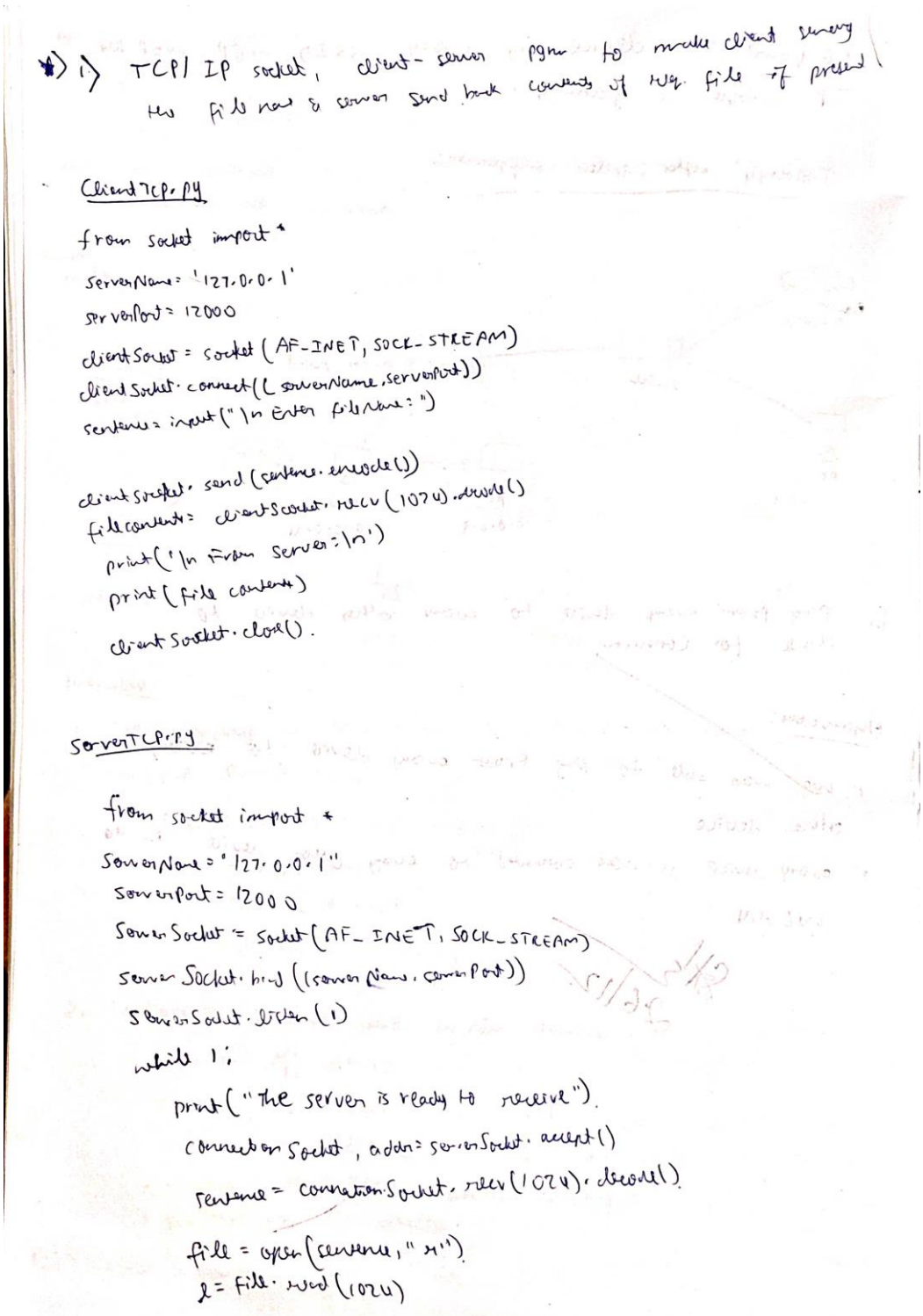
Output Clear

```
Generated packets: [80, 63, 57, 12, 69]
Enter bucket size: 60
Enter output rate: 30
Packet of size 80 bytes exceeds bucket capacity (60 bytes) - REJECTED
Packet of size 63 bytes exceeds bucket capacity (60 bytes) - REJECTED

Packet of size 57 bytes added to bucket
Bytes in bucket: 57
Transmitting 30 bytes
Bytes remaining in bucket: 27
Transmitting 27 bytes
Bytes remaining in bucket: 0

Packet of size 12 bytes added to bucket
Bytes in bucket: 12
Transmitting 12 bytes
Bytes remaining in bucket: 0
Packet of size 69 bytes exceeds bucket capacity (60 bytes) - REJECTED

=== Code Execution Successful ===
```

Program 16

Using TCP/IP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.

Code and Output:



1) TCP/IP socket, client-server pgm to make client send file name & server send back contents of req. file if present

ClientTCP.py

```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input("\n Enter filename: ")

clientSocket.send(sentence.encode())
filecontents = clientSocket.recv(1024).decode()
print('\n From Server:\n')
print(filecontents)
clientSocket.close()
```

ServerTCP.py

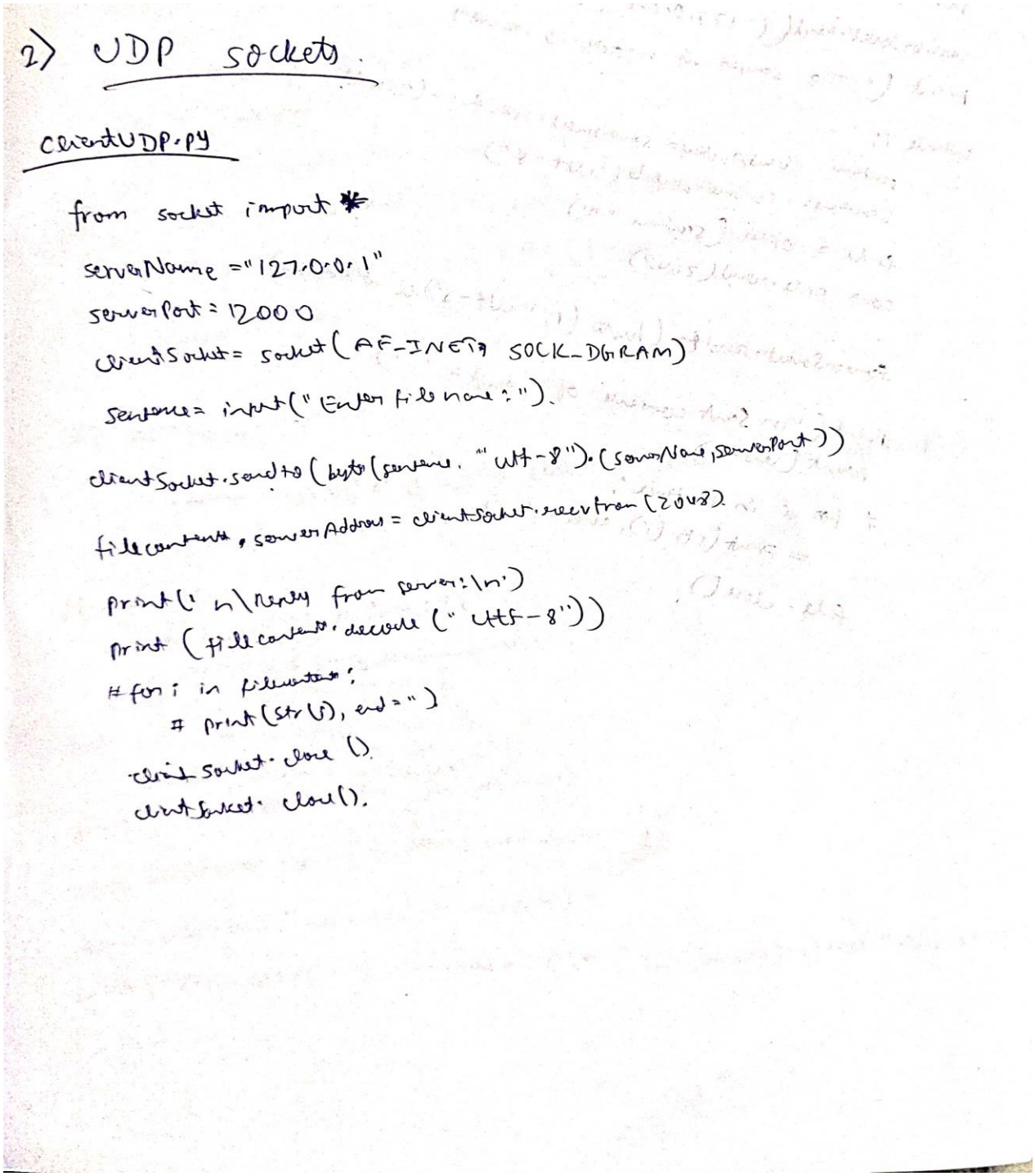
```
from socket import *
ServerName = '127.0.0.1'
ServerPort = 12000
ServerSocket = socket(AF_INET, SOCK_STREAM)
ServerSocket.bind((serverName, serverPort))
ServerSocket.listen(1)
while 1:
    print("The server is ready to receive")
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    file = open(sentence, "r")
    l = file.read(1024)
```

```
connectionSocket.send(l.encode())
print ("\n sent contents of ' + sentence)

file.close()
connectionSocket.close()
```

Program 17

Using UDP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.

Code and Output:



2) UDP sockets.

clientUDP.py

```
from socket import *
serverName = "127.0.0.1"
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
sentence = input("Enter file name :")
clientSocket.sendto(bytes(sentence, "utf-8"), (serverName, serverPort))
fileContents, serverAddress = clientSocket.recvfrom(2048)
print('\n\nReply from server:\n')
print(fileContents.decode("utf-8"))
# for i in fileContents:
    # print(str(i), end="")
clientSocket.close()
clientSocket.close().
```

ServerUDP.py.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("127.0.0.1", serverPort))
print("The server is ready to receive")
while 1:
    sentence, clientAddress = serverSocket.recvfrom(2048)
    sentence = sentence.decode("utf-8")
    file = open(sentence, "r")
    con = file.read(2048)
    serverSocket.sendto(bytes(1, "utf-8"),clientAddress)
    print('\n Sent contents of', end = ' ')
    print(sentence)
    # for i in sentence:
        # print(str(i), end = ' ')
    file.close()
```

WIRESHARK

Wireshark

It is a powerful used network protocol analyzer. It allows you to capture and inspect data packets traveling over a network in real time, making it a crucial tool for studying computer networks, troubleshooting network issues and understanding protocols.

Features:

1. Packet capture: captures live network traffic from various interfaces (eg. Wifi)
2. Protocol analysis: supports 100's of protocols like TCP, UDP.
3. Filtering: isolate specific packets.
6. Visualization: display packet details with hierarchical layers

Use cases:

1. Network Troubleshooting:
   - Diagnosing slow network speeds.
   - Identifying bottlenecks or misconfigurations.
2. Security analysis:
   - detecting malicious traffic or intrusions.
3. Protocol Study:
   - understanding packet structures & communication flows.

common filters:

- http: shows only HTTP traffic
- tcp.port == 80 : shows traffic on TCP port 80
- ip.addr == 192.168.1.1 : shows pkt to or from a single specific IP address.
- UDP : shows only UDP traffic.