**COMPUTER NETWORKS LAB**

**19131A0532**

**WEEK-4**

**Aim:**  4. Implement CRC and Hamming code for error handling.

**i)CRC:**

**Aim:**  To Implement CRC for error handling.

**Implementation:**

def xor(li1,dli,a,b):

    li2=[]

    for i in range(b):

        if li1[i]==dli[i]:

            li2.append(0)

        else:

            li2.append(1)

    return li2

nli=list(map(int,input("Enter the input:")))

dli=list(map(int,input("Enter the divisor:")))

print("senders data:",nli)

li1=[]

li2=[]

li3=[]

sender=[]

recieve=nli.copy()

for i in range(0,len(dli)-1):

    nli.append(0)

print("senders data after appending 0's:",nli)

a=len(nli)

b=len(dli)

li1=nli[0:b]

for i in range(b,a+1):

    if li1[0]==1:

        li1=xor(li1,dli,a,b)

    else:

        li3=[]

        for j in range(b):

            li3.append(0)

        li1=xor(li1,li3,a,b)

    li1=li1[1:]

    if i==a:

        break

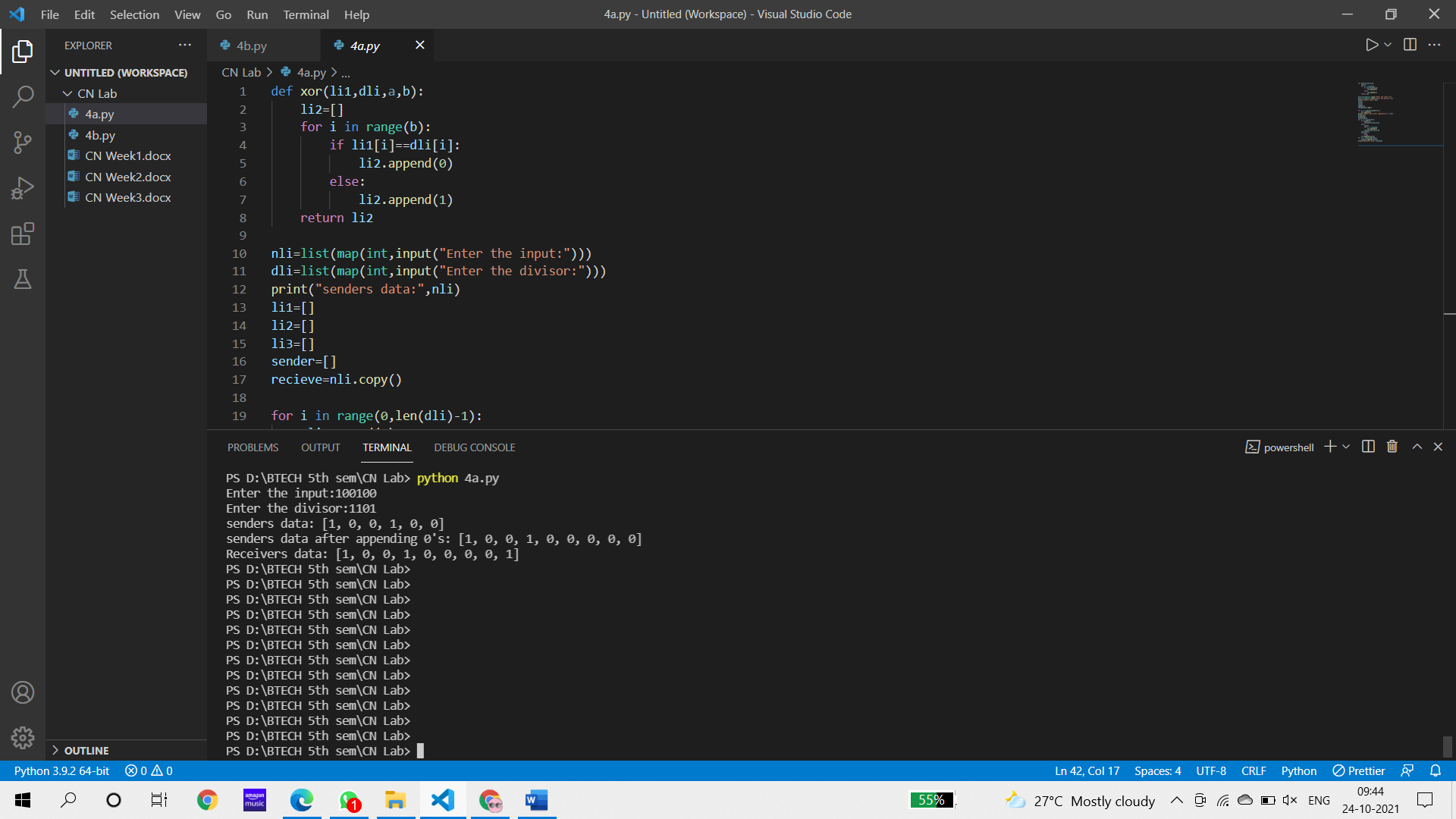
    li1.append(nli[i])

for i in range(len(li1)):

    recieve.append(li1[i])

print("Receivers data:",recieve)

**Output:**



**ii) Hamming Code:**

**Aim:**  Implement Hamming code for error handling.

**Implementation:**

import random

li=list(map(int,input("Enter the input to generate the hamming code:")))

m=len(li)

for r in range(0,m):

    if 2\*\*r>=m+r+1:

        break

li1=[0]\*(r+m+1)

for i in range(1,r+m+1):

    for j in range(r):

        if 2\*\*j==i:

            li1[i]="r"

j=0

for i in range(1,len(li1)):

    if li1[i]=="r":

        continue

    else:

        li1[i]=li[j]

        j+=1

print("After inserting redundancy bits at their positions:",li1)

li2=[]

li3=[]

li4=[]

li5=[]

for i in range(1,m+r+1):

    binary = bin(i).replace("0b","")

    while len(binary) < r:

        binary = "0"+binary;

    li2.append(binary)

k=1

for j in range(0,r):

    k=0

    li3=[]

    for item in li2:

        if item[j]=="1":

            li3.append(k+1)

        k=k+1

    li4.append(li3)

def Redundancy(li1,li4):

    for i in li4:

        count=0

        for j in i:

            if li1[j]==1:

                count+=1

        if count%2==1:

            li1[i[0]]=1

        else:

            li1[i[0]]=0

    return li1

print("Parity Bits Sequence:",li4)

print("Generated Hamming code:",Redundancy(li1,li4))

v=random.randint(1,len(li1)-1)

li1[v]=0

print("Replaced index with 0:",v)

print("Senders Code After Replacing:",li1)

li1=Redundancy(li1,li4)

print("Recievers code for detection:\nAfter Recieving:")

for i in range(1,r+m+1):

    for j in range(r):

        if 2\*\*j==i:

            li5.append(li1[i])

print(li5)

for i in li5:

    if i == 0:

        flag=1

    else:

        flag=0

        break

if flag==0:

    print("Error Occurred")

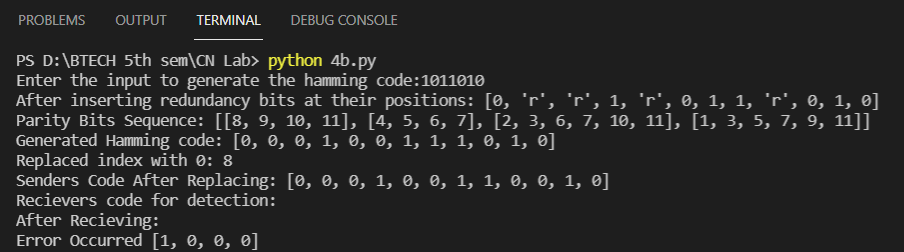
    li5.reverse()

    print("li5",li5)

else:

    print("Error did not occur")

**Output:**

****

**iii. Checksum**

**Aim:** To Implement checksum for error handling.

**Implementation:**

b1=input('Enter binary number\t')

def checksum(b1):

l=len(b1)

l1=l/4

k=0

b2=[]

b3=[]

b4=[]

b5=""

result=0

if(l1<=1):

print(b1)

else:

while l1!=0:

b2=b1[k:k+4]

result+= int("".join(str(i) for i in b2),2)

l1=l1-1

k+=4

a=result

b2=[]

while a!=0:

b2.append(a%2)

a=a//2

b2.reverse()

s=len(b2)-4

b3=b2[0:s]

b4=b2[s:s+4]

if(len(b3)!=0):

a=int("".join(str(i) for i in b3),2)

b=int("".join(str(i) for i in b4),2)

a=a+b

b2=[]

while a!=0:

b2.append(a%2)

a=a//2

b2.reverse()

b3=[]

if(len(b2)<4):

for i in range(4-len(b2)):

b3.append(1)

for j in range(len(b2)):

if b2[j]==1:

b3.append(0)

else:

b3.append(1)

if(len(b3)==0):

b3.append(b2)

b5= "".join(str(i) for i in b3)

return b5

rem=checksum(b1)

print(f"{rem} {b1}")

b2=input("enter binary number\t")

rem1=b2[0:4]

rem2=checksum(b2[4:])

if(rem1==rem2):

print("accepted")

else:

print("rejected")

**Output:**

