**PARITY**

def getParity( n ):

    parity = 0

    while n:

        parity = ~parity

        n = n & (n - 1)

    return parity

n=input("Enter the number: ")

n=int(n)

parity\_bit=( "1" if getParity(n) else "0")

print ("Even Parity bit of number ", n," = ", parity\_bit)

bit=str("{0:b}".format(n))

bit=bit+str(parity\_bit)

print("Encoded Message after appending parity bit: ", bit)

n=input("Enter the Encoded Message to check for Error: ")

n=str(n)

parity\_bit=n[-1]

orig=""

for i in range(0,len(n)-1):

    orig=orig+n[i]

orig= int(orig,2)

chk=( "1" if getParity(orig) else "0")

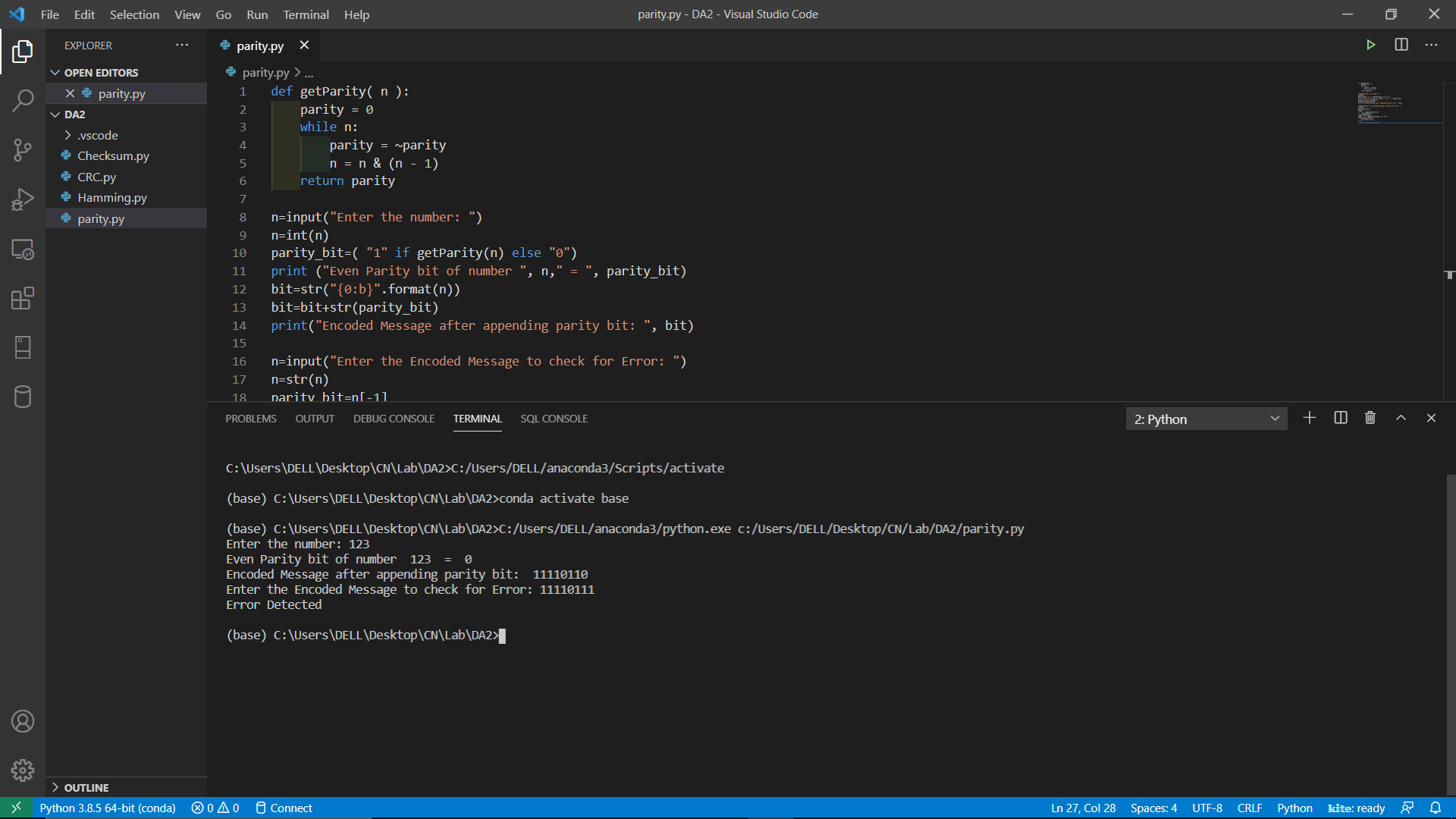
if(chk==parity\_bit):

    print("No Error")

else:

    print("Error Detected")

OUTPUT



**CRC**

def intToBinary(var):

    return (bin(var).split("0b")[1])

def binToInteger(var):

    return (int(var,2))

def stringToList(s):

    arr=['']\*(len(s))

    for i in range(len(s)):

        arr[i]=s[i]

    return arr

def listToString(arr):

    string = ""

    for element in arr:

        string += element

    return str(string)

def RemainderDecode(divisor,dividend):

    divisor=stringToList(divisor)

    dividend=stringToList(dividend)

    xorterm=''

    counter=0

    divisor=listToString(divisor)

    while (len(str(xorterm))<=len(divisor)):

        if (counter!=len(dividend)):

            xorterm = str(xorterm)+ dividend[counter]

            counter = counter + 1

        if (len(str(xorterm))==len(divisor)):

            xorterm=intToBinary(binToInteger(xorterm)^binToInteger(divisor))

            if (binToInteger(xorterm)==0):

                xorterm=''

        if (counter==len(dividend)):

            if xorterm==divisor:

                xorterm='0'

            return str(xorterm)

def Remainder(divisor, dividend):

    divisor=stringToList(divisor)

    dividend=stringToList(dividend)

    extension=len(divisor) - 1

    for i in range(extension):

        dividend.append('0')

    xorterm=''

    counter=0

    divisor=listToString(divisor)

    while (len(str(xorterm))<=len(divisor)):

        if (counter!=len(dividend)):

            xorterm = str(xorterm)+ dividend[counter]

            counter = counter + 1

        if (len(str(xorterm))==len(divisor)):

            xorterm=intToBinary(binToInteger(xorterm)^binToInteger(divisor))

            if (binToInteger(xorterm)==0):

                xorterm=''

        if (counter==len(dividend)):

            if xorterm==divisor:

                xorterm='0'

            return str(xorterm)

def CRCEncoder(divisor, dividend, remainder):

    n=len(divisor)-1

    remainder=str(remainder)

    while len(remainder)<n:

        remainder='0'+remainder

    dividend=dividend+remainder

    return dividend

#Encoding

divisor=str(input("Enter Divisor(in Binary Format): "))

dividend=str(input("Enter Dividend(in Binary Format): "))

print("The Remainder(error detection bit) is: ",Remainder(divisor,dividend))

print("The Encoded Message is: ",CRCEncoder(divisor,dividend,Remainder(divisor,dividend)))

#Decoding

dividend=str(input("Enter Encoded Message(in Binary Format): "))

divisor=str(input("Enter Divisor(in Binary Format): "))

if(RemainderDecode(divisor,dividend)==''):

    n='0'

else:

    n=RemainderDecode(divisor,dividend)

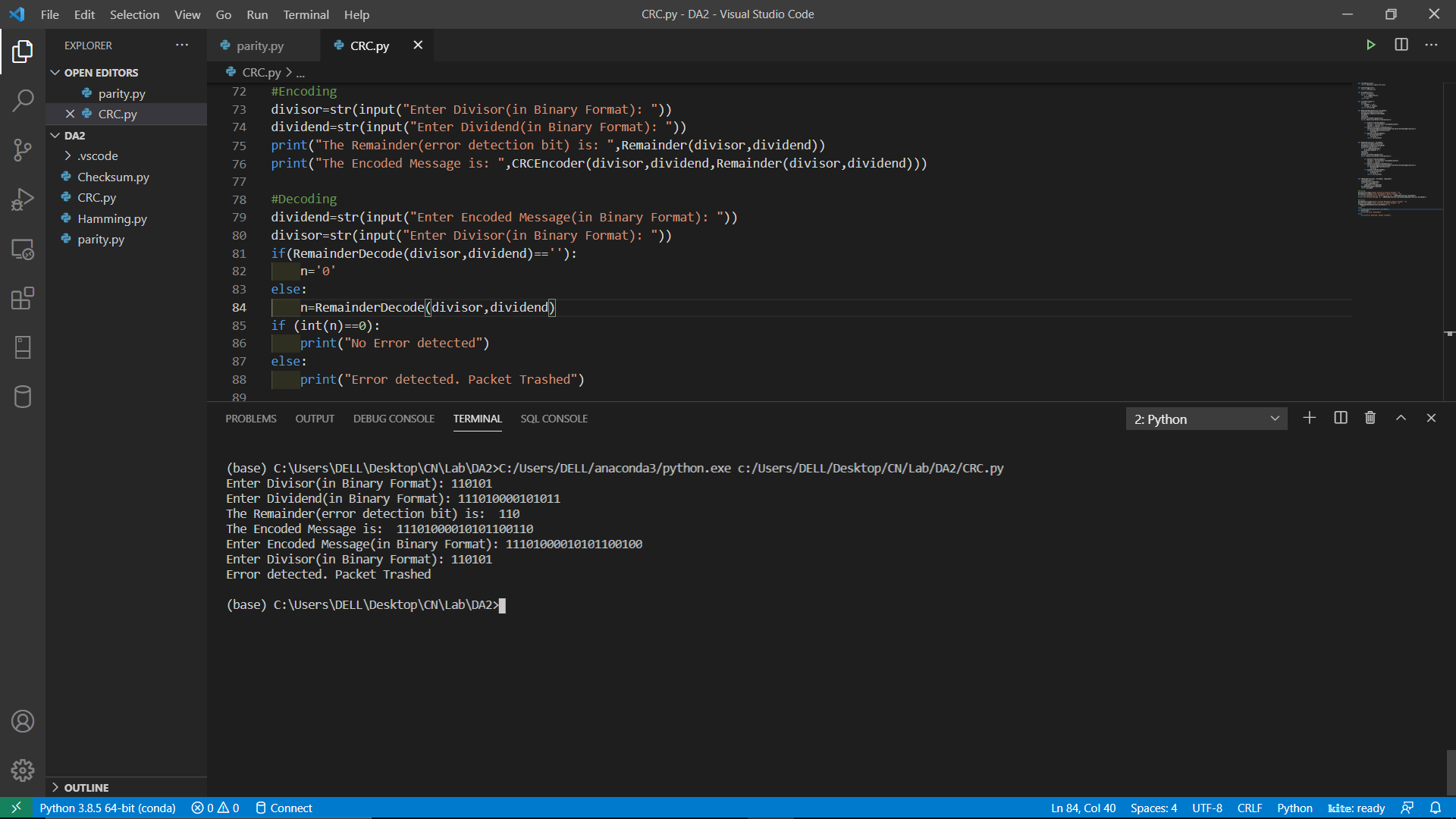
if (int(n)==0):

    print("No Error detected")

else:

    print("Error detected. Packet Trashed")

OUTPUT



**CHECKSUM**

def flip(c):

    return '1' if (c == '0') else '0'

def maximum(a, b):

    if a >= b:

        return a

    else:

        return b

def getOnes(bina):

    ones = ""

    for i in range(len(bina)):

        ones += flip(bina[i])

    return ones

def intToBinary(var):

    return (bin(var).split("0b")[1])

def binToInteger(var):

    return (int(var,2))

def BinaryAddition(binarr):

    sum=0

    for i in range(len(binarr)):

        temp=binToInteger(binarr[i])

        sum=sum+temp

    return(bin(sum).split('0b')[1])

def additionCompression(bina,length):

    arr=['']\*2

    counter=0

    while len(bina)>length:

        while (counter < length):

            arr[1]=bina[len(bina)-(counter+1)]+str(arr[1])

            counter=counter+1

        while(counter>=length and counter<len(bina)):

            arr[0]=bina[len(bina)-(counter+1)]+str(arr[0])

            counter=counter+1

        bina=BinaryAddition(arr)

    while len(bina)<length:

        bina='0'+bina

    return getOnes(bina)

**#CHECKSUM SENDING SIDE**

n=input("Enter the number of numbers: ")

n=int(n)

arr=['']\*n

length=0

for i in range(n):

    arr[i]=input("Enter Number (in integer without conversion to binary): ")

    arr[i]=intToBinary(int(arr[i]))

    length=maximum(length,len(arr[i]))

for i in range(n):

    while len(arr[i])<length:

        arr[i]='0'+arr[i]

Sum=BinaryAddition(arr)

print("Sum of the numbers:",Sum)

checksum=additionCompression(Sum,length)

print("Ones complement after compressing the addition: ",checksum)

arr.append(checksum)

print("Encoded Message:", arr)

**#CHECKSUM RECIEVING SIDE**

n=input("Enter the no. of words in the packet received (including checksum): ")

n=int(n)

arr=['']\*n

for i in range(n-1):

    print("Enter ",i+1," word in binary: ")

    arr[i]=input()

print("Enter checksum in binary: ")

arr[n-1]=input()

Sum=BinaryAddition(arr)

length=len(arr[0])

Check=additionCompression(Sum,length)

if binToInteger(Check)==0:

    print("No Error found")

    print("The encoded message is: ")

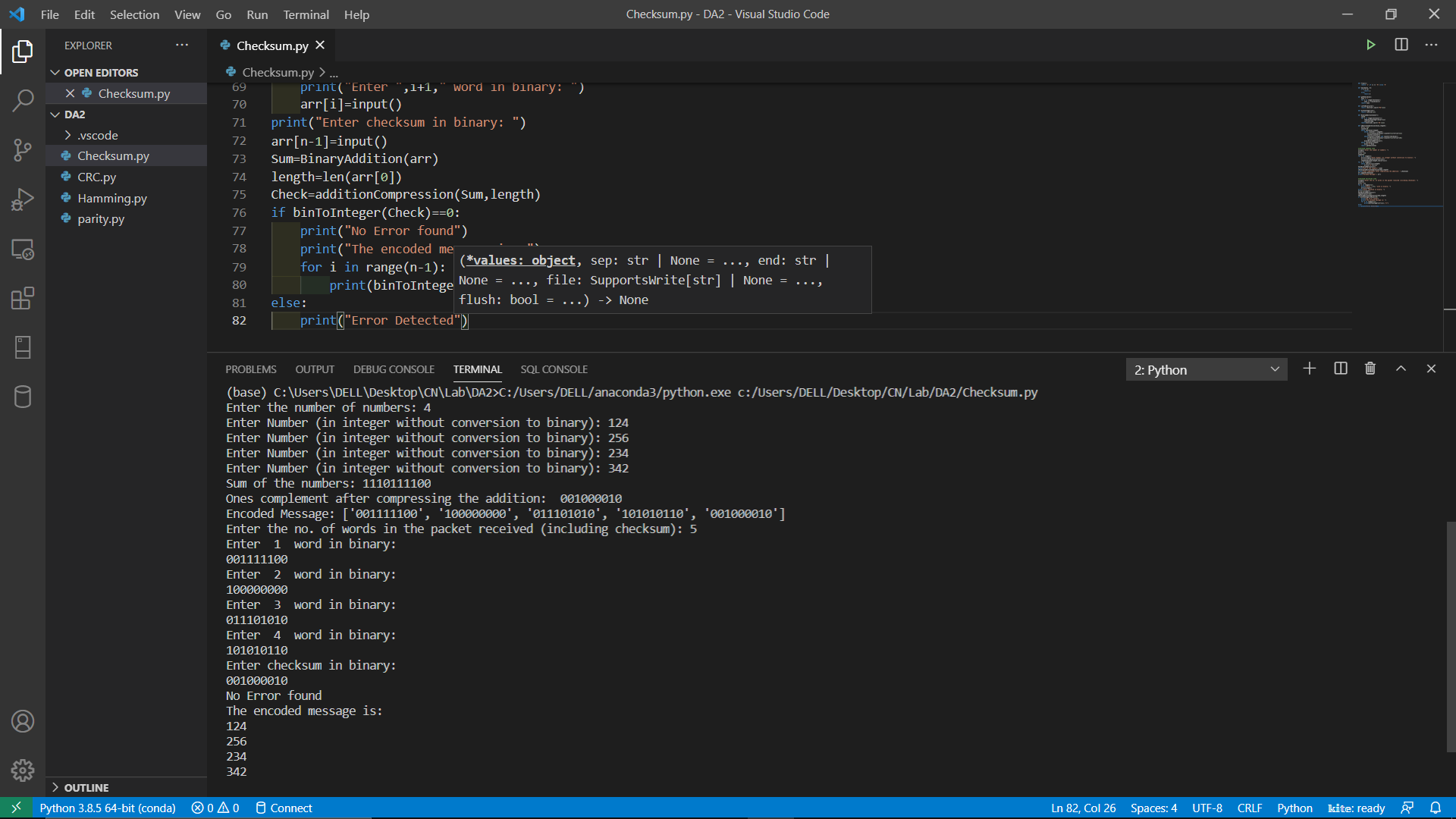
    for i in range(n-1):

        print(binToInteger(arr[i]),"\t")

else:

    print("Error Detected")

**OUTPUT**



**HAMMING CODE**

def calcRedundantBits(m):

    for i in range(m):

        if(2\*\*i >= m + i + 1):

            return i

def DecodeProcessing(arr):

    del arr[0]

    temp=(reverseArray(arr))

    return temp

def reverseArray(arr):

    temp=['']\*len(arr)

    for i in range(len(arr)):

        temp[i]=arr[-(i+1)]

    return temp

def getParity( n ):

    parity = 0

    while n:

        parity = ~parity

        n = n & (n - 1)

    return parity

def intToBinary(var):

    return (bin(var).split("0b")[1])

def binToInteger(var):

    return (int(var,2))

def listToString(arr):

    string = ""

    for element in arr:

        string += element

    return string

def arrRedundantPosition(r):

    arrRedundantPos=[]

    for i in range(r):

        arrRedundantPos.append(2\*\*i-1)

    return arrRedundantPos

def getParitybit(arr, r):

    temp=''

    for i in range(len(arr)):

        if (i & r == r) :

            temp += arr[i]

    return "1" if getParity(binToInteger(temp)) else "0"

def Hamming(message):

    print("Original message is: ", message)

    message=intToBinary(message)

    print("Message in binary is: ",message)

    m=len(message)

    r=calcRedundantBits(m)

    print("Number of redundant bits is: ",r)

    n=m+r

    arr=["0"]\*n

    arrRedundantPos=arrRedundantPosition(r)

    counter=0

    for i in range(n):

        if i not in arrRedundantPos:

            counter= counter+1

            arr[i]=message[-counter]

    arr.insert(0,'START')

    for i in range(len(arrRedundantPos)):

        arrRedundantPos[i] = arrRedundantPos[i]+1

    for i in arrRedundantPos:

        arr[i]=getParitybit(arr,i)

    for i in range(len(arr)-1):

        arr[i]=arr[i+1]

    arr.pop(-1)

    HammingArr=reverseArray(arr)

    print("The Hamming Code encoded message is: ",listToString(HammingArr))

def checkArrayRedundantBits(arr):

    n=len(arr)

    arr=[]

    for i in range(n):

        if 2\*\*i < n:

            arr.append(2\*\*i)

    return arr

#Encoding

n=int(input("Enter the number which you want to encode(in Decimals): "))

Hamming(n)

#Decoding

n=input("Enter the Encoded Message: ")

print("The Encoded Message is: ", n)

arr=['']\*(len(n))

for i in range(len(n)):

    arr[i]=n[i]

arr=reverseArray(arr)

RedArr=checkArrayRedundantBits(arr)

arr.insert(0,'START')

ErrorPos=[]

for i in RedArr:

    ErrorPos.append(getParitybit(arr, i))

if binToInteger(listToString(reverseArray(ErrorPos)))==0:

    print("No Error Found")

    arr=DecodeProcessing(arr)

else:

    errordet=binToInteger(listToString(reverseArray(ErrorPos)))

    print("Error Found at ",errordet, " position from the left(Big Endian)")

    arr=DecodeProcessing(arr)

    if (arr[-(errordet)]=='1'):

        arr[-errordet]='0'

    else:

        arr[-errordet]='1'

    print("The Corrected Encoded Message is: ",listToString(arr))

CArrBin=[]

for i in range(len(RedArr)):

    RedArr[i]=-RedArr[i]

for i in range(len(arr)):

    if -(i+1) not in RedArr:

        CArrBin.append(arr[-(i+1)])

CArrBin=reverseArray(CArrBin)

print("The Decoded Message in binary is: ", listToString(CArrBin))

print("The Message Recieved is: ", binToInteger(listToString(CArrBin)))

**OUTPUT**

