# JADAVPUR UNIVERSITY

# Computer Networks Lab

Assignment 1

By Gaurav Bhagat

Roll number – 302310501002 Class – BCSE 3rd year

Group – A1 Date of submission – 16-08-2024

Assignment 1 : Design and implement an error detection module which has two schemes namely Checksum and Cyclic Redundancy Check(CRC)

O B J E C T I V E : Test the above two schemes for the error types and CRC polynomials mentioned above for the following cases (not limited to).

- Error is detected by both CRC and Checksum.
- Error is detected by checksum but not by CRC.
- Error is detected by CRC but not by Checksum

# DESIGN:

I have implemented the assignment by making a PACKAGE named ErrorDetection .

## 1. Package: ErrorDetection

This package focuses on different error detection mechanisms commonly used in network communication to ensure data integrity. It contains the following files:

- **CRCReciever.py & CRCSender.py**:
  - Implements the **Cyclic Redundancy Check (CRC)** error-detecting mechanism.
  - Both methods have the common divisor that is used to detect error in the transmitted data.
- **ChecksumSender.py & ChecksumReceiver.py**:
  - Implements the **Checksum** error-detection mechanism.
  - In this methods, the data is divided into equal-sized segments(16 bits). The sum of these segments is calculated and sent along with the data. The receiver recalculates the sum and compares it with the transmitted checksum to check for errors.
- **Inputdata.txt**:
  - This file is designed to take input for the transmission.
- **fileReader.py**:
  - this file is used to read the data from the inputData.txt.

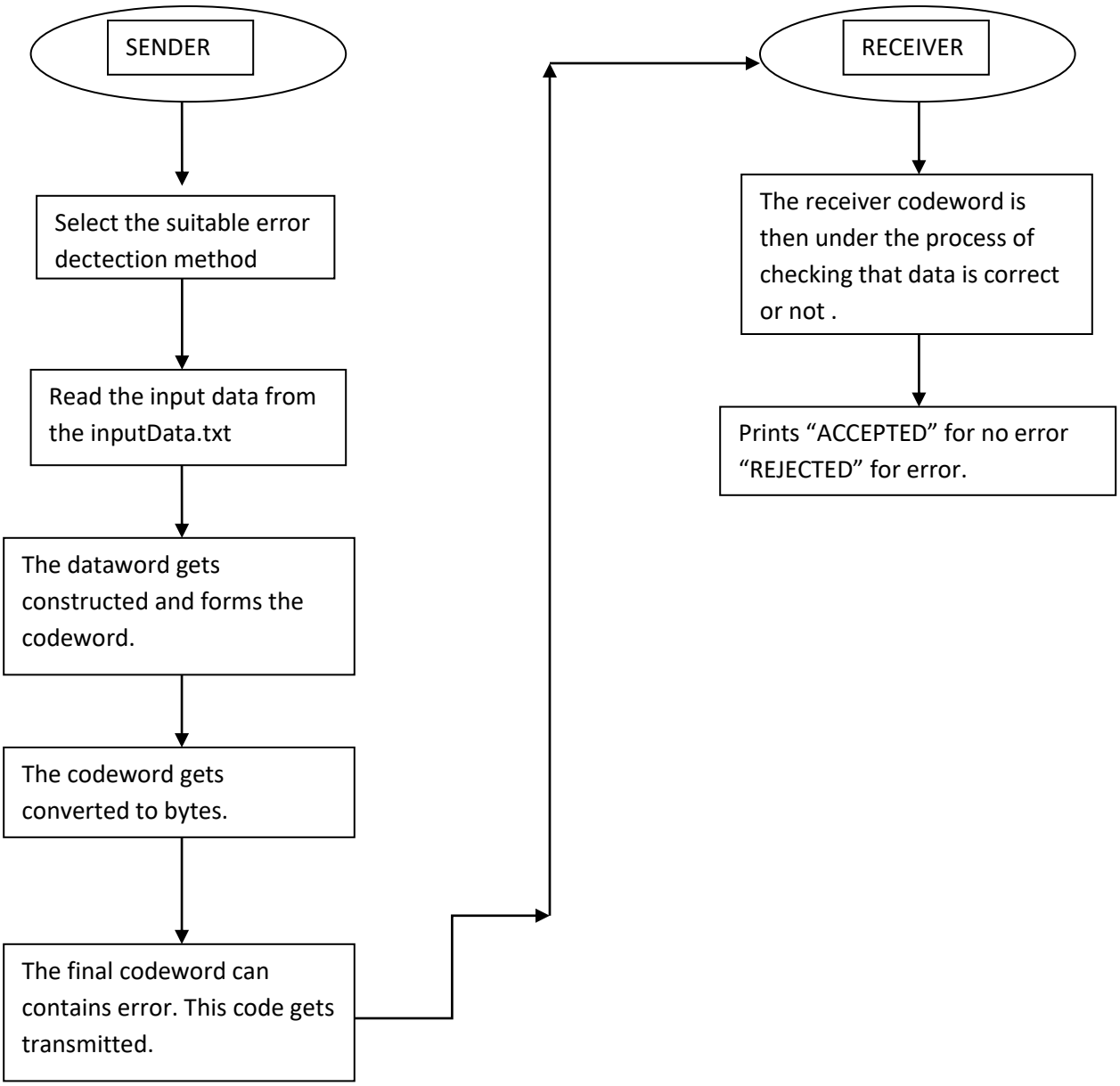
## 2. Socket Programming

This part of the assignment focuses on client-server communication using socket programming in Python. The files are:

- **server.py**:
  - Implements the server-side logic of a socket-based communication system.
  - The server listens for connections from clients, processes requests, and sends responses. Here it is used to receive the data from client side and give responses accordingly.
- **client.py**:
  - Implements the client-side logic.
  - The client establishes a connection with the server, sends data. It has also some function to inject error .
  - The client is basically sends the data with or without error.

## Overall Structure and Use Case

- **Error Detection**: This section provides tools to ensure the integrity of data being transmitted. It can be particularly useful in network communications.
- **Socket Programming**: This section allows for practical implementation of client-server communication, which is involve in sending and receiving data while ensuring its integrity using the error detection mechanisms provided in the ErrorDetection package.



## Input & Output Formats:

For input of data there is a text file named InputData.txt that contains the data which has to be fetched by the sender. That data is in string and for transmission we have to convert it into bytes.

Output is basically a Boolean . For Correct data printing “Accepted!!”.

If the transmission data gets corrupted then prints ”Rejected!!”.

## IMPLEMENTATION:

### 1. *ChecksumReceiver.py*

```
def setWrapSum(sum):
    temp =sum
    if(sum > 0xFFFF):
        temp = temp & 0xF0000
        temp = temp>>16
        sum += temp
        sum = sum & 0x0FFFF
    return sum

def calculate_checksum(data):
    sum =0
    for i in range(0,len(data),16):
        byte = data[i:i+16]
        sum += int(byte,2)
    wrapsum = setWrapSum(sum)
    checksum = (~wrapsum & 0xFFFF)
    return format(checksum,'016b')

def validate_checksum(data):
    calculated_checksum=calculate_checksum(data)
    return calculated_checksum == format(0,'016b')


def checkingChecksum(receivedData):
    if(validate_checksum(receivedData)):
        print("NO ERROR\nACCEPTED!!\n\n")
    else:
        print("ERROR!!\nREJECTED!!\n\n")
```

## 2. ChecksumSender.py

```
from ErrorDetection.fileReader import readingR

def setWrapSum(sum):
    temp =sum
    if(sum > 0xFFFF):
        temp = temp & 0xF0000
        temp = temp>>16
        sum += temp
        sum = sum & 0x0FFFF
    return sum

def calculate_checksum(data):
    sum =0
    for i in range(0,len(data),16):
        byte = data[i:i+16]
        sum += int(byte,2)
    wrapsum = setWrapSum(sum)
    checksum = (~wrapsum & 0xFFFF)
    return format(checksum,'016b')

def startCheckSum():
    data = readingR()
    x = calculate_checksum(data)  #sender
    data +=x
    return data
```

## 3.CRCSender.py

```
from ErrorDetection.fileReader import readingR


def crc(data, divisor):
    div_len = len(divisor)
    temp = data[0: div_len]
    while div_len < len(data):
        if temp[0] == '1':
            temp = strXor(divisor, temp) + data[div_len]
        else:
            temp = strXor('0' * div_len, temp) + data[div_len]
        div_len += 1

    if temp[0] == '1':
```

```
            temp = strXor(divisor, temp)
        else:
            temp = strXor('0' * div_len, temp)
        check = temp
        return check


    def strXor(a, b):
        result = ''
        for i in range(1, len(b)):
            if a[i] == b[i]:
                result += '0'
            else:
                result += '1'
        return result

    def checkCRC():
        data = readingR()
        divisor = "11101011"
        appended_data = data + '0' * (len(divisor) - 1)
        checkcrc = crc(appended_data, divisor)
        #print("Remainder is:", checkcrc)
        #print("Data to be sent to Receiver:",data + checkcrc)
        return (data + checkcrc)
```

## 4.CRCRECEIVER.PY

```
    def crc(data, divisor):
        div_len = len(divisor)
        temp = data[0: div_len]
        while div_len < len(data):
            if temp[0] == '1':
                temp = strXor(divisor, temp) + data[div_len]
            else:
                temp = strXor('0' * div_len, temp) + data[div_len]
            div_len += 1

        if temp[0] == '1':
            temp = strXor(divisor, temp)
        else:
            temp = strXor('0' * div_len, temp)
        check = temp
        return check
```

```
def strXor(a, b):
    result = ''
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result += '0'
        else:
            result += '1'
    return result

def checkingCRC(data):
    divisor = "11101011"
    crcsum = crc(data, divisor)
    rem = '0' * (len(divisor) - 1)
    if crcsum == rem:
            print("No error\nACCEPTED!!\n\n")
    else:
            print("Error!!\nREJECTED!!\n\n")
```

### *5.FILEREADER.PY*

```
import os

def readingR():  #InputData.txt reading...
    directory = os.getcwd()
    subpackage = 'ErrorDetection'
    filename =input("Enter the file name:")
    filepath = os.path.join(directory,subpackage,filename)
    f=open(filepath,'r')
    data = f.read()
    f.close()
    return data
```

### *6.INPUTDATA.TXT*

THIS TEXT FILE CONTAINS THE DATA FOR THE TRANSMISSION.

FOR EG.

1000100100010000

## SOCKET PROGRAMMING PART

THIS IS FOR THE TRANSMISSION OF THE DATA.

### SERVER.PY

```
from socket import *
import socket
import sys
import ErrorDetection.CRCReciever as a
import ErrorDetection.ChecksumReciever as b
s = socket.socket(family = AF_INET , type=SOCK_STREAM)
s.bind(('127.0.0.1',12345))
s.listen(5)

while True:
    try:
        print("Server is waiting")
        clt,addr = s.accept()
        print("client connected from",addr)
        while True:
            try:
                choice = clt.recv(1024)
                if not choice:
                    break
                choice = int.from_bytes(choice,
byteorder='big')
                if(choice == 1):
                    data = clt.recv(1024)
                    data = data.decode('utf-8')
                    sum = a.checkingCRC(data)
                else:
                    data = clt.recv(1024)
                    data = data.decode('utf-8')
                    sum = b.checkingChecksum(data)
            except socket.error as e:
                    print(f"Socket error: {e}")
                    break
            except Exception as e:
                    print(f"Unexpected error: {e}")
```

```
                    break
            clt.close()
            print("Connection closed.")
        except Exception as e:
            print(f"Failed to accept connection: {e}")

s.close()
print("Server shut down.")  # sever never shut downs....
```

CLIENT.PY

```
from socket import *
import socket
import sys
import ErrorDetection.CRCSender as a
import ErrorDetection.ChecksumSender as b
import random


def generateBit(len):
    return random.randint(0, len-1)

def generateTwoBits(len):
    a = random.randint(0, len//2)
    b = random.randint(len//2 + 1, len-1)
    if(a+1 == b):
        b = b + 1
    return {a,b}

def generateBurstError(data):
    n = len(data) -1
    st = random.randint(0,n//2)
    end = random.randint((n//2+1) ,n)
    dataList = list(data)
    for i in range (st,end+1):
        dataList[i] = '0' if dataList[i] == '1' else '1'
    return dataList


def generateoddErrors(data):
    n = len(data)
    odd=[]
    for i in range (1,n):
        if(i%2 == 1):
```

```
            odd.append(i)
    oddlen = len(odd)
    i = random.randint(0,oddlen)
    ind = odd[i]
    dataList =list(data)
    uniqueBits = random.sample(range(n),ind)
    for i in uniqueBits:
        dataList[i] = '0' if dataList[i] == '1' else '1'
    print("odd error:",ind)
    return dataList


def noisyChannel(data):
    choice = int(input("\nEnter 1 FOR single bit error:\n2 FOR double
bit error:\n3 FOR Odd numbers of errors\n4 FOR Burst Error:"))
    if(choice == 1):
        ind = generateBit(len(data))  # have choices to select the
error type....
        dataList = list(data)
        dataList[ind] = '0' if  dataList[ind] == '1' else '1'

    elif(choice == 2):
        #ind1 , ind2 = generateTwoBits(len(data))
        ind1 = 4
        ind2 = 36
        dataList = list(data)
        dataList[ind1] = '0' if  dataList[ind1] == '1' else '1'

        dataList[ind2] = '0' if  dataList[ind2] == '1' else '1'

    elif(choice == 3):
        dataList = generateoddErrors(data)

    else:
        dataList = generateBurstError(data)

    data = ''.join(dataList)
    return data


c = socket.socket(family = AF_INET , type=SOCK_STREAM)
c.connect(('127.0.0.1',12345))

z='y'
try:
    while z!='n':
```

```
        choice = int(input("\nEnter 1 FOR CYCLIC REDUNDENCY CHECK:\n2
FOR CHECKSUM:\n"))

        ch = choice.to_bytes(4, byteorder='big')
        c.send(ch)

        if(choice == 1):
            data = a.checkCRC()

            print("\nData to be transferred:",data)
            fu = int(input("\nEnter 1 FOR Noisy Channel:\n 2 FOR
Noiseless Channel:\n"))

            if(fu == 1):
                data = noisyChannel(data)     #give choices to noisy
or noiseless channels..
                print("\nData transferred due to the noisy
channel:",data)

            else:
                print("\nNo error has enduced as the channel is
Noiseless!!")
            c.send(data.encode('utf-8'))
        else:
            data = b.startCheckSum()

            print("Data to be transferred:",data)

            fu = int(input("\nEnter 1 FOR Noisy Channel:\n 2 FOR
Noiseless Channel:\n"))

            if(fu == 1):
                data = noisyChannel(data)     #give choices to noisy
or noiseless channels..
                print("\nData transferred due to the noisy
channel:",data)

            else:
                print("\nNo error has enduced as the channel is
Noiseless!!")
            c.send(data.encode('utf-8'))
        x = input("Want to send for data(y/n):")
        z = x.lower()
except KeyboardInterrupt:
    print("Exit")
c.close()
```

THIS CLIENT SIDE PROGRAMMING HAS THE FUNCTIONS FOR GENERATING THE RANDOM ERROR IN THE DATA TO BE TRANSMITTED. IT CAN GENERATE SINGLE BIT ERROR, DOUBLE BIT ERROR, ODD NUMBERS OF ERROR, BURST ERROR. ALL THIS FUNCTIONS ARE ABLE TO INJECT RANDOM BIT FILPS.

TEST CASES:

1.SINGLE BIT ERROR

| SL. NO. | DATAWORD | DATAWORD WITH ERROR | CHECKSUM | CRC-8 | CRC-10 | CRC 16 | CRC 32 |
|---|---|---|---|---|---|---|---|
| 1. | 0110110001101011<br>0110000101110100 | 0110110001101011<br>0110000101110101 | REJ | REJ | REJ | REJ | REJ |
| 2. | 0010000001001010<br>0110000101100100 | 0010010001001010<br>0110000101100100 | REJ | REJ | REJ | REJ | REJ |

CONCLUSION – REJECTED BY BOTH THE METHODS.

2.DOUBLE BIT ERROR

| SL. NO. | DATAWORD | DATAWORD WITH ERROR | CHECKSUM | CRC-8 | CRC-10 | CRC 16 | CRC 32 |
|---|---|---|---|---|---|---|---|
| 1. | 1111111011111111<br>1111111111111111 | 1111111111111111<br>1111111011111111 | ACC | REJ | REJ | REJ | REJ |
| 2. | 0010000001001010<br>0110000101100100 | 0010010001001010<br>0110000101100100 | REJ | REJ | REJ | REJ | REJ |

ACCEPTED BY CHECKSUM BUT NOT BY CRC.

```
Enter 1 FOR CYCLIC REDUNDENCY CHECK:
2 FOR CHECKSUM:
2
Enter the file name:InputData.txt
Data to be transferred: 1111111111111111111111111111111100000000000000000

Enter 1 FOR Noisy Channel:
 2 FOR Noiseless Channel:
1

Enter 1 FOR single bit error:
2 FOR double bit error:
3 FOR Odd numbers of errors
4 FOR Burst Error:2

Data transferred due to the noisy channel: 1111011111111111111111111111111100001000000000000
Want to send for data(y/n):
```

```
PS C:\Users\Gaurav\OneDrive\Desktop\BCSE\3RD YEAR\Computer Networks Lab\GAURAV-III\ASSIGNMENT 1> & C:/Python312/python.exe "c:/Users/Gaurav/OneDrive/Desktop/BCSE/3RD YEAR/Computer Networks Lab/GAURAV-III/ASSIGNMENT 1/server.py"
Server is waiting
client connected from ('127.0.0.1', 60728)
NO ERROR
ACCEPTED!!
```

### 3.ODD NUMBERS OF ERROR

| SL. NO. | DATAWORD | DATAWORD WITH ERROR | CHECKSUM | CRC-8 | CRC-10 | CRC 16 | CRC 32 |
|---|---|---|---|---|---|---|---|
| 1. | 0001100001101011 0001100001110000 | 0110100111001011 0011100001000011 | ACC | REJ | REJ | REJ | REJ |
| 2. | 0010111010100111 0100110010111100 | 1000101011101110 0011100110101100 | ACC | REJ | REJ | REJ | REJ |

CONCLUSION – IT HAS TO BE RUN ABOUT 100000 TIMES TO GET THE DATAWORD WITH ERROR TO BE ACCEPTED.

ACCEPTED BY CHECKSUM BUT NOT BY CRC.

```
AR\Computer Networks Lab\GAURAV-III\ASSIGNMENT 1\client.py"
Data to be transferred: 10101110101001110100110010111100000010010011100
```

```
MENT 1/server.py"
Server is waiting
client connected from ('127.0.0.1', 54944)
NO ERROR
ACCCCCCCCEEEEPPPPTED!!
```

4.Burst error

| Sl. No. | Dataword | Dataword with error | checksum | Crc-8 | Crc-10 | Crc 16 | Crc 32 |
|---|---|---|---|---|---|---|---|
| 1. | 1111000001000001 1111110000000011 | 1111000001000001 1111111111111111 | Acc | Rej | Rej | Rej | Rej |
| 2. | 1000001111000000 1111111111101000 | 0011000001000000 0110011011111101 | Acc | Rej | Rej | Rej | Rej |

Accepted by Checksum but not by crc.

```
I/ASSIGNMENT 1/client.py"
Data to be transferred: 10000011110000001111111111101000011111000101010110
```

```
Server is waiting
client connected from ('127.0.0.1', 55771)
NO ERROR
ACCCCCCCCEEEEPPPPTED!!
```

- Testcase for the dataword to be accepted by the CRC and rejected by checksum

For this we know that the error(e(x)) must be divisible by the g(x)(generator polynomial) .

| Sl. No. | dataword | Codeword with error | checksum | Crc-8 | Crc-10 | Crc 16 | Crc 32 |
|---|---|---|---|---|---|---|---|
| 1. | 1101001110100011 1111011110010000 | 1101001110011001 0101011110010000 | Rej | Acc | Rej | Rej | Rej |
| 2. | 1110010101010001 1110000000111110 | 1101010101010001 0100000000111110 | Rej | Rej | Acc | Rej | Rej |

Crc-8

```
-III\ASSIGNMENT 1> & C:/Python312/python.exe "c:/Users/Gaurav/OneDr
/BCSE/3RD YEAR/Computer Networks Lab/GAURAV-III/ASSIGNMENT 1/client

Data to be transferred: 1101001110100011111101111001000011101 00

orks Lab\GAURAV-III\ASSIGNMENT 1\server.py"
Server is waiting
client connected from ('127.0.0.1', 55821)
No error
ACCEPTED!!
```

Crc – 16

```
/BCSE/3RD YEAR/Computer Networks Lab/GAURAV-III/ASSIGNMENT 1/client.p

Data to be transferred: 1110010101010001111000000011111011110011 11
```

```
Server is waiting
client connected from ('127.0.0.1', 55858)
No error
ACCEPTED!!
```

CRC-32 IS IMMUNE TO ALL TYPES OF ERROR . IT DETECTS ALL TYPES OF ERROR GENERATED.CRC-16 ALSO GET TRICKED BY SOME RANDOM ERROR GENERATED BY THE FUNCTIONS AND ACCEPT THE WRONG DATA.

AS WE ARE GENERATING RANDOM ERROR WE HAVE TO CHECK ABOUT 100000 TIMES TO GET THE METHODS DOING WRONG CALCULATIONS.

ANALYSIS:

- **SINGLE BIT ERROR**: BOTH CRC AND CHECKSUM METHODS CONSISTENTLY REJECTED ERRONEOUS DATA, DEMONSTRATING HIGH ACCURACY IN DETECTING SINGLE-BIT ERRORS. THIS IS EXPECTED AS BOTH METHODS ARE DESIGNED TO DETECT EVEN THE SMALLEST ALTERATIONS IN THE DATA.

- **DOUBLE BIT ERROR**: HERE, THE CHECKSUM METHOD SOMETIMES ACCEPTS ERRONEOUS DATA WHILE CRC METHODS (CRC-8, CRC-10, CRC-16, AND CRC-32) TYPICALLY REJECT IT. THIS HIGHLIGHTS A LIMITATION OF THE CHECKSUM METHOD, WHERE ITS EFFECTIVENESS CAN DIMINISH WITH SPECIFIC TYPES OF ERRORS.

- **ODD NUMBERS OF ERRORS**: SIMILAR TO THE DOUBLE-BIT ERROR CASE, THE CHECKSUM METHOD OCCASIONALLY FAILS TO DETECT ERRORS, WHILE CRC METHODS CONTINUE TO SHOW ROBUSTNESS. THIS SCENARIO FURTHER UNDERLINES THE ADVANTAGE OF USING CRC OVER CHECKSUM IN ENVIRONMENTS WHERE DATA INTEGRITY IS CRITICAL.

- **Burst Error**: The Checksum method often accepts data with burst errors, which are significant sequences of erroneous bits. In contrast, all CRC methods reject such errors, proving CRC's superiority in detecting complex error patterns.

- **Case Where CRC Fails and Checksum Succeeds**: There are rare cases where CRC may accept an erroneous dataword if the error polynomial happens to be divisible by the generator polynomial. This scenario highlights that while CRC is robust, it is not infallible.

**CRC-32 is highly reliable .there are many cases where the checksum gets incorrect.there are very rare and documented cases for which the crc gets the calculations wrong.**

Improvements:

1. **Enhanced Checksum**: One potential improvement is to enhance the Checksum method by introducing additional checks or combining it with another error detection mechanism to improve its accuracy.

2. **Implementing CRC-32 Across the Board**: Given the robustness of CRC-32 in detecting all types of errors, standardizing on CRC-32 for all error detection might provide more consistent results across different error types.

Comments:

The coding part is challenging but finding the cases for which the crc will not work is tough , finding cases for checksum is little bit easy. Coding in python makes the things simpler for me but I have to always search for the function and methods that used in python. This process of making report give me a good idea about the error detection.