HAMMING CODE AND CRC USING SOCKETS (ERROR CORRECTION AND DETECTION TECHNIQUES)

CSE1004(NETWORK AND COMMUNICATION)LAB:L53-L54

MARCH 3, 2022

ANIRUDH VADERA
20BCE2940

QUESTION:

PERFORM ERROR CORRECTION AND DETECTION TECHNIQUES USING SOCKETS

1. **HAMMING CODE**
2. **CRC**

HAMMING CODE:

Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is **technique developed by R.W. Hamming for error correction**.

Redundant bits –

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

The number of redundant bits can be calculated using the following formula:

$2^r \geq m + r + 1$

where, r = redundant bit, m = data bit

Suppose the number of data bits is 7, then the number of redundant bits can be calculated using:

$= 2^4 \geq 7 + 4 + 1$

Thus, the number of redundant bits= 4

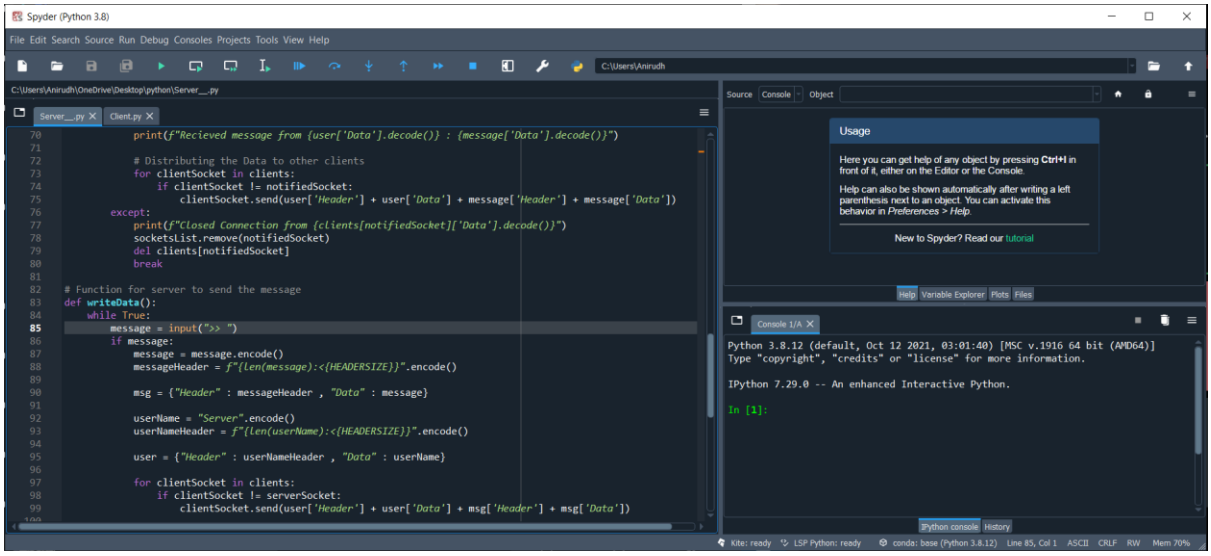
Parity bits –

A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data is even or odd. Parity bits are used for error detection. There are two types of parity bits:

1. **Even parity bit:**
   In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.
2. **Odd Parity bit –**
   In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

PROCEDURE:

- ➔ First Open your python ide
- ➔ I will be using anaconda distribution and a spyder IDE



- ➔ We will be using 2 files for our purpose
- ➔ A server file
- ➔ A client file

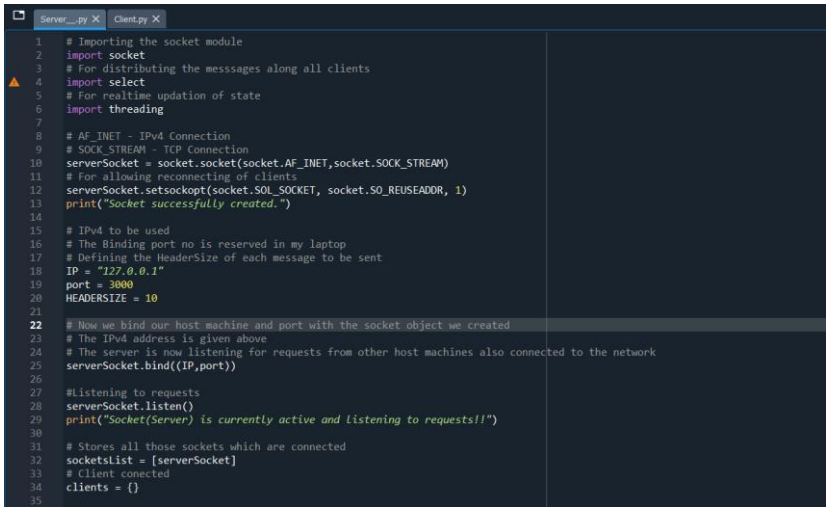
There are some common steps to be followed explained below

- ➔ A detailed explanation along with the code is given further below

Server.py: (Reciever)

1. Import the necessary files.
2. Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)
3. Bind the server using socket.bind(IP,port) method providing the IP and the port.
4. We now define the socketsList which stores all the sockets currently in action and make a client Dictionary which stores information about the clients.
5. We then define a function for reading messages using socket.recv() method
6. We then make a function for writing the messages to the client using the socket.send()
7. We then implement the hamming code logic

A code snippet for server.py:

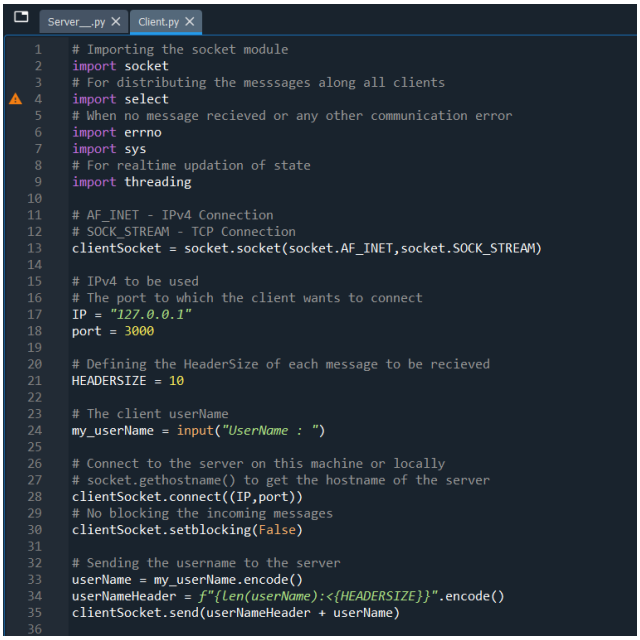


```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000
HEADERSIZE = 10

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}
```

Client.py: (Sender)

1. **Import the necessary files.**
2. **Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)**
3. **Connect to the server using socket.connect(IP) function by providing the appropriate IP address**
4. **Select a username and send it to the server.**
5. **We then define a function for reading messages using socket.recv() method**
6. **We try to catch as many errors as possible in it.**
7. **We then make a function for writing the messages to the client using the socket.send()**
8. **We then implement hamming code logic**

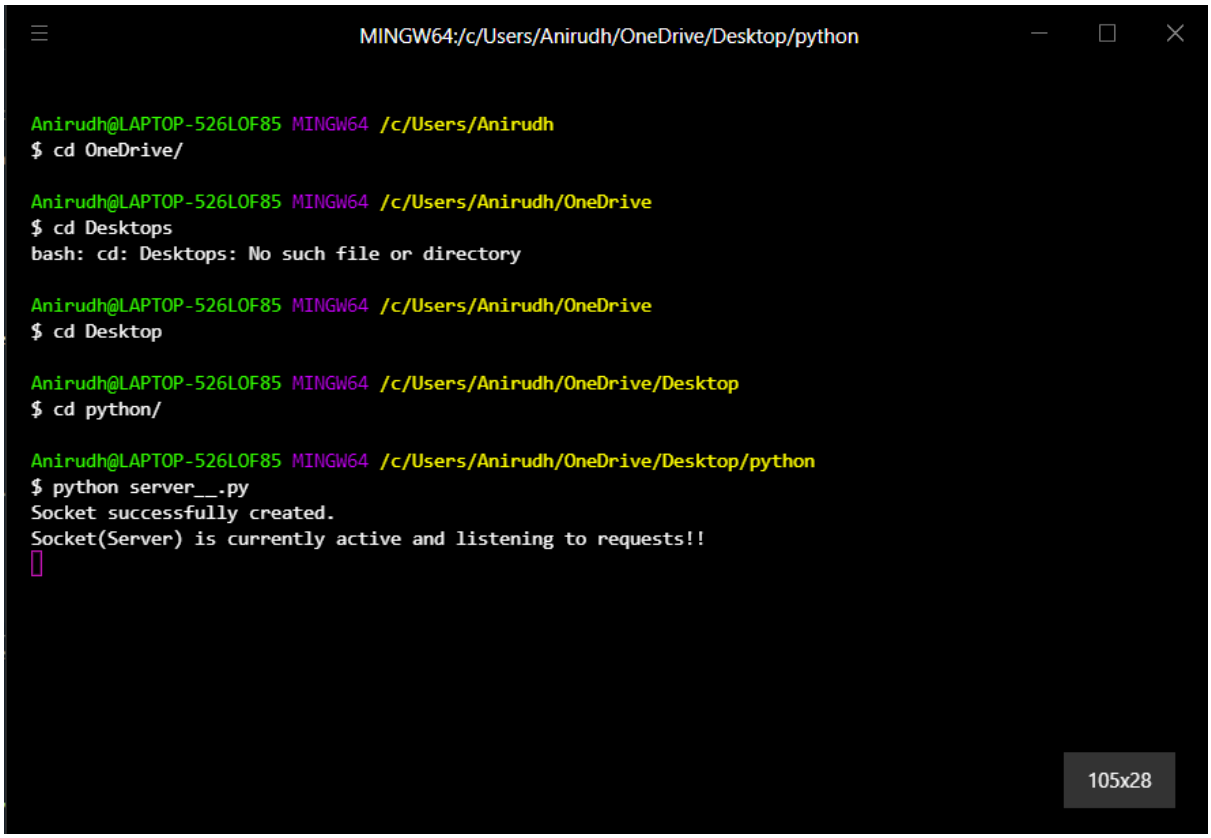
A code snippet for client.py:



```python
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# When no message recieved or any other communication error
import errno
import sys
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000

# Defining the HeaderSize of each message to be recieved
HEADERSIZE = 10

# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# No blocking the incoming messages
clientSocket.setblocking(False)

# Sending the username to the server
userName = my_userName.encode()
userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()
clientSocket.send(userNameHeader + userName)
```

In order to run our Application, we follow the following steps:

- ➔ **Open the Hyper terminal or Command Prompt**
- ➔ **Navigate onto your working file in our case server.py and client.py**
- ➔ **Write python filename to run a particular fine make sure python is installed beforehand.**
- ➔ **Now you can freely use the Chat Application**



CODE:

Sender.py

 # Importing the socket module

```
import socket
```

 # For realtime updation of state

```
import threading
import time
```

```
# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("ANIRUDH VADERA 20BCE2940")
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}
```

```
# Functions to implement Hamming Code:
def calcRedundantBits(m):
    r = 0
    while((2**r)<(m + r + 1)):
        r = r + 1
    return r;


def posRedundantBits(data, r):

    # Redundancy bits are placed at the positions
    # which correspond to the power of 2.
    j = 0
    k = 1
    m = len(data)
    res = ''

    # If position is power of 2 then insert '0'
    # Else append the data
    for i in range(1, m + r+1):
        if(i == 2**j):
            res = res + '0'
            j += 1
        else:
            res = res + data[-1 * k]
            k += 1
```

```
    # The result is reversed since positions are
    # counted backwards. (m + r+1 ... 1)
    return res[::-1]


def calcParityBits(arr, r):
    n = len(arr)

    # For finding rth parity bit, iterate over
    # 0 to r - 1
    for i in range(r):
        val = 0
        for j in range(1, n + 1):

            # If position has 1 in ith significant
            # position then Bitwise OR the array value
            # to find parity bit value.
            if(j!=(2**i)):
                if(j & (2**i) == (2**i)):
                    val = val ^ int(arr[-1 * j])
                    # -1 * j is given since array is reversed

        # String Concatenation
        # (0 to n - 2^r) + parity bit + (n - 2^r + 1 to n)
        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr


# A function to recieve messages from the clients connected over the network
```

```
def recieveMessage(clientSocket):
    try:
        return {"Data" : clientSocket.recv(128)}
    except:
        return False


flag = 1
check = 0


# Making a thread for every user connected to the server
def clientThread(notifiedSocket):
    global flag,check
    while True:
        if(check==0):
            data = input("Enter the dataword to be coded : ")


            # Calculate the no of Redundant Bits Required
            m = len(data)
            r = calcRedundantBits(m)


            print("The redundancy bits required are : ",r)


            # Determine the positions of Redundant Bits
            arr = posRedundantBits(data, r)


            # Determine the parity bits
            arr = calcParityBits(arr, r)
```

```
        print("Sending number of check bits : ")
        notifiedSocket.send(str(r).encode())

        print("The data after generation of r bits is : ",arr)
        print("Sending the coded data to reciever : ")
        notifiedSocket.send(arr.encode())
        check = 1


def recieve(notifiedSocket):
    global flag,check
    while True:
        # Checking if reciever wants more data
        flag = clientSocket.recv(1).decode().strip()
        flag = int(flag)
        if(int(flag)==0):
            print("Closing the sender side : ")
        if(flag == 0):
            check = 1
        else:
            check = 0


# Listening to requests infinitely untill interupted
while (flag!=0):
    # Accepting the user and storing its address in the below defined variables
    clientSocket, clientAddress = serverSocket.accept()
```

```
    # Getting the information user wants to send
    user = recieveMessage(clientSocket)
    if user is False:
        continue
    socketsList.append(clientSocket)
    clients[clientSocket] = user
    print(f"Connection from {clientAddress} has been established!! : UserName :
{user['Data'].decode()}")
    msg = "Welcome to the server,Thanks for connecting!!"
    # Sending information to client socket
    clientSocket.send(msg.encode())

    thread = threading.Thread(target = clientThread, args = (clientSocket,))
    thread.start()

    thread2= threading.Thread(target = recieve, args = (clientSocket,))
    thread2.start()
```

Reciever.py

```
# Importing the socket module
import socket

# For realtime updation of state
import threading

import random

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
```

```
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000


# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# Sending the username to the server
userName = my_userName.encode()
clientSocket.send(userName)

# Function to detect error in hamming code
def detectError(arr, nr):
    n = len(arr)
    res = 0

    # Calculate parity bits again
    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
```

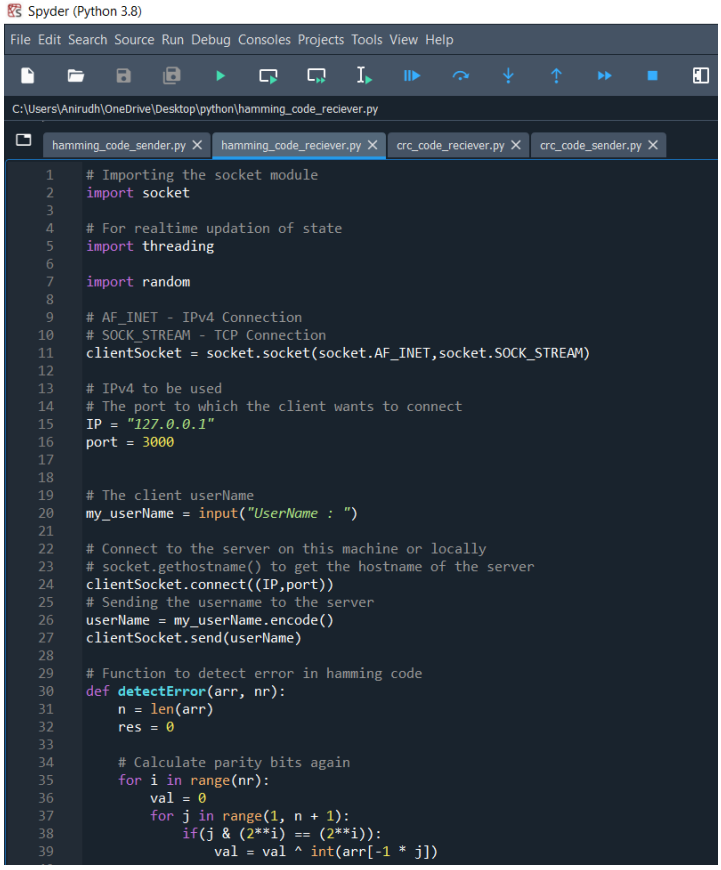
```
            val = val ^ int(arr[-1 * j])

        # Create a binary no by appending
        # parity bits together.

        res = res + val*(10**i)

    # Convert binary to decimal
    return int(str(res), 2)



# recieving chunks of data from the server
def recieveData():
    flag = 0
    yn = 1
    while True:
        # try:
            if(flag == 0):
                msg = clientSocket.recv(128).decode()
                print(f"Server > {msg}")
                flag = 1
            else:# For the subsequent messages
                r = int(clientSocket.recv(1).decode().strip())
                recievedData = clientSocket.recv(128).decode()
                x = random.randint(0,len(recievedData)-1)
                if(int(recievedData[x])==0):
                    recievedData = recievedData[0:x] + "1" + recievedData[(x +
1):len(recievedData)]
```

```
        else:
            recievedData = recievedData[0:x] + "0" + recievedData[(x +
1):len(recievedData)]
        print("The coded data recieved from Sender is : ",recievedData)
        print("Checking if the data is valid : ")
        correction = detectError(recievedData, r)
        if(correction==0):
            print("The recieved coeded data is correct : ")
            actual_data_word = ""
            j = 0
            for i in range(1,len(recievedData)+1):
                if(i==(2**j)):
                    j = j + 1
                else:
                    actual_data_word = actual_data_word + recievedData[-1*i]
            print("Hence the actual dataword is : ",actual_data_word[::-1])

            print("Do you wish to continue recieving data : ")
            print("1 : Yes")
            print("0 : No")
            yn = input()
            if(int(yn)==0):
                print("Closing the reciever side : ")
            clientSocket.send(yn.encode())
        else:
            print("The recieved coeded data is incorrect : ")
            print("The position of error is : " , len(recievedData) - correction + 1)
            print("If its a single bit error : ")
```

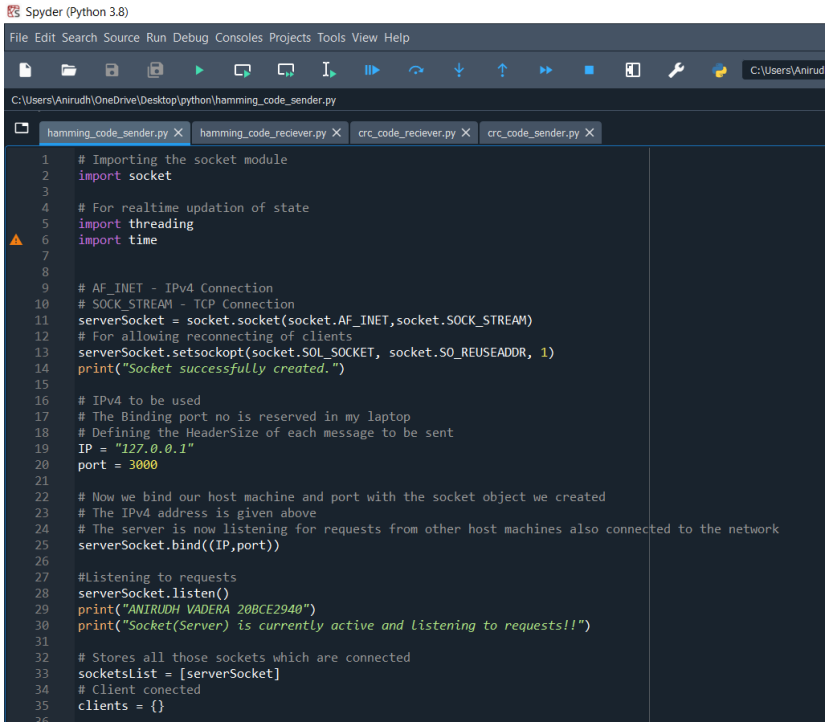
```
            actual_data_word = ""
            j = 0
            for i in range(1,len(recievedData)+1):
               if(i==(2**j)):
                  j = j + 1
               else:
                  if(i==correction):
                     if(int(recievedData[-1*i]) == 0):
                        actual_data_word = actual_data_word + "1"
                     else:
                        actual_data_word = actual_data_word + "0"
                  else:
                     actual_data_word = actual_data_word + recievedData[-1*i]
            print("Hence the actual dataword is : ",actual_data_word[::-1])

            print("Do you wish to continue recieving data : ")
            print("1 : Yes")
            print("0 : No")
            yn = input()
            if(int(yn)==0):
               print("Closing the reciever side : ")
            clientSocket.send(yn.encode())

recieveThread = threading.Thread(target = recieveData, args=())
recieveThread.start()
```

CODE SNIPPETS:

Receiver.py:



Sender.py:



OUTPUT:

Error Generation:

To stimulate a noisy channel, we are taking help of random library in python which on random changes a random bit in the data to be sent:

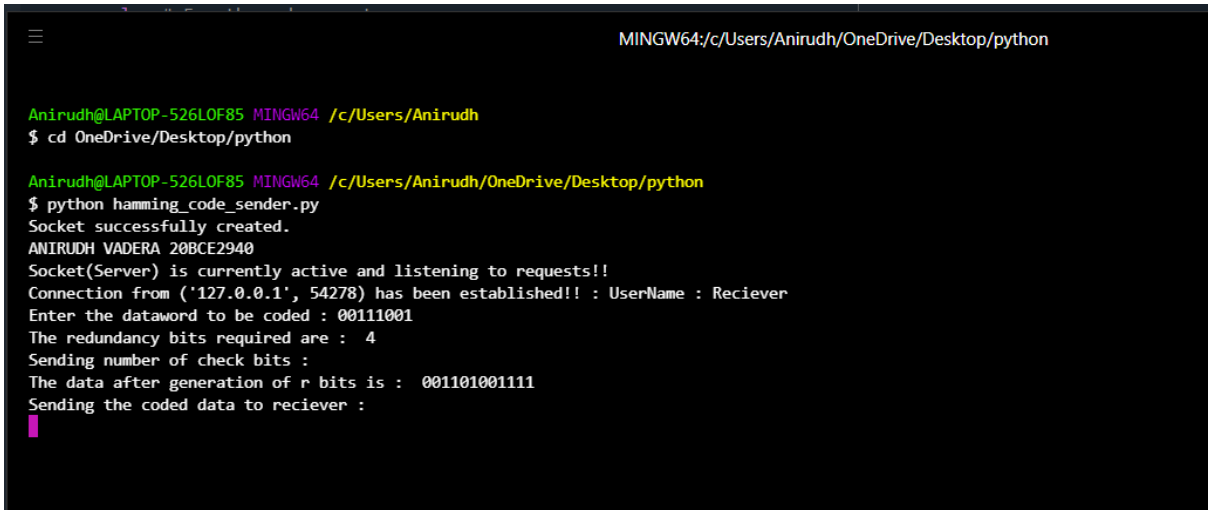
Code Related:

```
    x = random.randint(0,len(recievedData)-len(polynomial)-2)
    if(int(recievedData[x])==0):
        recievedData = recievedData[0:x] + "1" + recievedData[(x + 1):len(recievedData)]
    else:
        recievedData = recievedData[0:x] + "0" + recievedData[(x + 1):len(recievedData)]
    print("The coded data recieved from Sender is : ",recievedData)
```

Sending 8-bit data:

Ideal Case:

Sender.py:



Dataword: 00111001

The code sent is: 001101001111

The code received is: 001101001111

No Error

Conclusion(Reciever.py):

```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python/

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python hamming_code_reciever.py
UserName : Reciever
Server > Welcome to the server,Thanks for connecting!!
The coded data recieved from Sender is :  001101001111
Checking if the data is valid :
The recieved coeded data is correct :
Hence the actual dataword is :  00111001
Do you wish to continue recieving data :
1 : Yes
0 : No
```

If user wants to resend some new data:

Reciever.py:

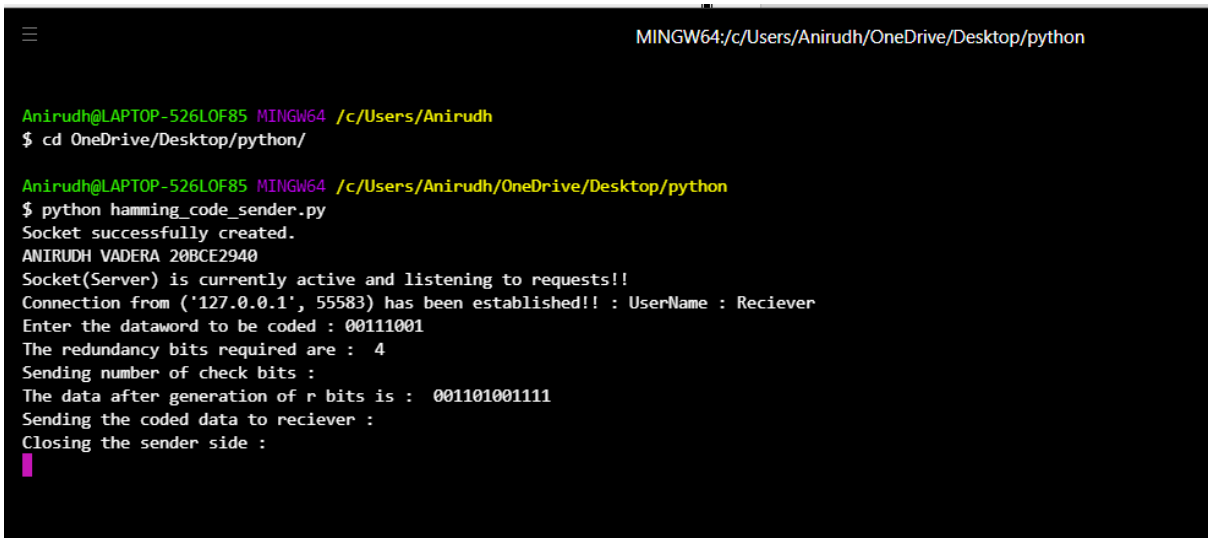
```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python/

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python hamming_code_reciever.py
UserName : Reciever
Server > Welcome to the server,Thanks for connecting!!
The coded data recieved from Sender is :  001101001111
Checking if the data is valid :
The recieved coeded data is correct :
Hence the actual dataword is :  00111001
Do you wish to continue recieving data :
1 : Yes
0 : No
1
The coded data recieved from Sender is :  111101110111
Checking if the data is valid :
The recieved coeded data is correct :
Hence the actual dataword is :  11111111
Do you wish to continue recieving data :
1 : Yes
0 : No
```

Sender.py:

```
MINGW64:/c/Users/Anirudh/OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh
$ cd OneDrive/Desktop/python

Anirudh@LAPTOP-526LOF85 MINGW64 /c/Users/Anirudh/OneDrive/Desktop/python
$ python hamming_code_sender.py
Socket successfully created.
ANIRUDH VADERA 20BCE2940
Socket(Server) is currently active and listening to requests!!
Connection from ('127.0.0.1', 54278) has been established!! : UserName : Reciever
Enter the dataword to be coded : 00111001
The redundancy bits required are :  4
Sending number of check bits :
The data after generation of r bits is :  001101001111
Sending the coded data to reciever :
Enter the dataword to be coded : 11111111
The redundancy bits required are :  4
Sending number of check bits :
The data after generation of r bits is :  111101110111
Sending the coded data to reciever :
```

(Code is damaged/corrupted) When the code is changed due to a noisy channel:

Sender.py:



Dataword: 00111001

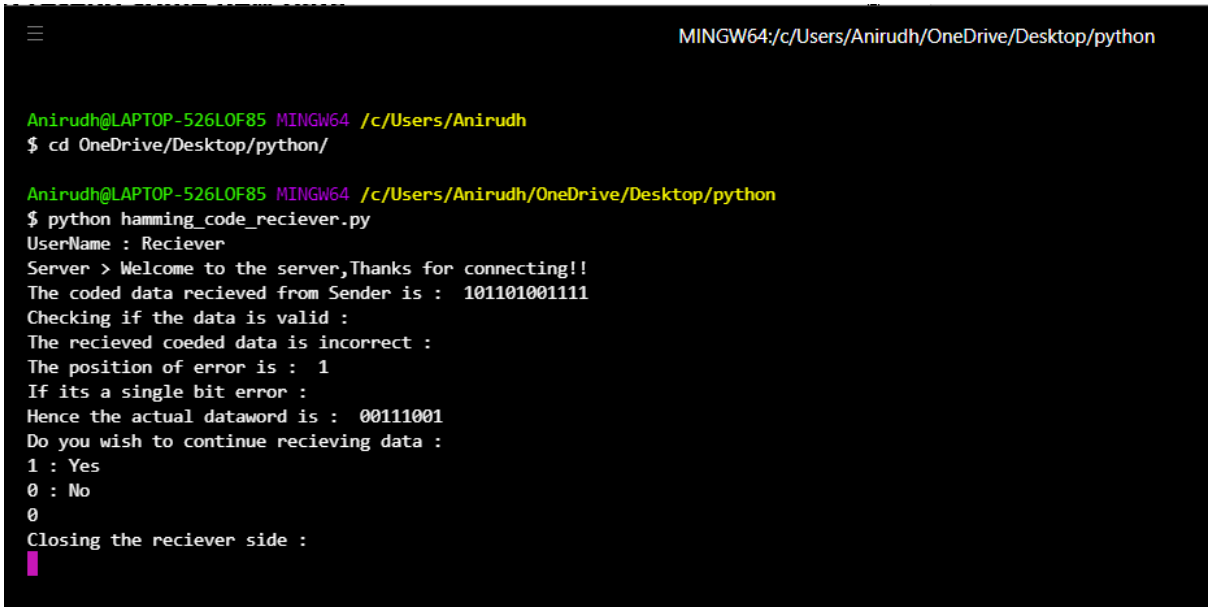
The code sent is: 001101001111

The code received is: 101101001111

Error at 1st bit

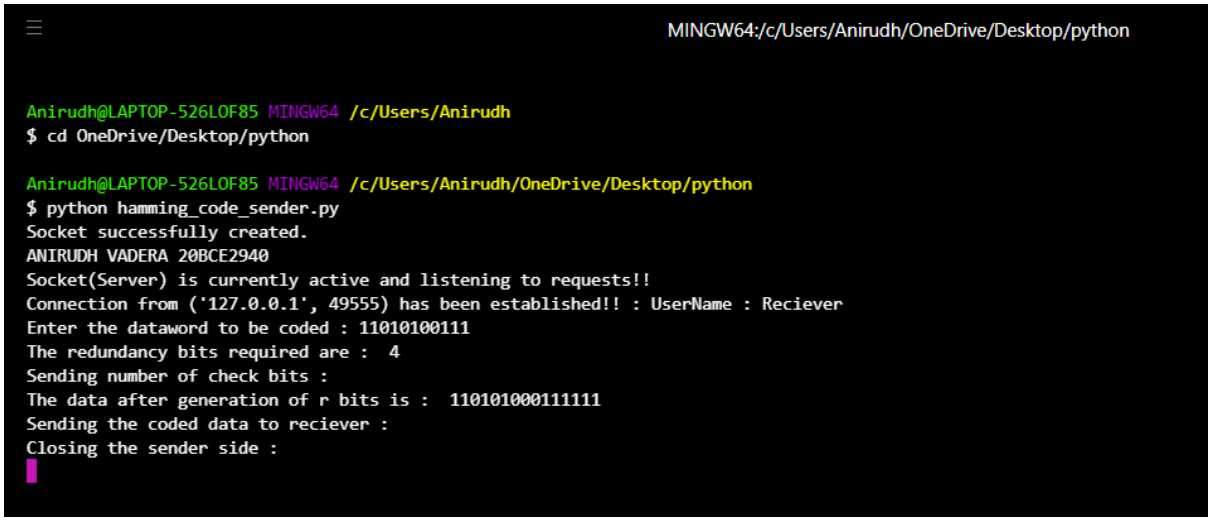
Conclusion(Reciever.py):

Corrected the error as well.



Sending 11-bit data:

Sender.py:



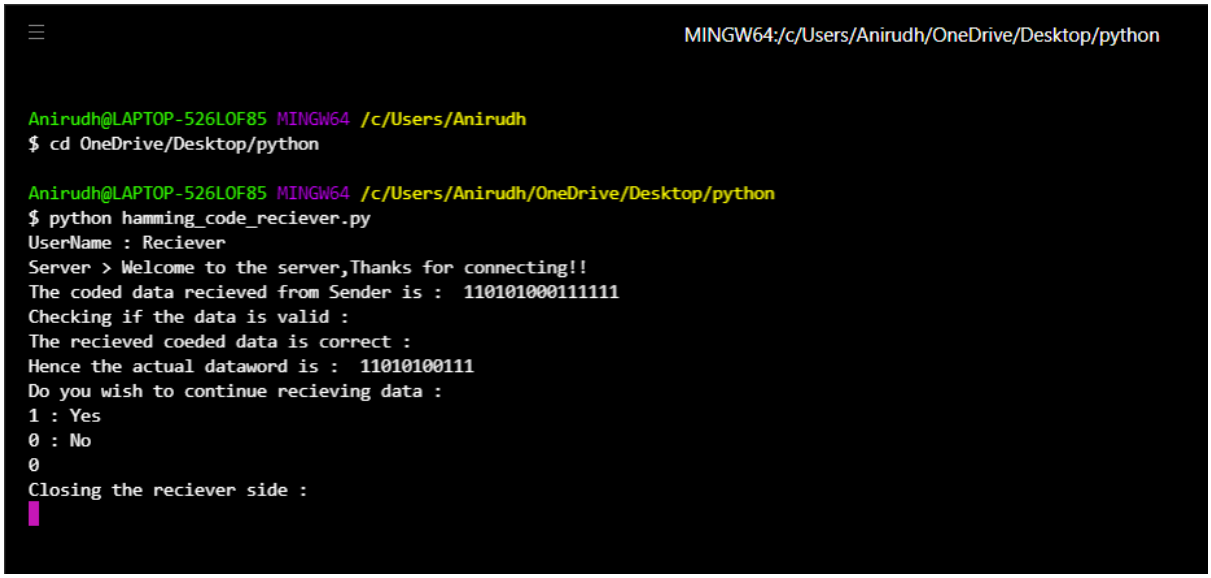
Dataword: 11010100111

The code sent is: 110101000111111

The code received is: 110101000111111

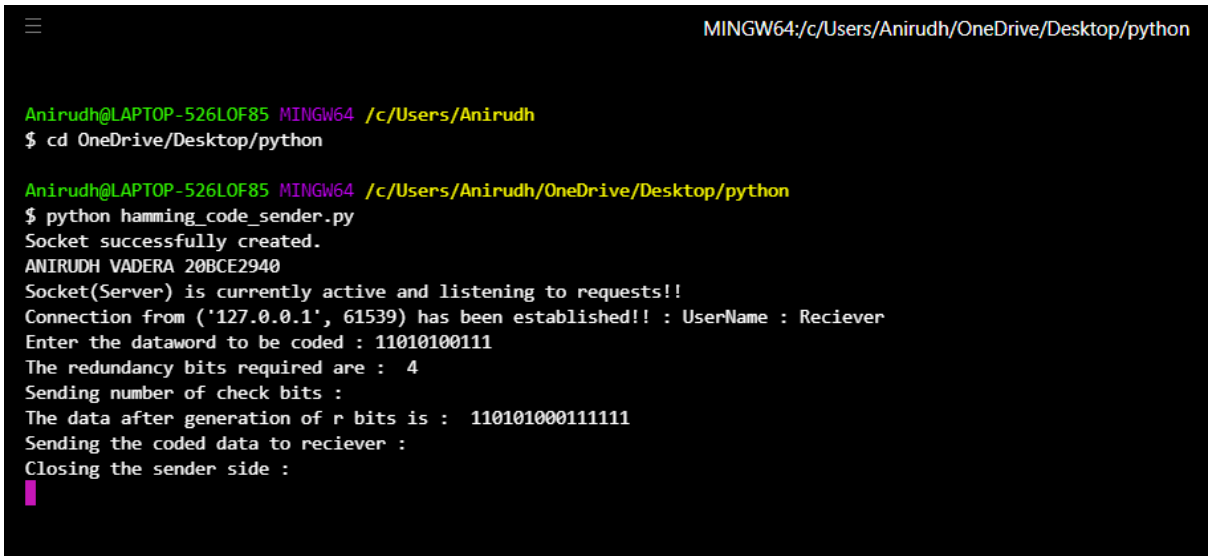
No Error

Conclusion(Reciever.py):



(Code is damaged/corrupted) When the code is changed due to a noisy channel:

Sender.py:



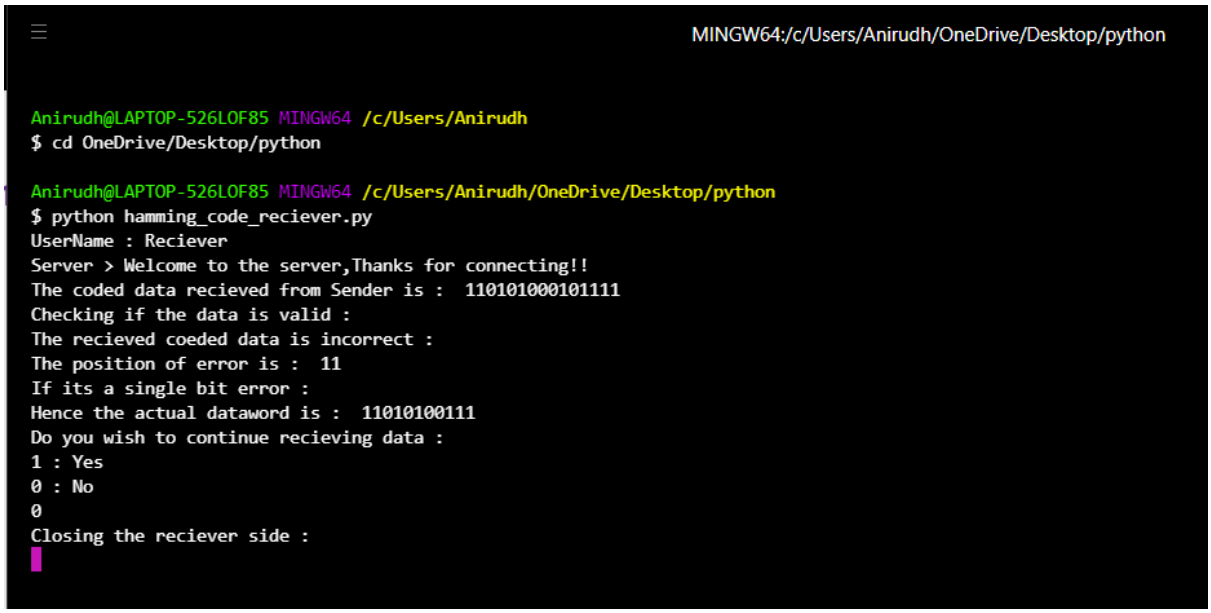
Dataword: 11010100111

The code sent is: 110101000111111

The code received is: 110101000101111

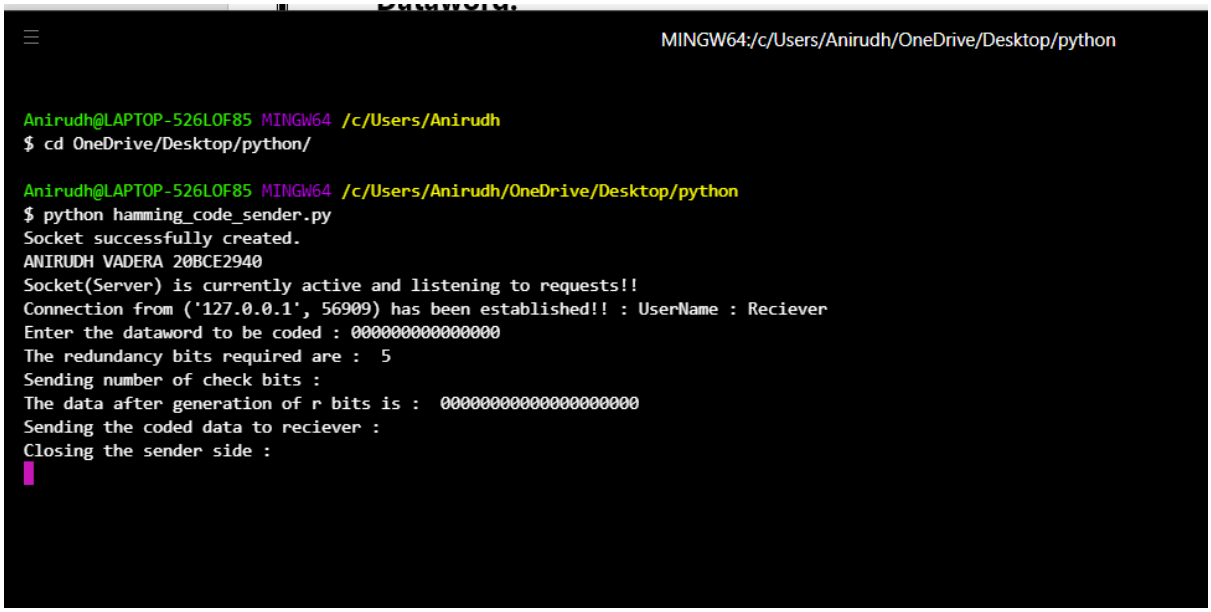
Error in 11th bit

Conclusion(Reciever.py):



Sending 15-bit data:

Sender.py:



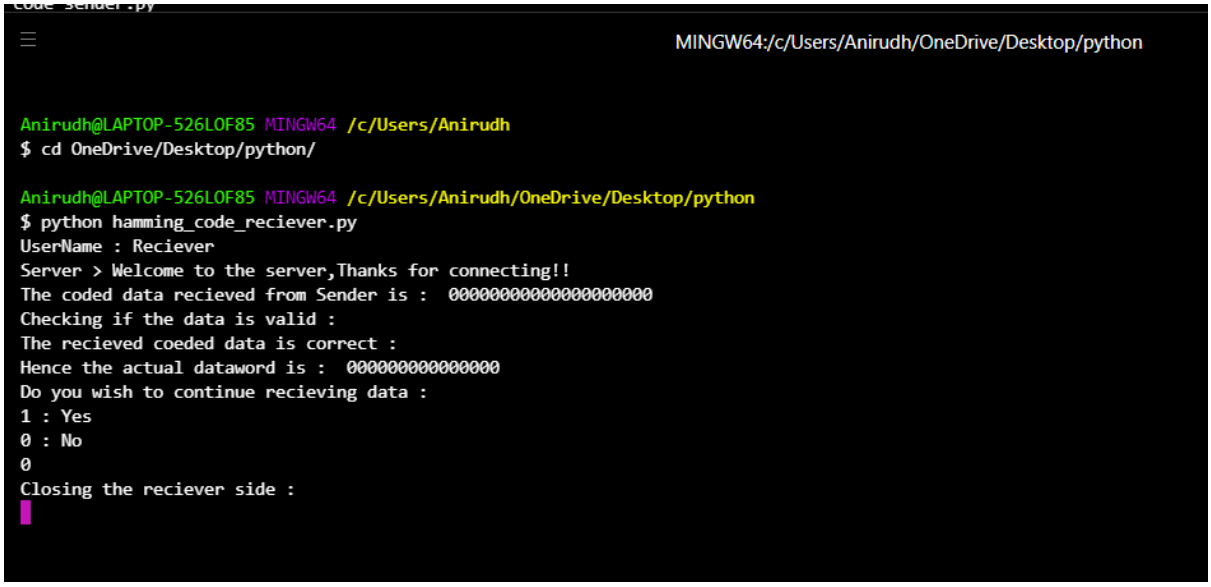
Dataword: 000000000000000

The code sent is: 0000000000000000000

The code received is: 0000000000000000000

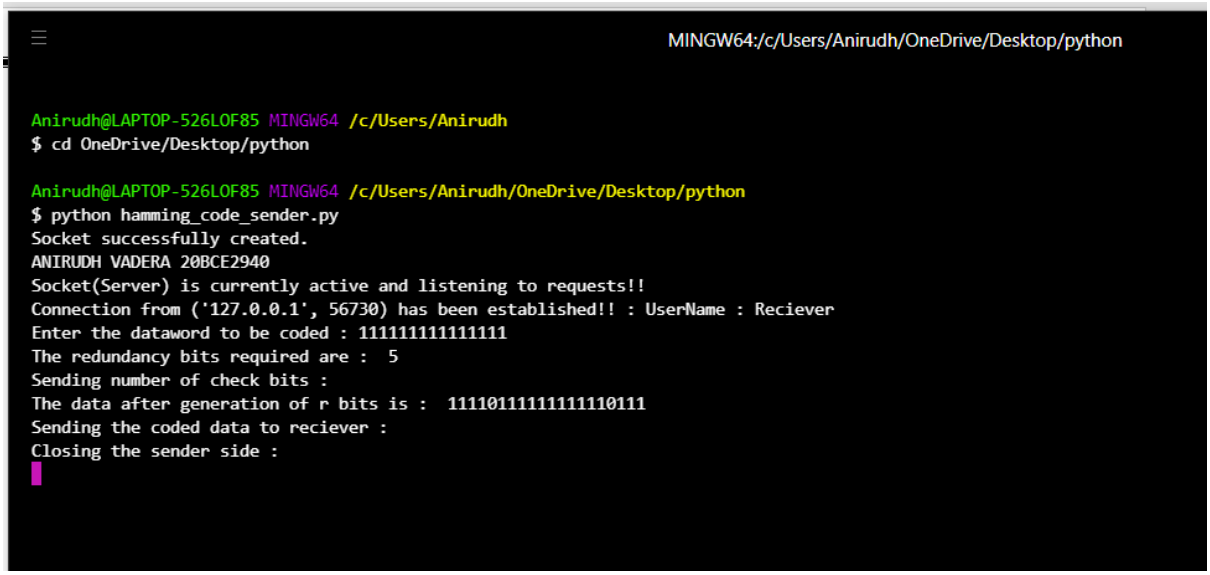
No Error

Conclusion(Reciever.py):



(Code is damaged/corrupted) When the code is changed due to a noisy channel:

Sender.py:



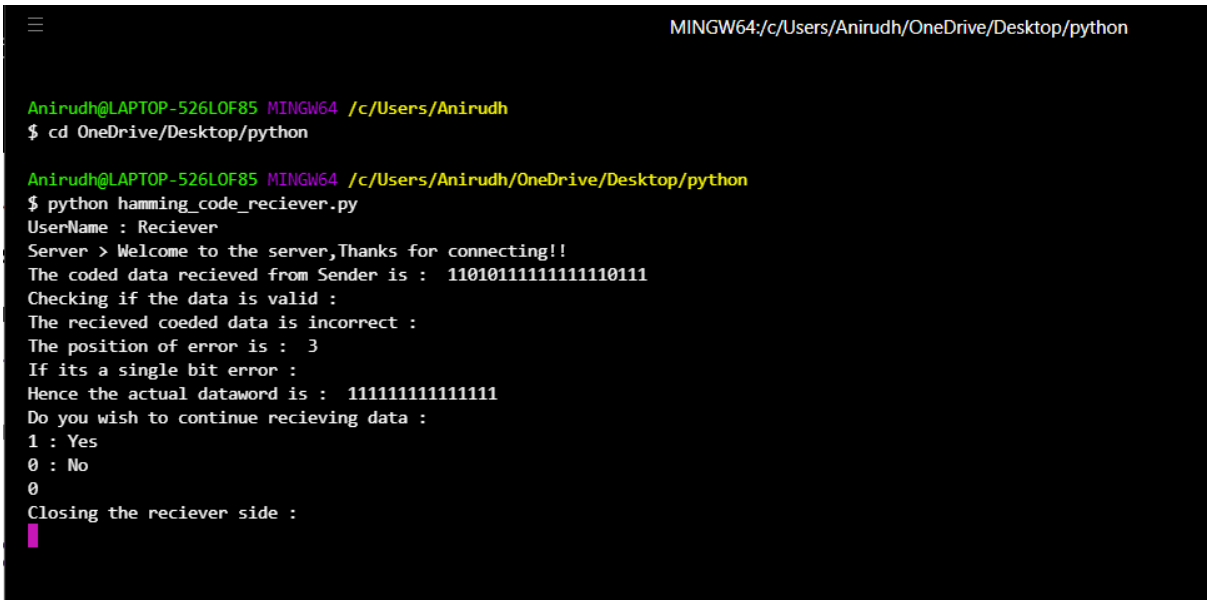
Dataword: 111111111111111

The code sent is: 11110111111111110111

The code received is: 11010111111111110111

Error in 3rd bit

Conclusion(Reciever.py):



CRC CODE:

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.
CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.
n : Number of bits in data to be sent

from sender side.

k : Number of bits in the key obtained

from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

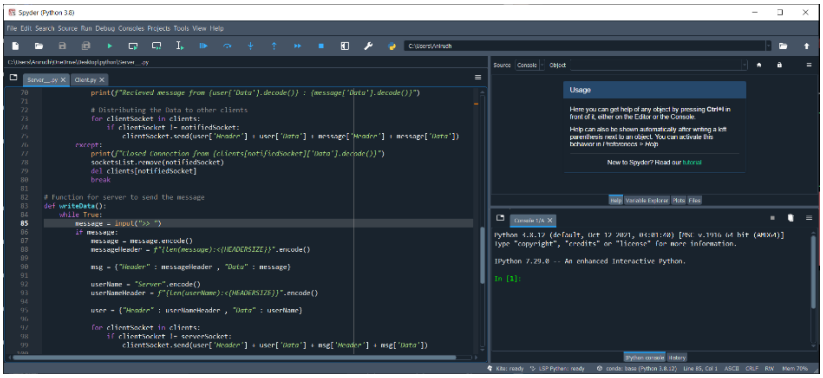
Receiver Side (Check if there are errors introduced in transmission)
Perform modulo-2 division again and if the remainder is 0, then there are no errors.
In this article we will focus only on finding the remainder i.e. check word and the code word.

Modulo 2 Division:
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

PROCEDURE:

➔ **First Open your python ide**
➔ **I will be using anaconda distribution and a spyder IDE**



➔ **We will be using 2 files for our purpose**
➔ **A server file**
➔ **A client file**

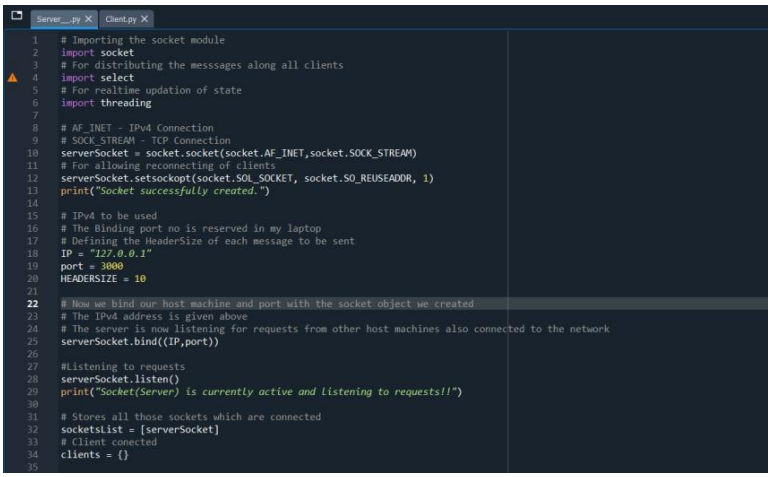
There are some common steps to be followed explained below

➔ **A detailed explanation along with the code is given further below**

Server.py: (Reciever)

8. **Import the necessary files.**
9. **Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)**
10. **Bind the server using socket.bind(IP,port) method providing the IP and the port.**
11. **We now define the socketsList which stores all the sockets currently in action and make a client Dictionary which stores information about the clients.**
12. **We then define a function for reading messages using socket.recv() method**
13. **We then make a function for writing the messages to the client using the socket.send()**
14. **We then implement the crc code logic**

A code snippet for server.py:



```
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
serverSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# For allowing reconnecting of clients
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000
HEADERSIZE = 10

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}
```

Client.py: (Sender)

9. **Import the necessary files.**
10. **Using a IPv4 connection and a TCP connection initiate the server side socket using socket.socket(socket.AF_INET,socket.SOCK_STREAM)**
11. **Connect to the server using socket.connect(IP) function by providing the appropriate IP address**
12. **Select a username and send it to the server.**
13. **We then define a function for reading messages using socket.recv() method**
14. **We try to catch as many errors as possible in it.**
15. **We then make a function for writing the messages to the client using the socket.send()**
16. **We then implement crc code logic**

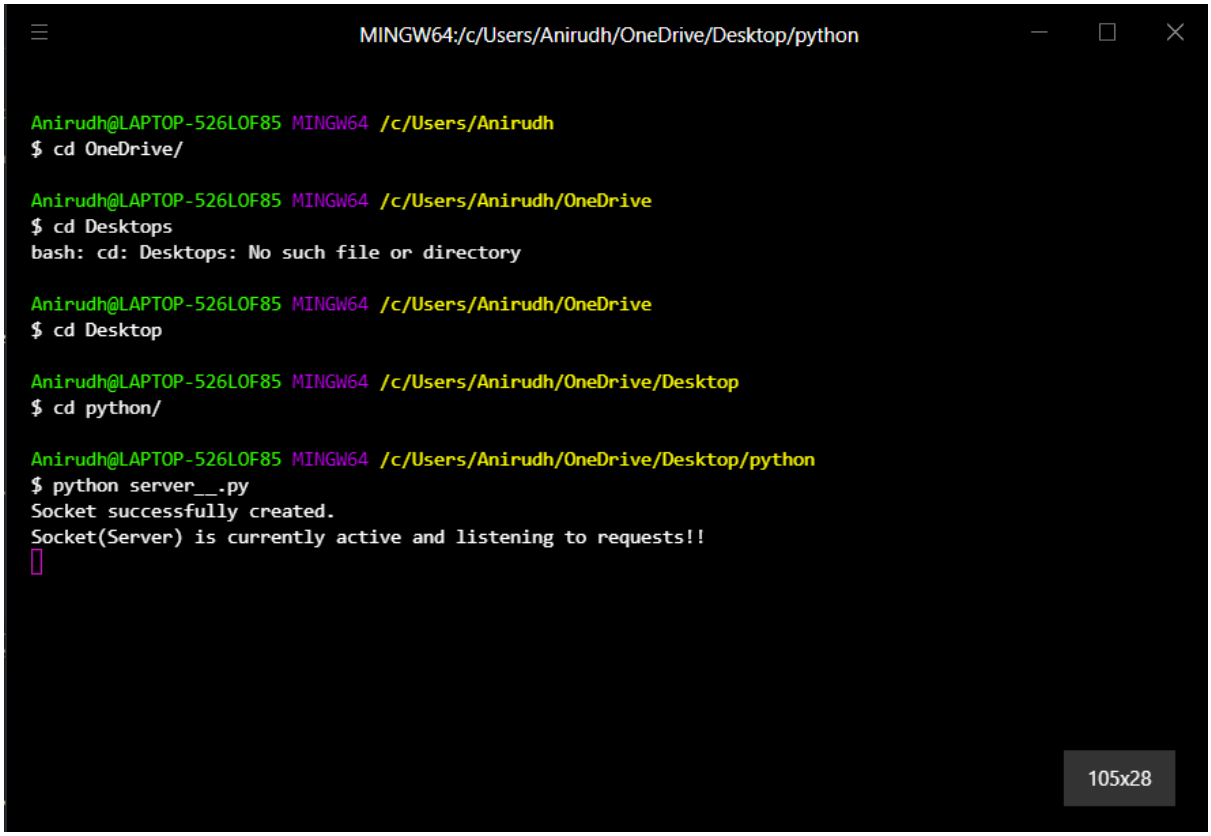
A code snippet for client.py:

Server__.py | Client.py

```
# Importing the socket module
import socket
# For distributing the messsages along all clients
import select
# When no message recieved or any other communication error
import errno
import sys
# For realtime updation of state
import threading

# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
port = 3000

# Defining the HeaderSize of each message to be recieved
HEADERSIZE = 10

# The client userName
my_userName = input("UserName : ")

# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# No blocking the incoming messages
clientSocket.setblocking(False)

# Sending the username to the server
userName = my_userName.encode()
userNameHeader = f"{len(userName):<{HEADERSIZE}}".encode()
clientSocket.send(userNameHeader + userName)

```

In order to run our Application, we follow the following steps:

- ➔ **Open the Hyper terminal or Command Prompt**
- ➔ **Navigate onto your working file in our case server.py and client.py**
- ➔ **Write python filename to run a particular fine make sure python is installed beforehand.**
- ➔ **Now you can freely use the Chat Application**



CODE:

Sender.py:

 # Importing the socket module

import socket

 # For realtime updation of state

import threading

import time

 # AF_INET - IPv4 Connection

 # SOCK_STREAM - TCP Connection

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

 # For allowing reconnecting of clients

```
serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket successfully created.")

# IPv4 to be used
# The Binding port no is reserved in my laptop
# Defining the HeaderSize of each message to be sent
IP = "127.0.0.1"
port = 3000

# Now we bind our host machine and port with the socket object we created
# The IPv4 address is given above
# The server is now listening for requests from other host machines also connected to the network
serverSocket.bind((IP,port))

#Listening to requests
serverSocket.listen()
print("ANIRUDH VADERA 20BCE2940")
print("Socket(Server) is currently active and listening to requests!!")

# Stores all those sockets which are connected
socketsList = [serverSocket]
# Client conected
clients = {}

# Function for binary 2 division
def binary_division(divident,polynomial):
    global code
```

```
    if(len(divident)==len(polynomial)-1):
        code=divident
        return
    temp=divident[0:len(polynomial)]
    if(temp[0]=="0"):
        binary_division(divident[1:],polynomial)
    else:
        temp2=""
        for i in range(len(polynomial)):
            if(temp[i]=="1" and polynomial[i]=="1"):
                temp2+="0"
            if(temp[i]=="1" and polynomial[i]=="0"):
                temp2+="1"
            if(temp[i]=="0" and polynomial[i]=="1"):
                temp2+="1"
            if(temp[i]=="0" and polynomial[i]=="0"):
                temp2+="0"
        divident=temp2+divident[len(polynomial):]
        binary_division(divident,polynomial)
```

 # A function to recieve messages from the clients connected over the network

```
def recieveMessage(clientSocket):
    try:
        return {"Data" : clientSocket.recv(128)}
    except:
        return False

flag = 1
```

```
check = 0
code=''

# Making a thread for every user connected to the server
def clientThread(notifiedSocket):
   global flag,check,code
   while True:
      if(check==0):

         polynomial=input("Enter the polynomial : ")

         data = input("Enter the dataword to be coded : ")
         ss=''
         for i in range(len(polynomial)-1):
            ss+="0"
         divident=data+ss
         binary_division(divident,polynomial)
         if(len(code)<len(polynomial)-1):
            ss=''
            for i in range(len(polynomial)-1-len(code)):
               ss+="0"
            code=ss+code
         data=list(data)
         data="".join(data)

         print("Sending  Polynomial : ")
         notifiedSocket.send(str(len(polynomial)).encode())
         notifiedSocket.send(polynomial.encode())
```

```
            print("The data after generation of code bits is : ",(data+code))
            print("Sending the coded data to reciever : ")
            notifiedSocket.send((data+code).encode())
            check = 1


def recieve(notifiedSocket):
    global flag,check
    while True:
        # Checking if reciever wants more data
        flag = clientSocket.recv(1).decode().strip()
        flag = int(flag)
        if(int(flag)==0):
            print("Closing the sender side : ")
        if(flag == 0):
            check = 1
        else:
            check = 0
```

 # Listening to requests infinitely untill interupted

```
while (flag!=0):
    # Accepting the user and storing its address in the below defined variables
    clientSocket, clientAddress = serverSocket.accept()
    # Getting the information user wants to send
    user = recieveMessage(clientSocket)
```

```
    if user is False:
        continue
    socketsList.append(clientSocket)
    clients[clientSocket] = user
    print(f"Connection from {clientAddress} has been established!! : UserName : {user['Data'].decode()}")
    msg = "Welcome to the server,Thanks for connecting!!"
    # Sending information to client socket
    clientSocket.send(msg.encode())

    thread = threading.Thread(target = clientThread, args = (clientSocket,))
    thread.start()

    thread2= threading.Thread(target = recieve, args = (clientSocket,))
    thread2.start()
```

Reciever.py:

```
# Importing the socket module
import socket
# For realtime updation of state
import threading
import random
# AF_INET - IPv4 Connection
# SOCK_STREAM - TCP Connection
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# IPv4 to be used
# The port to which the client wants to connect
IP = "127.0.0.1"
```

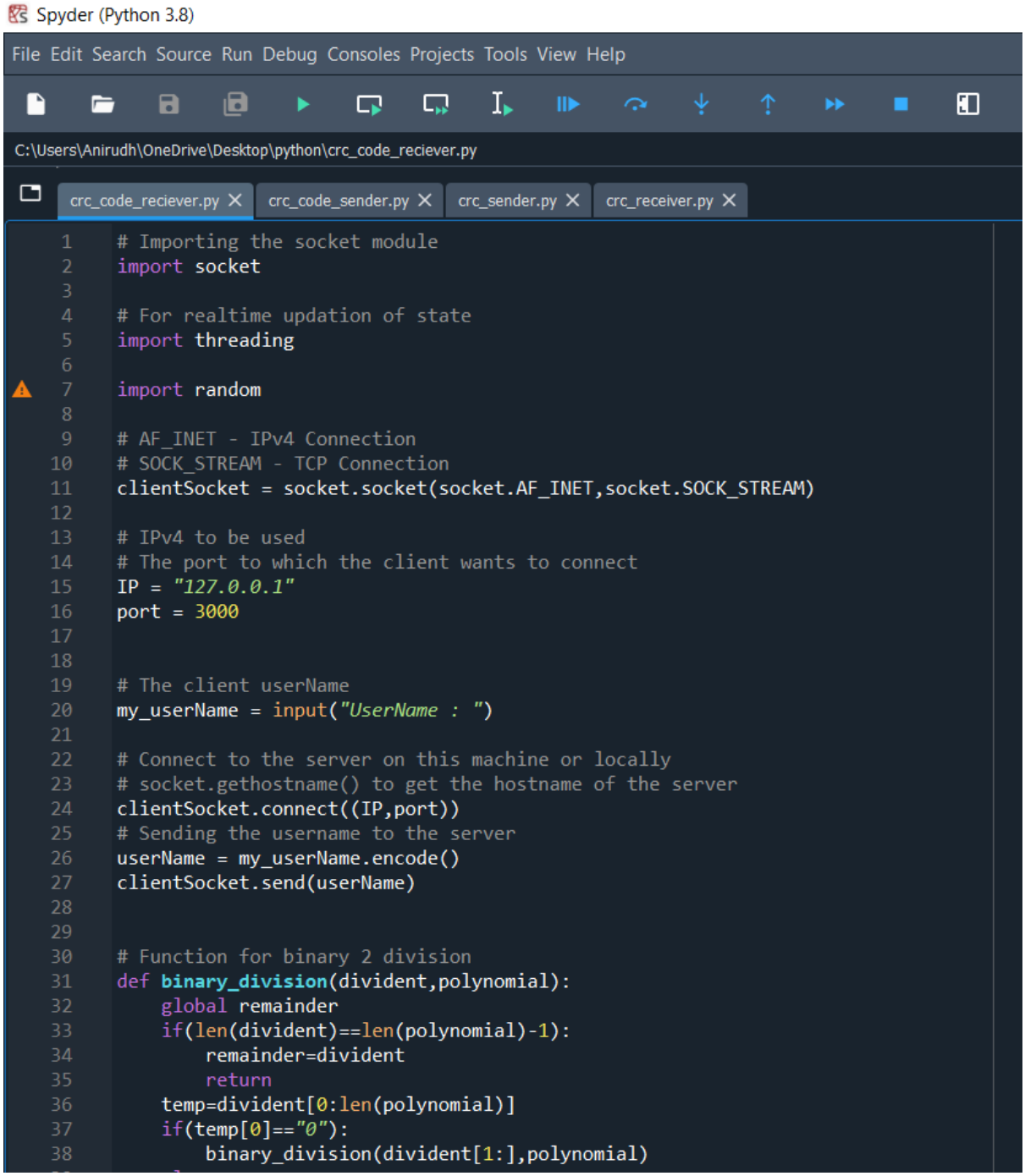
```
port = 3000


# The client userName
my_userName = input("UserName : ")


# Connect to the server on this machine or locally
# socket.gethostname() to get the hostname of the server
clientSocket.connect((IP,port))
# Sending the username to the server
userName = my_userName.encode()
clientSocket.send(userName)
# Function for binary 2 division
def binary_division(divident,polynomial):
    global remainder
    if(len(divident)==len(polynomial)-1):
        remainder=divident
        return
    temp=divident[0:len(polynomial)]
    if(temp[0]=="0"):
        binary_division(divident[1:],polynomial)
    else:
        temp2=""
        for i in range(len(polynomial)):
            if(temp[i]=="1" and polynomial[i]=="1"):
                temp2+="0"
            if(temp[i]=="1" and polynomial[i]=="0"):
                temp2+="1"
            if(temp[i]=="0" and polynomial[i]=="1"):
```

```
            temp2+="1"
        if(temp[i]=="0" and polynomial[i]=="0"):
            temp2+="0"
    divident=temp2+divident[len(polynomial):]
    binary_division(divident,polynomial)
remainder = ""
# recieving chunks of data from the server
def recieveData():
  flag = 0
  yn = 1
  global remainder
  while True:
    # try:
      if(flag == 0):
        msg = clientSocket.recv(128).decode()
        print(f"Server > {msg}")
        flag = 1
      else:# For the subsequent messages
        len_polynomial = int(clientSocket.recv(1).decode().strip())
        polynomial = clientSocket.recv(len_polynomial).decode()
        recievedData = clientSocket.recv(128).decode()
        # x = random.randint(0,len(recievedData)-len(polynomial)-2)
        # if(int(recievedData[x])==0):
        #     recievedData = recievedData[0:x] + "1" + recievedData[(x +
1):len(recievedData)]
        # else:
        #     recievedData = recievedData[0:x] + "0" + recievedData[(x +
1):len(recievedData)]
```

```
            print("The coded data recieved from Sender is : ",recievedData)
            print("Checking if the data is valid : ")
            binary_division(recievedData,polynomial)
            if(int(remainder)==0):
               print("The recieved coeded data is correct : ")
               print("Hence the actual dataword is : ",recievedData[:-
1*(len(polynomial)-1)])

               print("Do you wish to continue recieving data : ")
               print("1 : Yes")
               print("0 : No")
               yn = input()
               if(int(yn)==0):
                  print("Closing the reciever side : ")
               clientSocket.send(yn.encode())
            else:
               print("The recieved coeded data is incorrect : ")
               print("The Remainder is : " , remainder)

               print("Do you wish to continue recieving data : ")
               print("1 : Yes")
               print("0 : No")
               yn = input()
               if(int(yn)==0):
                  print("Closing the reciever side : ")
               clientSocket.send(yn.encode())
recieveThread = threading.Thread(target = recieveData, args=())
recieveThread.start()
```

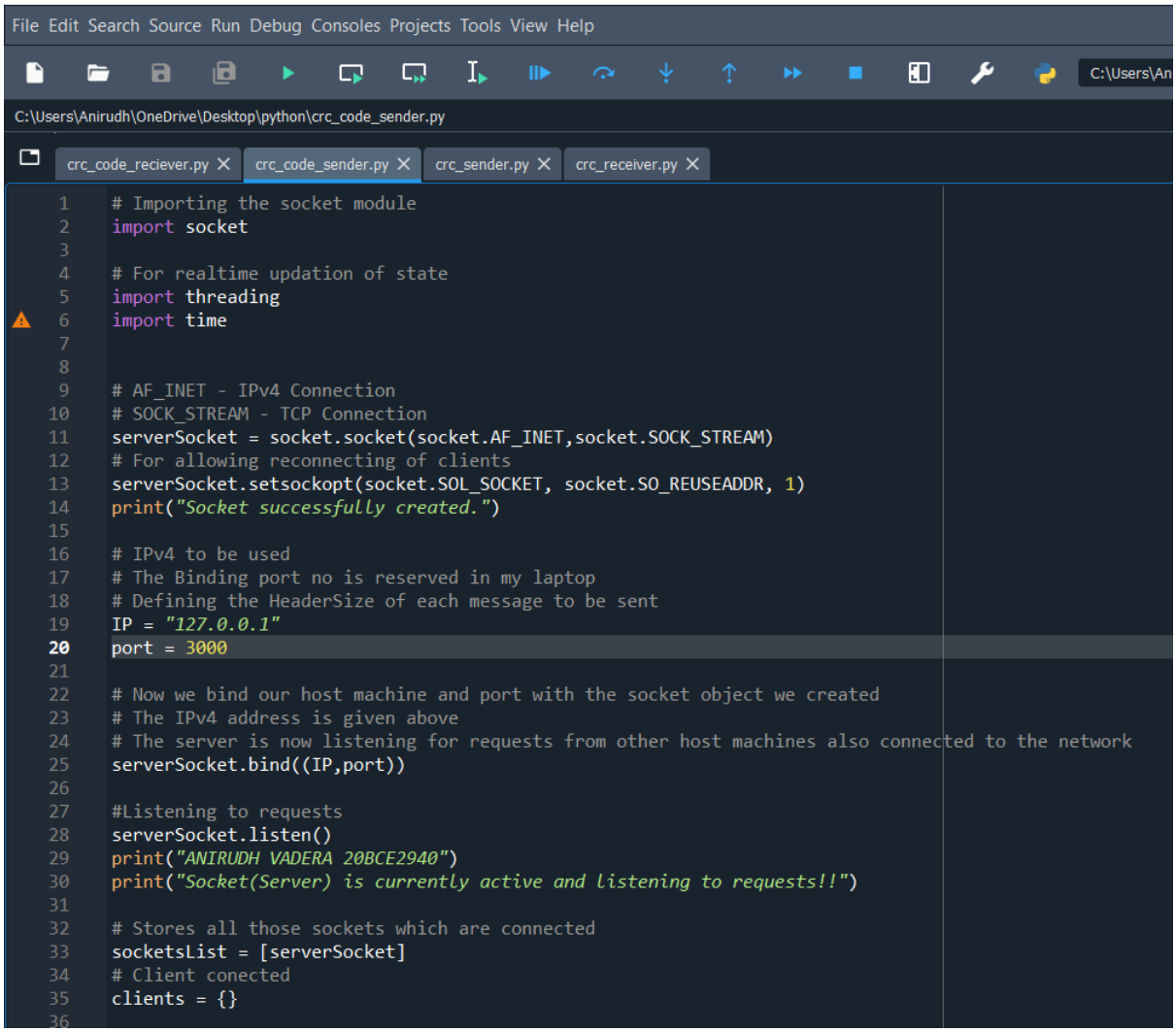
CODE SNIPPETS:

Receiver.py:



Sender.py:

Spyder (Python 3.8)



OUTPUT:

Error Generation:

To stimulate a noisy channel, we are taking help of random library in python which on random changes a random bit in the data to be sent:

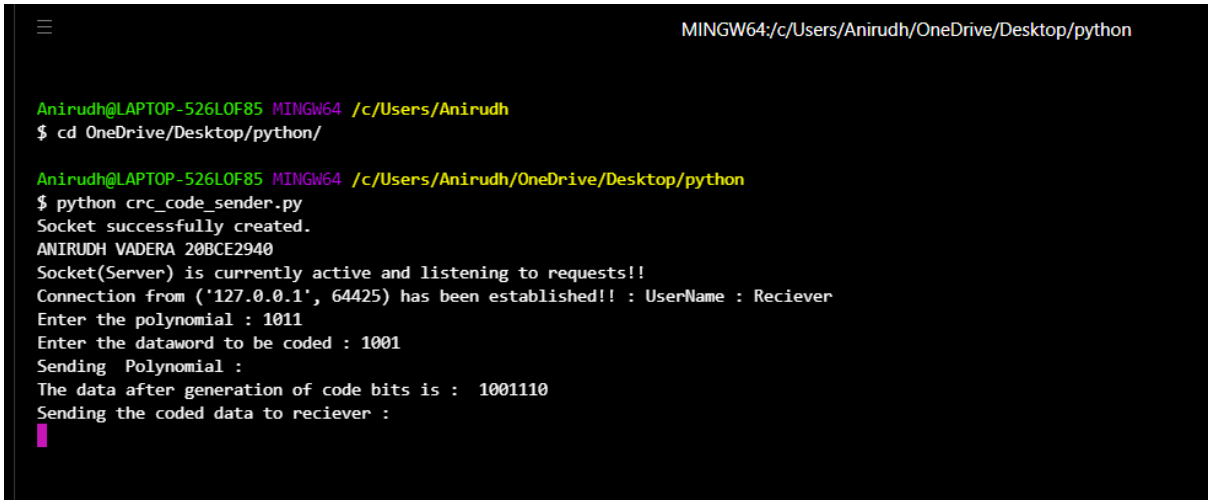
Code Related:

```
x = random.randint(0,len(recievedData)-len(polynomial)-2)
if(int(recievedData[x])==0):
    recievedData = recievedData[0:x] + "1" + recievedData[(x + 1):len(recievedData)]
else:
    recievedData = recievedData[0:x] + "0" + recievedData[(x + 1):len(recievedData)]
print("The coded data recieved from Sender is : ",recievedData)
```

Sending 4-bit dataword and 4-bit divisor:

Ideal Case:

Sender.py:



Dataword: 1001

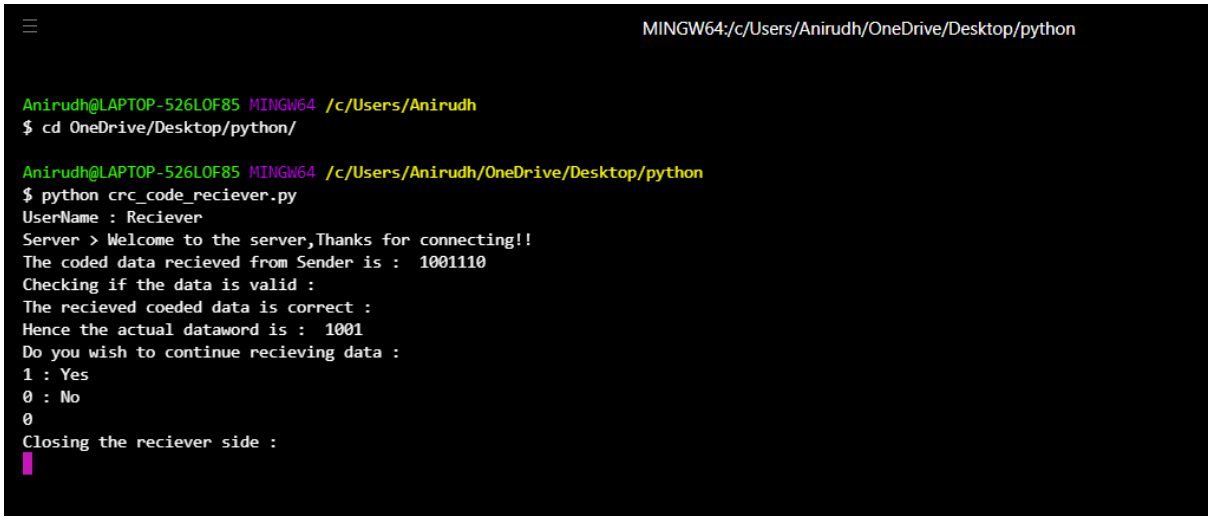
Polynomial(divisor): 1011

The code(dataword+remainder) sent is: 1001110

The code received is(at recievers side): 1001110

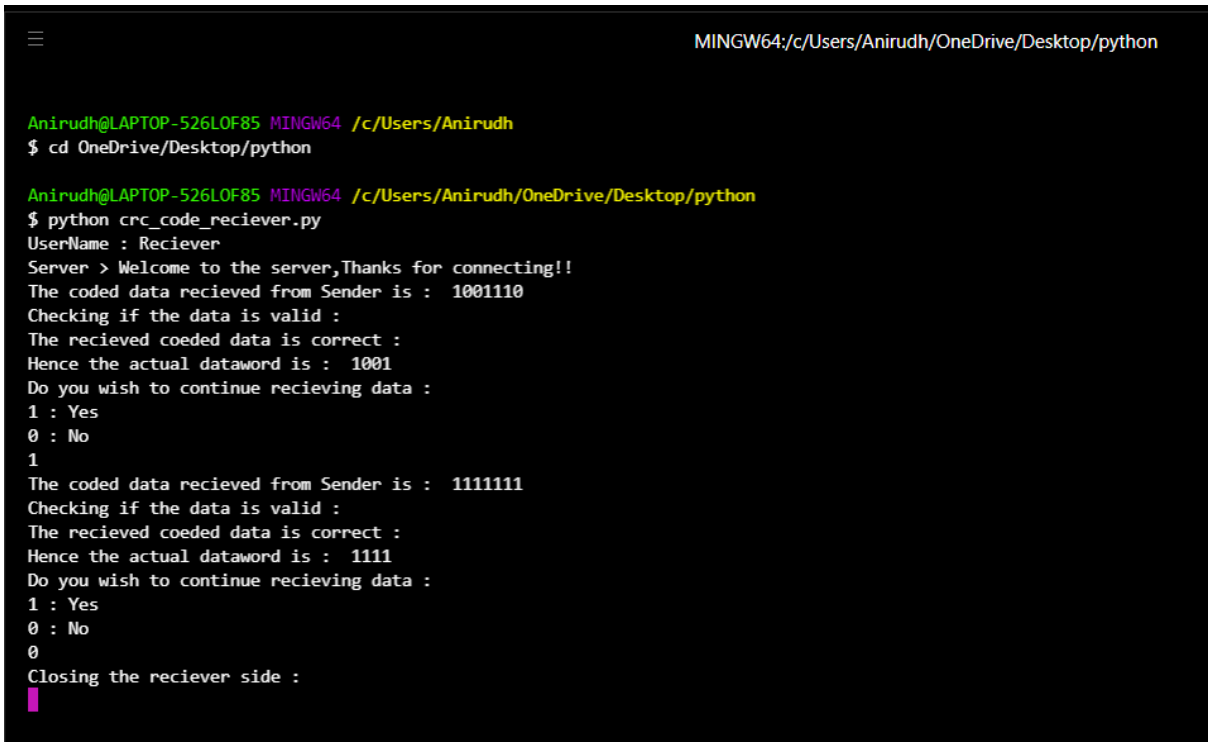
No Error

Conclusion(Reciever.py):

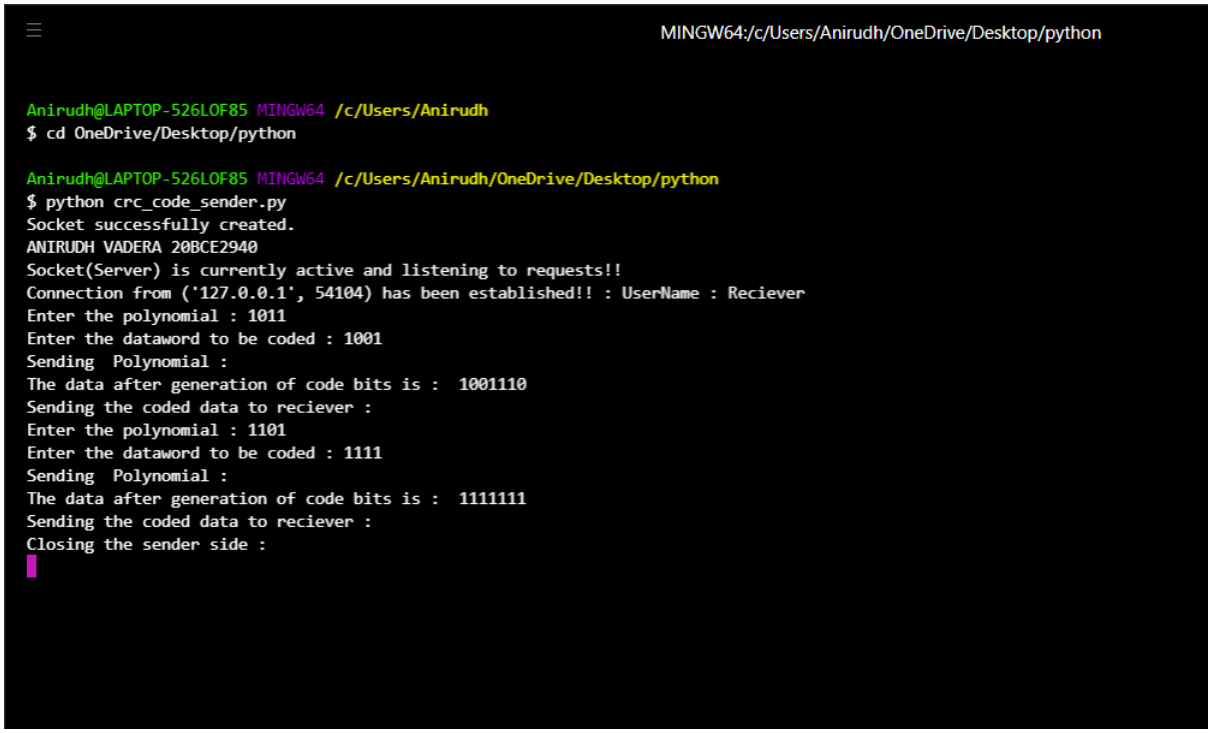


If user wants to resend some new data:

Reciever.py:

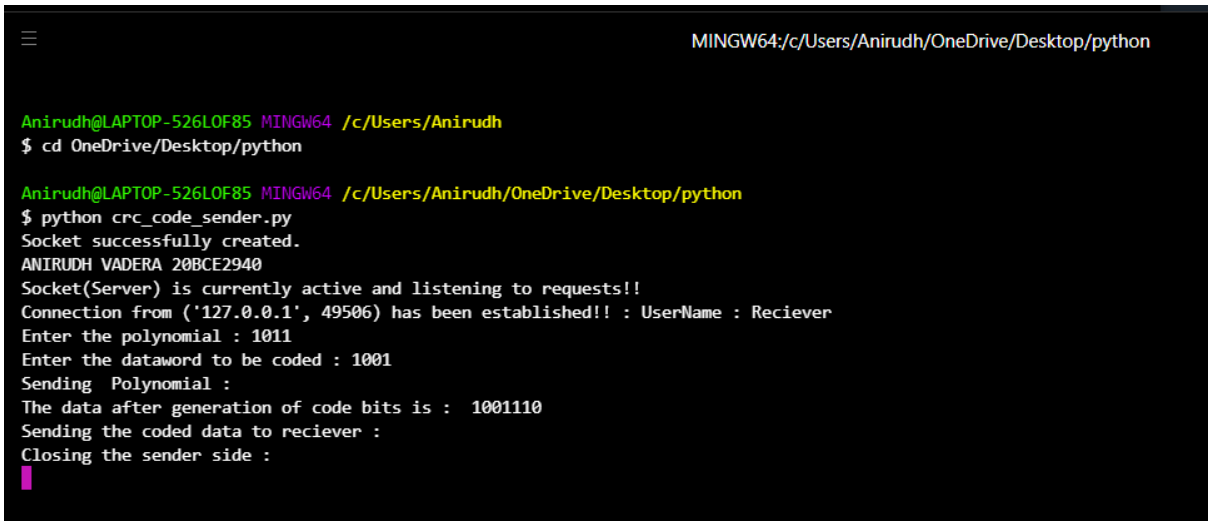


Sender.py:



(Code is damaged/corrupted) When the code is changed due to a noisy channel:

Sender.py:



Dataword: 1001

Polynomial(divisor): 1011

The code(dataword+remainder) sent is: 1001110

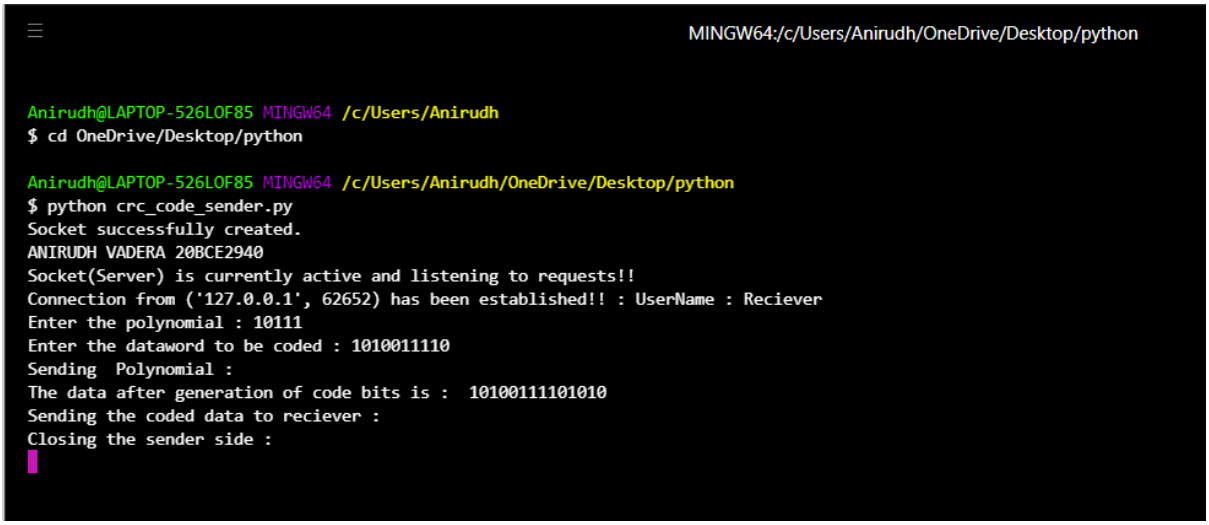
The code received is(at recievers side): 1101110

Error Occurred as syndrome is not 0 instead it is 111

Conclusion(Reciever.py):

Sending 10-bit data:

Sender.py:



Dataword: 1010011110

Polynomial(divisor): 10111

The code(dataword+remainder) sent is: 10100111101010

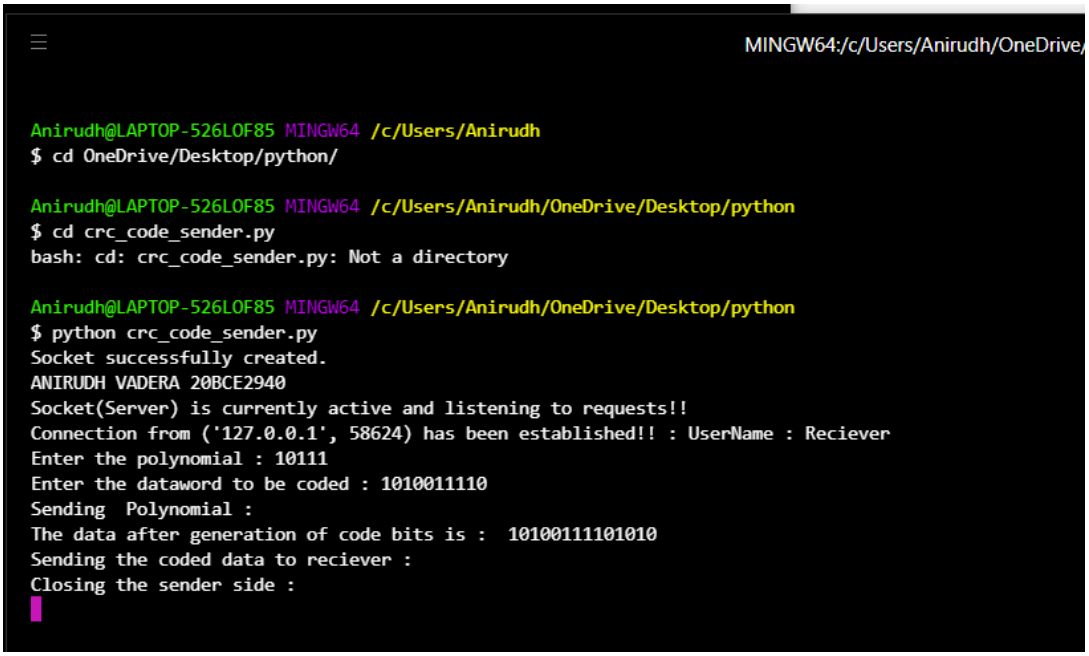
The code received is(at recievers side): 10100111101010

No Error

Conclusion(Reciever.py):

(Code is damaged/corrupted) When the code is changed due to a noisy channel:

Sender.py:



Dataword: 1010011110

Polynomial(divisor): 10111

The code(dataword+remainder) sent is: 10100111101010

The code received is(at recievers side): 10000111101010

Error Occurred as syndrome is not 0 instead it is 0111

Conclusion(Reciever.py):