LONDON METROPOLITAN UNIVERSITY

ITAHARI INTERNATIONAL COLLEGE

Module Code & Module Title

CS4051NT Fundamentals of Computing

Assessment Weightage & Type

100% Individual Coursework

Year and Semester

2019-20 Spring

Student Name: Susan Shrestha

London Met ID: 19033540

College ID: NP05CP4S200039

Assignment Due Date: 20th September 2020

Assignment Submission Date: 20th September 2020

Contents

Table of Figures

Table of Tables

Introduction

Python is an objected oriented, high-level and general purpose programming language designed and developed by Guido van Rossum in between 1985- 1990. It can be used by both beginners and professionals as Python has simple easy to use syntax which makes it more understandable. (Python Tutorial)

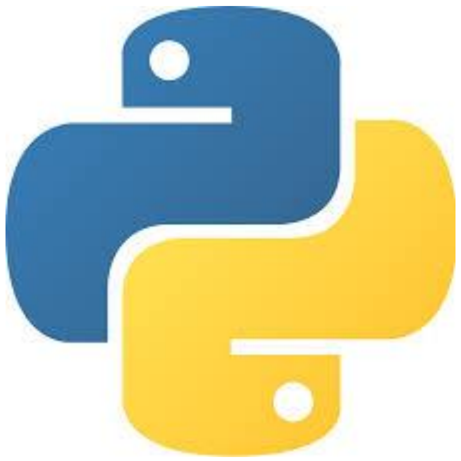


Figure 1: Python logo

Features of Python:

- It is an object oriented programming language supporting functional and structured programming.
- It has a special feature of automatic garbage collection.
- It can be easily combined with various programming languages like C, C++, CORBA and JAVA.
- It can be used as scripting language.
- It is an open source programming language.
- It is portable and platform programming independent language that means code written in windows can be easily shifted on Mac OS or Linux.

Application of Python:

- It can be used for web application development because of its framework that python uses to create applications.
- Python can also be used for game development because it provides libraries such as PySoy that is a 3D game engine and PyGame.
- Machine learning and Artificial Intelligence
- Data visualization and Data science
- GUI based application development

(Python Tutorial)

Aim and objective:

This particular coursework is an individual task given to all students learning this module. From my point of view, the main aim of this coursework is to learn python programming and its implementation on creating byte adder program using the model based on bit added as given in the coursework question using different data structures of python and presenting the program in suitable CLI (Command Line Interface). After the completion of coursework, the main aim is to implement python programming in real life as well.

The objective of this coursework is:

- To construct the model of byte adder assembled using gates based on the model of bit adder given in question.
- To write an algorithm for the program to add two integer values based on bitwise operation.
- To write pseudo code for the program and design the flow chart.
- To code a python program that implements the byte adder model.
- To test the program and validate it.
- To write the report to present the whole work.

Software used to complete the coursework:

For completing the coursework, I have used software like:,

- **MS Word** to write the report,

Figure 2: MS-Word

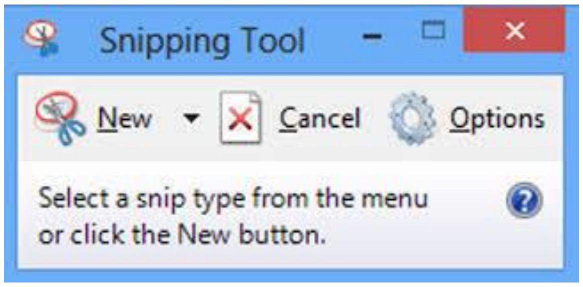
- **Logic.ly** to make the byte adder model,

Figure 3: logic.ly

- **Python IDLE** to code the program,

Figure 4: Python IDLE

- **Snipping tool and MS Paint** for screenshots and diagram design,



Select a snip type from the menu or click the New button.

Figure 5: Snipping Tool

Model

1. Bit adder model

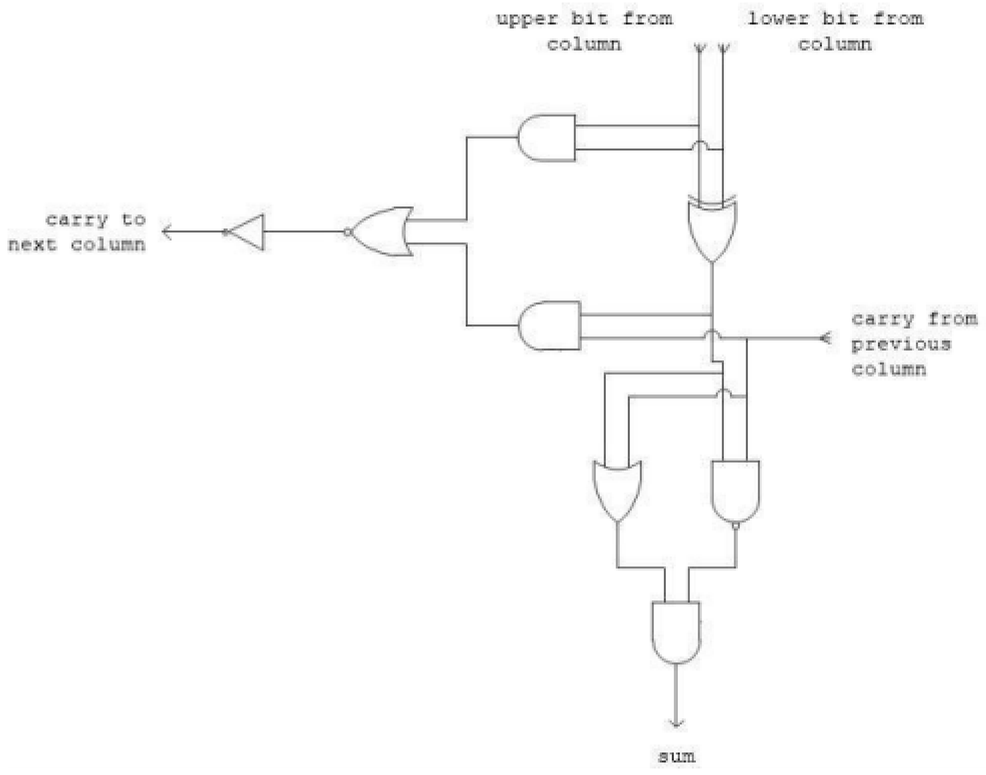


Figure 6: bit adder model

The above circuit diagram is bit adder model provided by the coursework question, which is used to add two binary digits (bits) producing output as sum and carry. Here, in this model two different digits are given as input as A and B that are given as upper bit and lower bit respectively which is processed as,

For sum: input A and B are XORed first, which is ORed with the XORed output and carry input initially as 0 or the carry may be from previous column, also at the same time the XORed output is NANDed with the same carry input that is taken in OR operation and finally the both ORed output and NANDed output are ANDed that produces our final sum.

For carry: input A and B are ANDed, at the same time input A and B are XORed, after that the output of XOR operation and carry initially as 0 or may be from previous column is ANDed and the AND operation of A and B is NORed with AND operation of XOR output and carry input. Finally, the NORed output is passed through the NOT gate that produces the carry output.

In this bit adder model, total 6 gates are used namely; OR, AND, NOR, NAND, NOT and XOR.

To make this model more understandable, I am using truth table using combinations of 0 and 1 for three inputs A, B and carry in that produces the output as sum and carry out.

Input			**Output**	
A	**B**	**Carry in**	**sum**	**Carry out**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table for Bit adder model

2. Byte (8-bit) adder model

As the bit adder model can only add two binary digits, for adding two different binary number of 8 bit each, I designed the byte adder model following the bit adder model as shown in the below image,

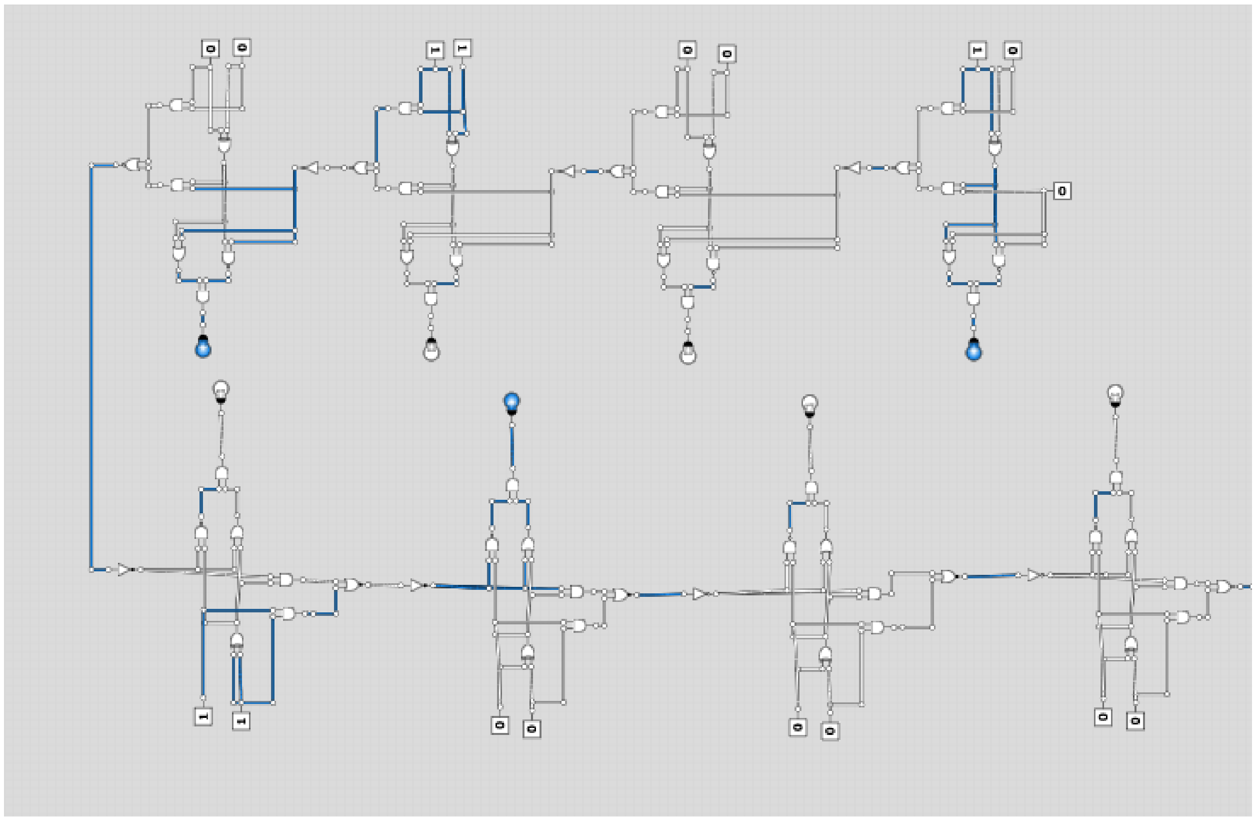


Figure 7: Byte (8-bit) adder model

I designed this model using logic.ly. In this model, firstly, I made the bit adder and then made 7 copies of that model and then connected them. Here, I used bulb for the output preference that is if the bulb is blue the output is 1 and if the bulb is white the output is 0.

Looking forward, I connected these adders using carry that is initially the carry is 0 and after the first adder is done producing sum and carry output the carry output of first adder is carry input to the second adder and likewise the carry output of second adder is carry input to third adder and likewise for the rest adders.

In my byte adder model design, I took 2 binary numbers 00010101 and 00010100. Tracing out these binary numbers in program, the last digits of both binary numbers i.e. 1 and 0 are taken as input for the first adder where 0 is initially taken as carry input. Also for second adder our model takes the input as 0 and 0 respectively from both binary numbers' second last digit and for third adder the third last digit from both binary numbers are taken and similar for other adders too. So, performing the bit addition using the bit adder model, we get,

A = 1 (first input)

B = 0 (second input)

Carry in = 0 (initial)

Operation for sum = ((A ⊕ B) ∨ Carry in) ∧ ((A ⊕ B) $\barwedge$ Carry in))

= (1 ∨ 0) ∧ (1 $\barwedge$ 0)

= 1 ∧ 1

Sum = 1

Operation for carry = $\neg\ \overline{((A \land B) \lor ((A \oplus B) \land Carry\ in))}$

$= \neg\ \overline{(0 \lor (1 \land 0))}$

$= \neg\ \overline{(0 \lor 0)}$

$= \neg\ \bar{0}$

Carry = 0

Now, the sum of first adder is the last digit of our addition of two binary numbers and the carry output of the first adder is the carry input to the second adder and sum of second adder will be placed before the last digit and its carry output is given as carry input to third adder and likewise the whole model works.

As shown in Figure 7, the blue lighted bulb i.e. ON means 1 and white bulb i.e. OFF means 0 can be easily seen which represent our output i.e. sum of two 8-bit binary numbers. The addition of last adder is the first digit of sum, the output of second last adder is second digit of sum and likewise the output will be,

```
        0 0 0 1 0 1 0 1
      + 0 0 0 1 0 1 0 0
Sum = 0 0 1 0 1 0 0 1
```

To make my design model more understandable, I am using truth table to validate my user input binary numbers,

Input			Output	
A	**B**	**Carry in (previous carry out)**	**sum**	**Carry out**
1	0	0	1	0
0	0	0	0	0
1	1	0	0	1
0	0	1	1	0
1	1	0	0	1
0	0	1	1	0
0	0	0	0	0
0	0	0	0	0

Table 2: Truth Table for user input binary numbers

In this truth table, the user input binary numbers can be seen on input column and the final output is achieved by going down to up in the sum column.

Exactly, this technique is used in the coding part of the program also. First, the program asks the user to input the binary or decimal choice and if decimal is chosen then it asks for two different decimal numbers and validates the input then converts the decimal number to binary 8-bit by adding 0 to the front if required and adds the binary digits one by one as per the designated byte (8-bit) adder model and stores the added digits in an array and lastly, reverse the array items and prints. If binary mode is chosen by the user it asks the user to input two binary numbers and directly add those using byte adder model and stores the added digits in an array and lastly, reverse the array items and prints.

3. Parallel Circuit Model

Parallel circuits are usually made to understand the circuit diagrams or model in more easy and convenient way.

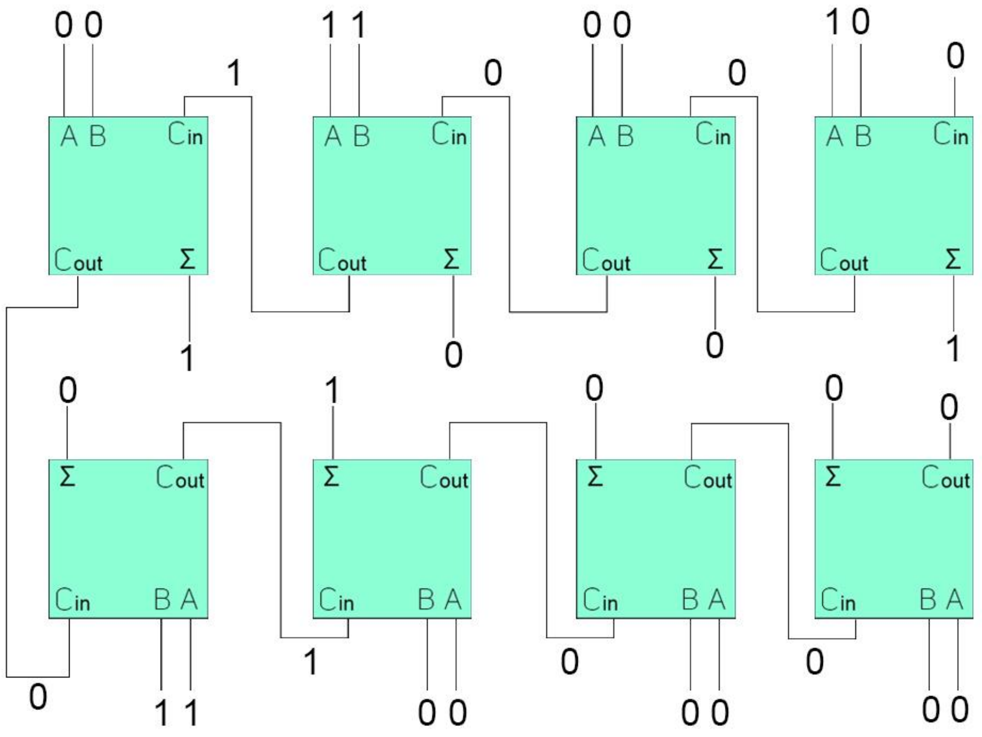


Figure 8: Parallel Circuit Model (User Input)

The above figure depicts the parallel circuit model for 8-bit adder where, all the inputs are taken from user input which produces the sum and carry value in parallel manner.

In figure 8: the denotations are as, ∑ = sum

Cin = Carry in,

Cout = Carry out

A and B = Input

Algorithm

Algorithm is a step by step process to solve the problem by providing a certain set of inputs to produce the required output. Algorithm can be used for solving a simple problem as well as complex operations. It is usually written before writing the pseudo code or program which makes coding efficient and the flow of program can be understood easily. Algorithm is written using English language so it can be understood easily by non-programmers also.

For my coursework program, the algorithm is as:

Step 1: Choose either decimal or binary mode.

Step 2: If decimal mode is chosen go to step 3 else go to step 14.

Step 3: Enter the first decimal number and validate it.

Step 4: Typecast first decimal number and store it as integer data type.

Step 5: Enter the second decimal number and validate it.

Step 6: Typecast second decimal number and store it as integer data type.

Step 7: Add two decimal numbers and display the sum.

Step 8: Convert first decimal number and second decimal number to eight bit binary number by appending 0s in front of converted binary number as if required.

Step 9: Take from last to first bit of both first and second eight bit binary number and perform binary addition of each bit one by one using the full adder model made using logic gates.

Step 10: Now, store each output in an array list.

Step 11: Reverse the array list and display the binary sum.

Step 12: Choose to continue or exit.

Step 13: If continue mode is chosen go to Step 1 else close the program.

Step 14: Enter the first binary number and validate it.

Step 15: Typecast first binary number and store it in data structure.

Step 16: Enter the second binary number and validate it.

Step 17: Typecast second binary number and store it in data structure.

Step 18: Convert first and second binary number to eight bit binary number by appending 0s in front of converted binary number as if required.

Step 19: Go to Step 9, 10, 11.

Step 20: Convert both binary input numbers to decimal number and perform addition of decimal numbers and display the sum of decimal numbers.

Step 21: Go to Step 12.

Pseudocode

Pseudocode can be defined as an informal process of writing a program in which no programming language syntax is followed. It is used to create rough draft of the program. Basically, pseudocode is written using English language and half programming language, not following any grammatical rules. (What is Pseudocode)

For this particular coursework, the pseudo code for methods in each python module or file is as:

1. main module

```
import from getInput, adder file

function main():

    Set flag as False

    while flag is False:

        Ask user: “Enter d/D for decimal number and b/B for binary number mode”

        if choice is d:

            call decimalMethod()

        else if choice is b:

            call binaryMethod()

        else:

            print “invalid choice”

        Set correctInput as False

        while correctInput is False:

            Ask user: “Enter c to continue and e to exit the program”
            if input is e:
                Set flag as True and correctInput as True
            else if input is c:
                Set flag as False and correctInput as True
            else:
                Print “Enter a valid choice”
function decimalMethod():
    Declare binaryNumberList as List
    Set decimalNumberList = call getInput(“d”)
    Set firstDecimalNumber = decimalNumberList[0]
    Set secondDecimalNumber = decimalNumberList[1]
    print “Adding in Decimal method”
    print “firstDecimalNumber”
    print “secondDecimalNumber”
    pSum = firstDecimalNumber + secondDecimalNumber
    print pSum
    set firstEightBitBinaryNumber = call convertToBinary(firstDecimalNumber)
    set secondEightBitBinaryNumber = call convertToBinary(secondDecimalNumber)
    append firstEightBitBinaryNumber, secondEightBitBinaryNumber in
binaryNumberList
    Set binarySum = call adder(binaryNumberList)
    For i in binarySum:
        print i, end = " "
function binaryMethod():
    Declare binaryNumberList as List
    Set binaryNumberList = call getInput("b")
    Set binaryNumber1 = binaryNumberList[0]
    Set binaryNumber2 = binaryNumberList[1]
    Set fitstEightBitBinaryNumber = call eightBit(binaryNumber1)
    Set secondEightBitBinaryNumber = call eightBit(binaryNumber2)
    Append fitstEightBitBinaryNumber, secondEightBitBinaryNumber in
binaryNumberList
    Set binarySum = call adder(binaryNumberList)
    for i in binarySum:
        print I, end = " "
    print "Adding in decimal method"
    Set firstDecimalNumber = call convertToDecimal(binaryNumber1)
    Set secondDecimalNumber = call convertToDecimal(binaryNumber2)
    Set pSum = firstDecimalNumber + secondDecimalNumber
    print pSum
Run main()
```

2. getInput module

```
import form conversion file

function getInput(choice):

    Declare inputData[] as List

    if choice is d:

        Set validDecimalInput as False

        while validDecimalInput is Fasle:

            Set firstNumber = Ask user: “Enter first decimal number”

            If call isValidDecimalInput(firstNumber):

                If firstNumber < 0 or firstNumber > 255

                    Print “Enter decimal number between 0 and 255”

            else:

                continue

            Set secondNumber = Ask user: “Enter second decimal number”

            If call isValidDecimalInput(secondNumber):

                If secondNumber < 0 or secondNumber > 255

                    Print “Enter decimal number between 0 and 255”

            else:

                continue

            Set input1 = int(firstNumber)

            Set input2 = int(secondNumber)
            if (input1+input2) > 255:
                print “Enter numbers with sum less than 255”
            else:
                Append input1, input2 in inputData[]
            return inputdata[]
    else if choice is b:
        Set validBinaryInput as False
        while validBinaryInput is Fasle:
            Set number1 = Ask user: “Enter First binary number”
            if call isValidBinaryInput(number1):
                if call convertToDecimal(number1) < 0 or call
            convertToDecimal(number2) > 255:
                    print “Enter valid 8-bit binary number”
                else:
                    print ("Please enter a valid binary number!")
            Set number2 = Ask user: "Enter second binary number:"
            if call isValidBinaryInput(number2):
                if call convertToDecimal(number2) < 0 or call
            convertToDecimal(number2) > 255:
                    print "Please enter valid 8 digit binary number!"
            else:
                print "Please enter a valid binary number!”
            if call (convertToDecimal(number1) + call
            convertToDecimal(number2)) > 255:
                print "Enter binary numbers having sum less than
            11111111")
            else:
                Append call split(number1), split(number2) in inputData[]
                Set ValidBinaryInput as True
        return inputData[]
function split(word):
    return [int(char) for char in word]
function isValidDecimalInput(number):
    try:
        Typecast number to integer
        return True
    except:
    if number is null:
        print "You have not entered any input"
        return False
    else:
        print “Do not enter the float and alphabetical value!")
        return False
function isValidBinaryInput(input):
    try:
        for i in input:
            if i not in 0, 1:
                print “Please enter a valid binary number!"
                return False
    except:
        if number is null:
            print “Do not enter empty field!"
            return False
        else:
            print “Please, do not enter the float and alphabetical value!”
            return False
    return True
```

3. conversion module

```
function convertToDecimal(number):
    Set decimalNumber = 0
    Set i = 0
    for j in range(length of number -1, -1, -1):
        Set decimalNumber += int(number[j]) * (2**i)
        i += 1
    return decimalNumber
function convertToBinary(number):
    Declare reversedBinaryList as List
    Declare binaryList as List
    if number is 0:
        Append 0 in binaryList
    else:
        while number is not 0:
            Set r = number % 2
            Append value of r in reversedBinaryList
            Set number //= 2
    for i in range(length of reversedBinaryList -1, -1, -1):
        Append reversedBinaryList[i] in binaryList
    Set eightBitBinaryList = call eightBit(binaryList)
    return eightBitBinaryList
function eightBit(binaryList):
    if length of binaryList is not equal to 8:
        for i in range(length of binaryList), 8:
            Insert (0,0) in binaryList

    return binaryList
```

4. adder module

```
import from conversion and gates file

function adder(binaryList):

    Set binaryNumber1 = binaryList[0]

    Set binaryNumber2 = binaryList[1]

    Set carry = 0

    Declare reverseSum, binarySum as List

    print “Adding in Binary Method”

    for i in binaryNumber1:

        print I, end = “ “

    for i in binaryNumber2:

        print i, end = “ “

    for i in range(length of binaryNumber1 -1, -1, -1):

        Set bit1 = int(binaryNumber1[i])

        Set bit2 = int(binaryNumber2[i])

        XOR = XOR_gate(bit1,bit2)

        OR = OR_gate(XOR,carry)

         NAND = NAND_gate(XOR,carry)

        Sum = AND_gate(OR,NAND)
```

```
            AND1 = AND_gate(bit1,bit2)
            AND2 = AND_gate(XOR,carry)
             carry = NOT_gate(NOR_gate(AND1,AND2))
            Append sum in reverseSum
    for i in range(length of reverseSum -1, -1, -1):
            Append reverseSum[i] in binarySum
    return binarySum
```

5. gates module

```
function AND_gate(a,b):
    return (a & b)
function OR_gate(a,b):
    return (a | b)
function XOR_gate(a,b):
    return (a ^ b)
function NOT_gate(a):
    return (~a) + 2
function NAND_gate(a,b):
    return NOT_gate(AND_gate(a,b))
function NOR_gate(a,b):
    return NOT_gate(OR_gate(a,b))
```

Flowchart

Flowchart is a graphical or pictorial representation of a process, system, computer algorithm or workflow of a program. It is mostly used in various fields to understand study, document and communicate mostly complex processes in a simple and easy to understand diagrams. Flowchart uses a defined set of shapes such as oval, rectangle, diamond and other shapes to represent the type of process in a flow. A flowchart can be a simple hand drawn diagram as well as complex computer drawn diagram illustrating multiple steps.

Talking about use and necessity of flowchart in computer programs/algorithms, some of them are,

- It helps in visualizing the execution of code in a program.
- It helps to understand how user uses the program.
- It represents the structure of the program.
- It shows the way how code is organized.

(What is a Flowchart)

Some commonly used defined shapes while making a flowchart are,

1. **Terminator symbol:** This symbol is used to represent the Start point, End point and outcomes of a path.

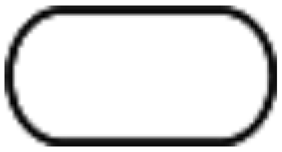


Figure 9: Terminator symbol

2. **Process symbol:** It is also called as action symbol as it represents an action or process or function in a flowchart. It is mostly used symbol while making

flowchart.

3. **Decision symbol:** It indicates a question to be answer; usually yes/no or true/false. It is also known as branching symbol.

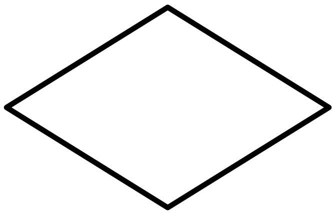


Figure 11: Decision symbol

4. **Data symbol:** It is also known as Input/output symbol. It represents the input or output and resource to be used or generated.

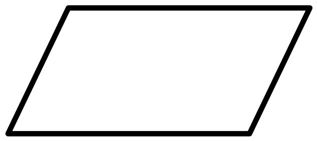


Figure 12: Data symbol

5. **Document symbol:** It is used to represent the input or output of a document.

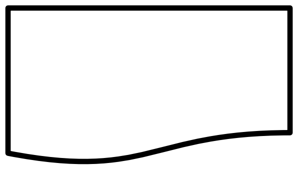


Figure 13: Document symbol

(Flowchart symbol and notations)

The flowchart for byte adder program is presented below,

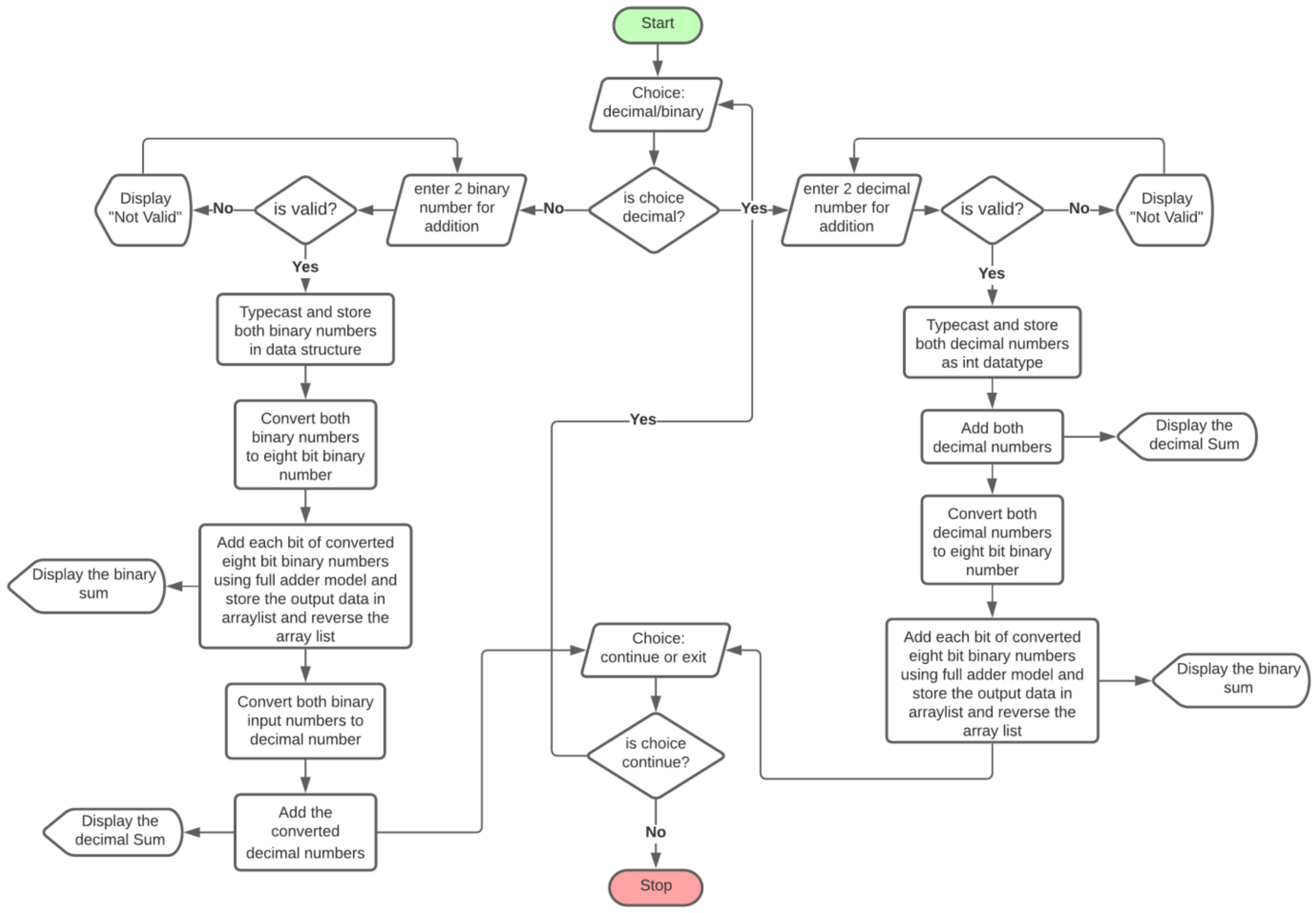

Start
Choice: decimal/binary
is choice decimal?
Yes
No
enter 2 binary number for addition
is valid?
No
Display "Not Valid"
Yes
Typecast and store both binary numbers in data structure
Convert both binary numbers to eight bit binary number
Add each bit of converted eight bit binary numbers using full adder model and store the output data in arraylist and reverse the array list
Display the binary sum
Convert both binary input numbers to decimal number
Add the converted decimal numbers
Display the decimal Sum
enter 2 decimal number for addition
is valid?
No
Display "Not Valid"
Yes
Typecast and store both decimal numbers as int datatype
Add both decimal numbers
Display the decimal Sum
Convert both decimal numbers to eight bit binary number
Add each bit of converted eight bit binary numbers using full adder model and store the output data in arraylist and reverse the array list
Display the binary sum
Choice: continue or exit
is choice continue?
Yes
No
Stop

Figure 14: Flowchart

The above flowchart represents the flow of 8-bit adder program. In that, the program can add two integers or two decimal numbers as per the user decision.

Data structure

Data structure is the process of organizing, managing and storage format that enables efficient access and modification of data. (Black, 2004) Some of the data structures mostly used while programming in python are lists, tuples, dictionaries, strings, sets and frozensets. Lists, strings and tuples are ordered sequences of objects. Unlike strings that stores only characters, list and tuples can contain any type of objects. Lists and tuples are like arrays. Tuples like strings are immutables. Lists are mutables so they can be extended or reduced at will. Sets are mutable unordered sequence of unique elements whereas frozensets are immutable sets. (Cokelaer)

In python programming, there are data structures such as Strings, Lists, Dictionary, Sets and Tuple. These data structures are used as per their requirement in the code.

The data structures that I used in my python program code are,

1. Strings

Basically, string is a sequence of characters where a character is a symbol that can be alphabets, numbers, or any Unicode characters. In python, strings are created by enclosing characters in single or double quotes. (programiz)

For example, strAbc = ‘abc’ or strDef = “def” or strText = “Hello world”

In my coursework, I used string to take input from the user as well as for choice input and to store other values such as binary sum values and also to display the information.

For example,

- To get the input in lowercase, I used lower() method of string for choice such as if(choice.lower() == "d"):
  This line of code will take the input in lower case ever if it is entered in uppercase

- To store and display the information or values in the string I used print() method pSum=firstDecimalNumber+secondDecimalNumber print(pSum)

This will first add two numbers and store in the string variable pSum and print it

2. Lists

A list in python is an ordered collection of items which is changeable and it allows the duplicate members. It is the most versatile data structure in python which is written as a list of items separated by comma within the square bracket. Items in a list can be of different data types. (w3schools)

Creating a list in python is very simple and easy. **For example,**

list1 = [‘susan’, ’bipin’, ’December’, 2005, 40563]

To access, update or manipulate the items in the list there are some methods of list object. They are append(), extend(), insert(), remove(), pop(), clear(), index(), count() reverse(), copy(), etc.

In this coursework, I used list to lists for storing the collection for storing the input data from the user as well as to store converted binary numbers from decimal, binary number collection and to access the specific items from the list.

For example,

- To add the items in the list, I used append() method,

  binaryNumberList.append(firstEightBitBinaryNumber)

  This line of code will append firstEightBitBinaryNumber to the list.

- To access the items in the list, I used index value,
  binaryNumber1 = binaryNumberList[0]
  This line of code will store the 0th item in binaryNumberList[] in binaryNumber1 String variable.

- For converting binary number to eightBit binary number I used insert() method,

```
if(len(binaryList) !=  8):
        for i in range(len(binaryList),8):
            binaryList.insert(0,0)
    return binaryList
```

This block of code will first check if the binaryList is of length 8 or not, if the length is not equal to 8 then, a for loop will run for 8 times and insert 0 value til next 0 is found and return the binaryList once done.

Testing

In programming or software development, testing is one of the important parts to validate the final program considering its reliability, performance and speed. One can test the program by creating several test cases such giving invalid input and see how the program responds on that or if the program is giving valid input or not. Also the programmer can identify bugs and errors on the program and code after doing some general testing that will help in debugging and improve the performance of the program.

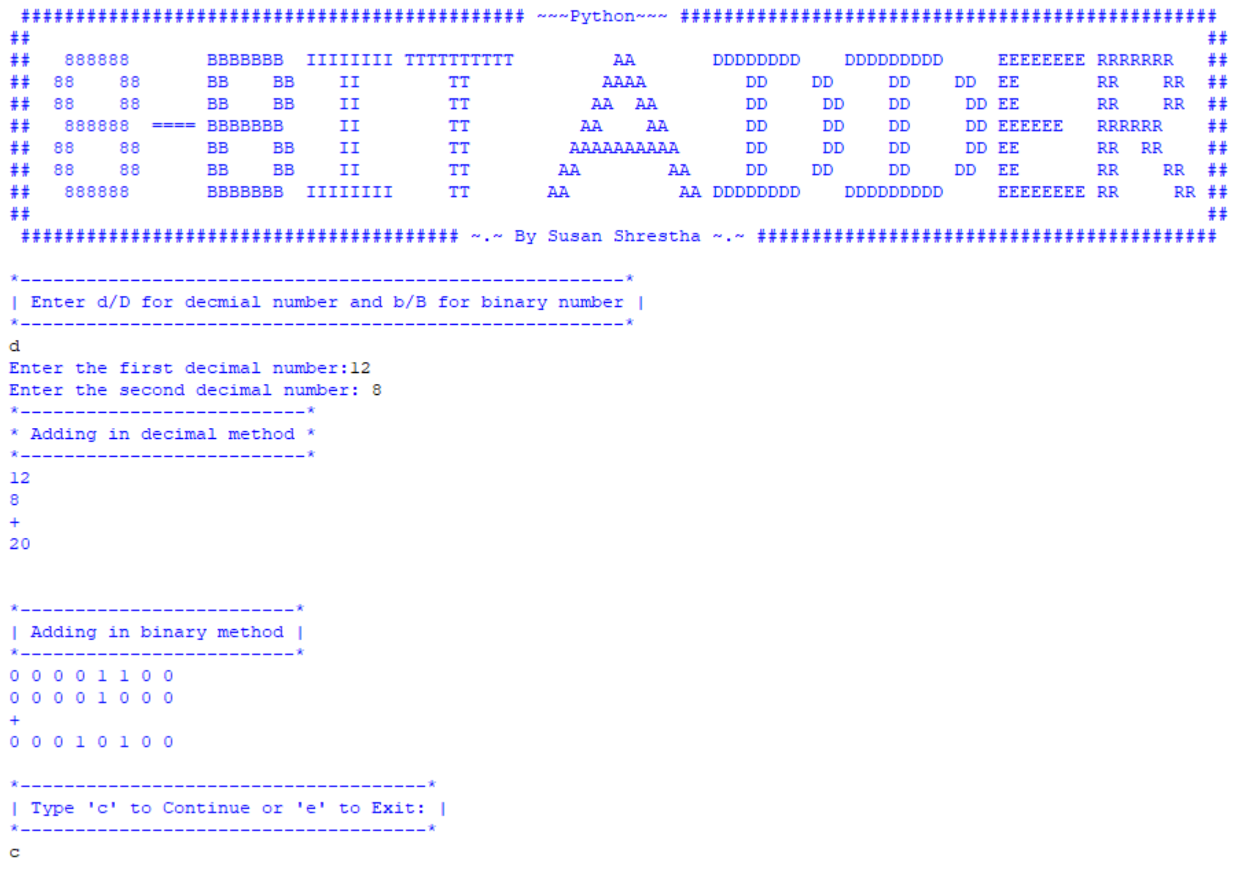
So to validate my final python program code, I have done several test on some test cases. The tests are,

Test 1:

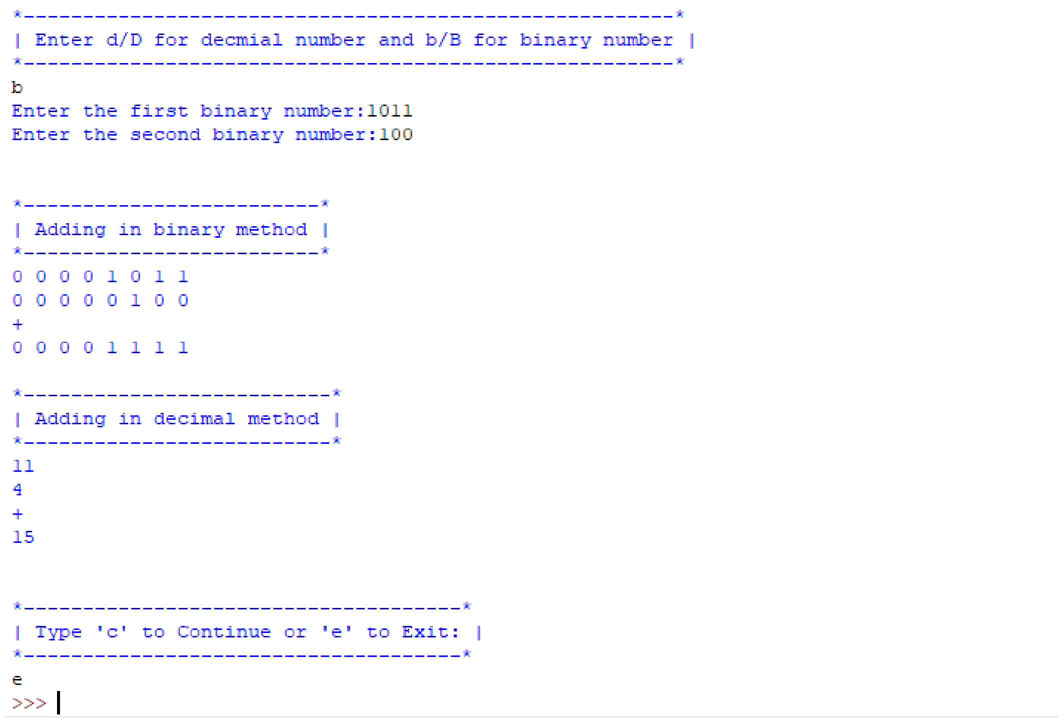
Testing if the program can perform both binary and decimal addition on normal test input values,

Objective	To perform both binary and decimal addition
Action	Run the program, Choose decimal mode and enter 1st and 2nd decimal number as 12 and 8, It will print the decimal and binary addition, Choose continue to run the program, Now, choose binary mode and enter 1st and 2nd binary number as 1011 and 100, It will print both decimal and binary addition Choose exit to close the program,
Expected Result	Both decimal and binary addition with proper output will be displayed
Actual Result	Both decimal and binary addition with proper output is printed
Conclusion	Test succeed

Table 3: Test 1



```
 ############################################## ~~~Python~~~ #################################################
##                                                                                                           ##
##   888888       BBBBBBB  IIIIIIII TTTTTTTTTT         AA       DDDDDDDD    DDDDDDDDD     EEEEEEEE RRRRRRR   ##
##  88    88      BB    BB    II        TT            AAAA         DD    DD     DD    DD  EE       RR    RR  ##
##  88    88      BB    BB    II        TT           AA  AA        DD     DD    DD     DD EE       RR    RR  ##
##   888888  ==== BBBBBBB     II        TT          AA    AA       DD     DD    DD     DD EEEEEE   RRRRRR    ##
##  88    88      BB    BB    II        TT         AAAAAAAAAA      DD     DD    DD     DD EE       RR  RR    ##
##  88    88      BB    BB    II        TT        AA        AA     DD    DD     DD    DD  EE       RR    RR  ##
##   888888       BBBBBBB  IIIIIIII     TT       AA          AA DDDDDDDD    DDDDDDDDD     EEEEEEEE RR     RR ##
##                                                                                                           ##
 ######################################## ~.~ By Susan Shrestha ~.~ ##########################################

*-------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*-------------------------------------------------------*
d
Enter the first decimal number:12
Enter the second decimal number: 8
*--------------------------*
* Adding in decimal method *
*--------------------------*
12
8
+
20


*-------------------------*
| Adding in binary method |
*-------------------------*
0 0 0 0 1 1 0 0
0 0 0 0 1 0 0 0
+
0 0 0 1 0 1 0 0

*-------------------------------------*
| Type 'c' to Continue or 'e' to Exit: |
*-------------------------------------*
c
```



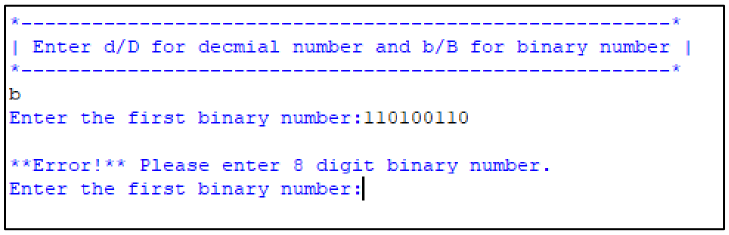
```
*--------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*--------------------------------------------------------*
b
Enter the first binary number:1011
Enter the second binary number:100


*-------------------------*
| Adding in binary method |
*-------------------------*
0 0 0 0 1 0 1 1
0 0 0 0 0 1 0 0
+
0 0 0 0 1 1 1 1

*--------------------------*
| Adding in decimal method |
*--------------------------*
11
4
+
15


*-------------------------------------*
| Type 'c' to Continue or 'e' to Exit: |
*-------------------------------------*
e
>>> |
```

Figure 15: Test 1

Test 2:

Objective	To test how the program responds when more than 8-bit is entered in binary number addition
Action	Enter the 1st binary number as 110100110
Expected Result	Error message should be printed as “enter 8 digit binary number”
Actual Result	“**Error!** Please enter 8 digit binary number.” is printed
Conclusion	Test succeed

Table 4: Test 2



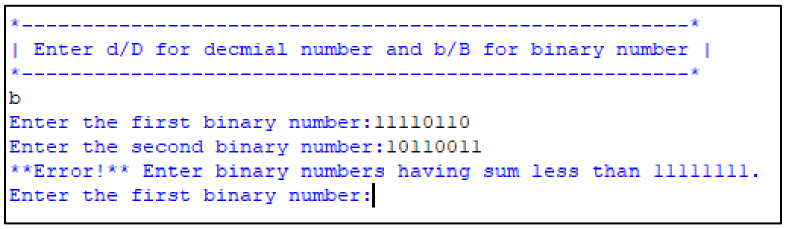
```
*--------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*--------------------------------------------------------*
b
Enter the first binary number:110100110

**Error!** Please enter 8 digit binary number.
Enter the first binary number:|
```

Figure 16: Test 2

Test 3:

Objective	To test how the program responds when trying to add two binary numbers that has sum more than 11111111 i.e. sum more than max value (255 in decimal)
Action	Enter the 1st binary number as 11110110, Enter the 2nd binary number as 10110011
Expected Result	Error message should be printed as "Enter binary numbers having sum less than 11111111"
Actual Result	"**Error!** Enter binary numbers having sum less than 11111111." is printed
Conclusion	Test succeed

Table 5: Test 3



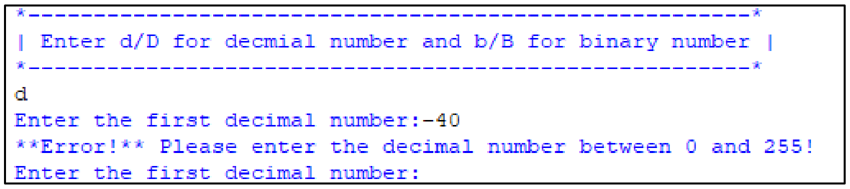
```
*-------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*-------------------------------------------------------*
b
Enter the first binary number:11110110
Enter the second binary number:10110011
**Error!** Enter binary numbers having sum less than 11111111.
Enter the first binary number:
```

Figure 17: Test 3

Test 4:

Objective	To test how the program responds when entering negative values in decimal number mode
Action	Enter the 1st binary number as -40
Expected Result	Error message should be printed as "enter the decimal number between 0 and 255"
Actual Result	"**Error!** Please enter the decimal number between 0 and 255!" is printed
Conclusion	Test succeed

Table 6: Test 4



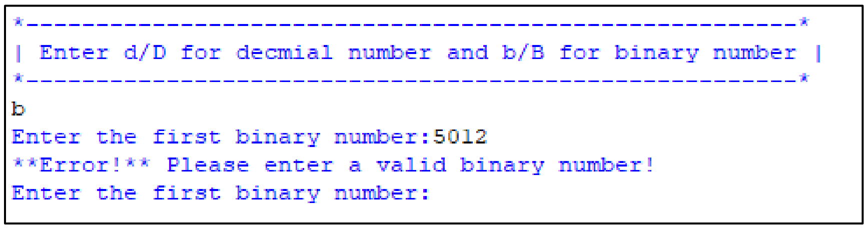
```
*-----------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*-----------------------------------------------------------*
d
Enter the first decimal number:-40
**Error!** Please enter the decimal number between 0 and 255!
Enter the first decimal number:
```

Figure 18: Test 4

Test 5:

Objective	To test how the program responds when entering decimal values in binary number mode
Action	Enter the 1st binary number as 5012
Expected Result	Error message should be printed as "Please enter a valid binary number"
Actual Result	"**Error!** Please enter a valid binary number!" is printed
Conclusion	Test succeed

Table 7: Test 5



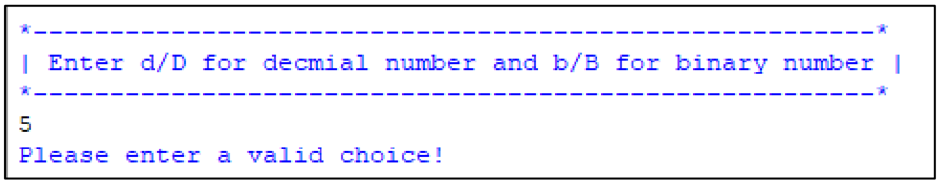
```
*-----------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*-----------------------------------------------------------*
b
Enter the first binary number:5012
**Error!** Please enter a valid binary number!
Enter the first binary number:
```

Figure 19: Test 5

Test 6:

Objective	To test how the program responds when entering random integer value for choice
Action	Enter the choice as 5 instead of b/d
Expected Result	Error message should be printed as “Please enter a valid choice”
Actual Result	“**Error!** Please enter a valid choice!” is printed
Conclusion	Test succeed

Table 8: Test 6



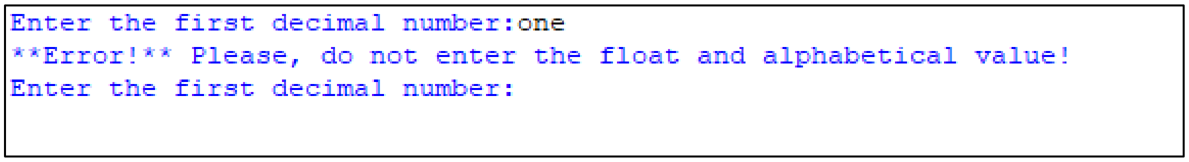
```
*-------------------------------------------------------*
| Enter d/D for decmial number and b/B for binary number |
*-------------------------------------------------------*
5
Please enter a valid choice!
```

Figure 20: Test 6

Test 7:

Objective	To test how the program responds when entering string for integer data type as input
Action	Enter the 1st decimal number input as one
Expected Result	Error message should be printed as “do not enter float or alphabet value”
Actual Result	“**Error!** Please, do not enter the float and alphabetical value!” is printed
Conclusion	Test succeed

Table 9: Test 7



```
Enter the first decimal number:one
**Error!** Please, do not enter the float and alphabetical value!
Enter the first decimal number:
```

Figure 21: Test 7

Conclusion

To conclude this report, this coursework was all about addition of two binary numbers and addition of their decimal values. Doing this coursework, I got a lot of things to learn about python programming, programming elements and techniques to code in python.

As this coursework was based on bit adder constructed using logical gates, I had to design a 8-bit (byte) adder in order to perform addition of two different binary numbers that results 8-bit number having the maximum value up 11111111. For designing the full adder model, I used logic.ly which basically is a circuit designing tool. As my 8-bit adder model got successfully designed, I wrote the algorithm for my program on how to add two binary numbers and their decimal values too. With the help of that algorithm, I developed a flowchart that depicts the overall flow of my 8-bit adder program including branches, decisions, processes and results. After the completion of flowchart, I moved on writing the pseudocode for each module of the program including each method in all modules.

After completion of model, algorithm, flowchart and pseudocode the next move was to choose the data structures in the program which should be suitable and efficient for the program. In this, I did some research on data structures in python, from that I found strings and lists the best choice required for my program which I later implemented during coding phase.

Finally, the time to write the code was arrived. Coding it was the most difficult task in the whole coursework because I faced a lot of errors and mistakes in my code which I later fixed with the help of tutorial videos and teacher's guidance. From the coding part, I learned a lot of things such as debugging, indentation, good programming style, documenting the code, etc. During the whole coding phase, the most interesting part for me was implementing the designated logic circuit bit adder using the logical gates. Also, I learned the proper use of loops and branching statements in python programming and the proper use of available data structures.

The last and the final portion of my report or documentation was testing which I personally enjoyed. As my program code was fully ready, I started the execution of the program and tested the program using several test cases such as entering the wrong data, providing maximum values as input, workflow of program on normal data and so on.

After testing part, my report is finally ready to submit. I am very humble and thankful to my respected teacher who guided all students including me during the coursework. I feel, I developed some knowledge and skills by completing this particular coursework and I hope I would be able to use my skills in future days.

Bibliography

Black, P. E. (2004, December 15). Retrieved september 2020, from https://en.wikipedia.org/wiki/Data_structure#:~:text=In%20computer%20science%2C%20a%20data,be%20applied%20to%20the%20data.

Cokelaer, T. (n.d.). *Data structure*. Retrieved from Thomas-Cokelaer: http://thomas-cokelaer.info/tutorials/python/data_structures.html#:~:text=There%20are%20quite%20a%20few,contain%20any%20type%20of%20objects.

Flowchart symbol and notations. (n.d.). Retrieved from lucidchart: https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained

programiz. (n.d.). *Python Strings*. Retrieved from programiz: https://www.programiz.com/python-programming/string

Python Tutorial. (n.d.). Retrieved 9 13, 2020, from www.tutorialspoint.com: https://www.tutorialspoint.com/python/index.htm

Python Tutorial. (n.d.). Retrieved from tutorialspoint: https://www.tutorialspoint.com/python/index.htm

w3schools. (n.d.). *Python Lists*. Retrieved from w3schools: https://www.w3schools.com/python/python_lists.asp

What is a Flowchart. (n.d.). Retrieved from Lucid chart: https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial#section_1

What is Pseudocode. (n.d.). Retrieved from indiatimes: https://economictimes.indiatimes.com/definition/pseudocode

Appendix

1. Main() module

```
#importing files for accessing methods and values in main file

from getInput import*

from adder import*



#main function

def main():

    #decoration

    print(" ################################################ ~~~Python~~~
###################################################")

    print("##                                                                    ##")

    print("##   888888      BBBBBBB  IIIIIII TTTTTTTTTT        AA       DDDDDDDD
DDDDDDDDD     EEEEEEEE RRRRRRR   ##")

    print("##  88    88     BB    BB    II       TT          AAAA        DD    DD     DD    DD
EE       RR    RR  ##")

    print("##  88    88     BB    BB    II       TT         AA  AA       DD     DD    DD     DD
EE       RR    RR  ##")

    print("##   888888  ==== BBBBBBB     II       TT         AA    AA       DD     DD    DD
DD EEEEEE   RRRRRR    ##")

    print("##  88    88     BB    BB    II       TT       AAAAAAAAAA      DD     DD    DD
```

```
DD EE       RR  RR    ##")

    print("##  88    88      BB    BB    II       TT        AA        AA     DD    DD     DD    DD
EE       RR    RR  ##")

    print("##   888888       BBBBBBB  IIIIIII     TT       AA         AA DDDDDDDD
DDDDDDDDD     EEEEEEEE RR     RR ##")

    print("##                                                                            ##")

    print(" ########################################## ~.~ By Susan Shrestha ~.~
#########################################")

    print("")



    flag=False

    while(flag==False):

        #getting choice from user

        print("*-----------------------------------------------------*")

        choice = input("| Enter d/D for decmial number and b/B for binary number |\n*--------
---------------------------------------------*\n")

        if(choice.lower() == "d"):

            decimalMethod()

        elif(choice.lower() == "b"):

            binaryMethod()

        else:

            print("Please enter a valid choice!")
            continue

        correctInput = False

        while(correctInput == False):

            print("\n")

            print("*------------------------------------*")

            exitprogram = input("| Type 'c' to Continue or 'e' to Exit: |\n*---------------------------
---------*\n")

            if(exitprogram.lower()=="e"):

                flag = True

                correctInput = True

            elif(exitprogram.lower() == "c"):

                flag = False

                correctInput = True

            else:

                print("Please enter a valid choice!")


#for decimal addition

def decimalMethod():

    binaryNumberList = []

    decimalNumberList = getInput("d")

    firstDecimalNumber = decimalNumberList[0]
    secondDecimalNumber = decimalNumberList[1]

    print("*--------------------------*")

    print("* Adding in decimal method *")

    print("*--------------------------*")

    print(firstDecimalNumber)

    print(secondDecimalNumber)

    pSum = firstDecimalNumber + secondDecimalNumber

    print("+")

    print(pSum)

    firstEightBitBinaryNumber = convertToBinary(firstDecimalNumber)

    secondEightBitBinaryNumber = convertToBinary(secondDecimalNumber)

    binaryNumberList.append(firstEightBitBinaryNumber)

    binaryNumberList.append(secondEightBitBinaryNumber)

    binarySum = adder(binaryNumberList)

    print("+ ")

    for i in binarySum:

        print(i ,end = " ")


#for binary addition

def binaryMethod():
```

```
binaryNumberList = []

binaryNumberList = getInput("b")

binaryNumber1 = binaryNumberList[0]

binaryNumber2 = binaryNumberList[1]

firstEightBitBinaryNumber = eightBit(binaryNumber1)

secondEightBitBinaryNumber = eightBit(binaryNumber2)

binaryNumberList.append(firstEightBitBinaryNumber)

binaryNumberList.append(secondEightBitBinaryNumber)

binarySum = adder(binaryNumberList)

print("+ ")

for i in binarySum:

    print(  i, end = " ")

print("\n")

print("*--------------------------*")

print("| Adding in decimal method |")

print("*--------------------------*")

firstDecimalNumber = convertToDecimal(binaryNumber1)

secondDecimalNumber = convertToDecimal(binaryNumber2)

print(firstDecimalNumber)

print(secondDecimalNumber)
```

```
    pSum=firstDecimalNumber+secondDecimalNumber

    print("+")

    print(pSum)


#execution of main() function

main()
```

2. getInput() module

```
from conversion import*

def getInput(choice):

    inputData = []


    #for Decimal Input

    if(choice.lower() == "d"):

        validDecimalInput = False

        while(not validDecimalInput):

            firstNumber =  input("Enter the first decimal number:")

            if(isValidDecimalInput(firstNumber)):

                if(int(firstNumber) < 0 or int(firstNumber) > 255):

                    print("**Error!** Please enter the decimal number between 0 and 255!")
                    continue
            else:
                continue
            secondNumber =  input("Enter the second decimal number:")
            if(isValidDecimalInput(secondNumber)):
                if(int(secondNumber) < 0 or int(secondNumber) > 255):
                    print("**Error!** Please enter the decimal number between 0 and 255!")
                    continue
            else:
                continue
            input1 = int(firstNumber)
            input2 = int(secondNumber)
            if ((input1 + input2) > 255):
                print("**Error!** Enter decimal numbers having sum less than 255!")
            else:
                inputData.append(input1)
                inputData.append(input2)
                validDecimalInput = True
        return inputData
```

```
#for Binary Input
elif(choice.lower() == "b"):
    ValidBinaryInput = False
    while(ValidBinaryInput == False):
            number1 = input("Enter the first binary number:")
            if(isValidBinaryInput(number1)):
                if(convertToDecimal(number1) < 0 or convertToDecimal(number1) > 255):
                    print("\n**Error!** Please enter valid 8 digit binary number.")
                    continue
            else:
                print("Please enter a valid binary number!")
                continue
            number2 = input("Enter the second binary number:")
            if(isValidBinaryInput(number2) ):
                if(convertToDecimal(number2) < 0 or convertToDecimal(number2) > 255):
                    print("\n**Error!** Please enter valid 8 digit binary number:")
                    continue
            else:
                print("Please enter a valid binary number:")
                continue
            if(convertToDecimal(number1) + convertToDecimal(number2)) > 255:
                print("**Error!** Enter binary numbers having sum less than 11111111.")
            else:
                inputData.append(split(number1))
                inputData.append(split(number2))
                ValidBinaryInput = True
        return inputData


#split function
def split(word):
    return[int(char) for char in word]


#checking valid decimal input
def isValidDecimalInput(number):
    try:
        int(number)
        return True
    except:
        if(number == ""):
            print("**Error!** You have not entered any input")
            return False
        else:
            print("**Error!** Please, do not enter the float and alphabetical value!")
            return False


#checking valid binary input
def isValidBinaryInput(input):
    try:
        for i in input:
            if(i not in ["0","1"]):
                print("**Error!** Please enter a valid binary number!")
                return False
    except:
        if(number == ""):
            print("**Error!** Do not enter empty field!")
            return False
        else:
            print("**Error!** Please, do not enter the float and alphabetical value!")
            return False
    return True
```

3. conversion() module

```
#conversion

def convertToDecimal(number):

    decimalNumber = 0

    i = 0

    for j in range(len(number)-1,-1,-1):

        decimalNumber += int(number[j]) * (2**i)

        i += 1

    return decimalNumber


def convertToBinary(number):

    reversedBinaryList = []

    binaryList = []

    if(number == 0):

        binaryList.append(0)

    else:

        while(number != 0):

            r = number % 2

            reversedBinaryList.append(r)
            number //= 2
    for i in range(len(reversedBinaryList)-1,-1,-1):
        binaryList.append(reversedBinaryList[i])
    eightBitBinaryList = eightBit(binaryList)
    return eightBitBinaryList


def eightBit(binaryList):
    if(len(binaryList)!= 8):
        for i in range(len(binaryList),8):
            binaryList.insert(0,0)
    return binaryList
```

4. adder() module

```
#adder file
from conversion import*
from gates import*


#8-bit adder model implementation
def adder(binaryList):
    binaryNumber1 = binaryList[0]
```

```
binaryNumber2 = binaryList[1]

carry = 0

reverseSum = []

binarySum = []

print("\n")

print("*-------------------------*")

print("| Adding in binary method |")

print("*-------------------------*")


for i in binaryNumber1:

    print(i, end = " ")

print()


for i in binaryNumber2:

    print(i, end = " ")

print()


for i in range(len(binaryNumber1)-1,-1,-1):

    bit1 = int(binaryNumber1[i])

    bit2 = int(binaryNumber2[i])
```

```
        #sum operation
        XOR = XOR_gate(bit1,bit2)
        OR = OR_gate(XOR,carry)
        NAND = NAND_gate(XOR,carry)
        Sum = AND_gate(OR,NAND)

        #carry operation
        AND1 = AND_gate(bit1,bit2)
        AND2 = AND_gate(XOR,carry)
        carry = NOT_gate(NOR_gate(AND1,AND2))

        reverseSum.append(Sum)

    for i in range(len(reverseSum)-1,-1,-1):
        binarySum.append(reverseSum[i])
    return binarySum
```

5. **gates() module**

```
#gates functions creation using bitwise operators
```

```
def AND_gate(a,b):

    return a & b


def OR_gate(a,b):

    return a | b


def XOR_gate(a,b):

    return a ^ b


def NOT_gate(a):

    return (~a) + 2


def NAND_gate(a,b):

    return NOT_gate(AND_gate(a,b))


def NOR_gate(a,b):

    return NOT_gate(OR_gate(a,b))
```