LONDON METROPOLITAN UNIVERSITY

Module Code & Module Title

CS4051NT Fundamental of Computing

Assessment Weightage &Type

100% Individual Coursework

Year and Semester

2019-20 Spring

Student Name: Alisha Shrestha

London Met ID: 19033571

College ID: np05cp4s200028

Assignment Due Date: 20 September, 2020

Assignment Submission Date: 20 September, 2020

Word Count (Where Required): 6467

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware thatlate submissions will be treated as non-submission and a marks of zero will be awarded.

Proposal

This proposal is written to address about the coursework of Fundamental of Computing Module which was assigned to us as an individual task. This coursework is all about developing a software application with python which demonstrates the integer addition based on binary operation. This coursework was provided on 3rd August and must be submitted on 20th September.

- **Purpose**

  The purpose of this coursework is to write algorithm for the program in python which performs the integer addition based on binary operation. Suitable data structure is selected with the preparation of flow chart of the program. This task also includes writing the pseudocode of the program. The construction of model of the byte adder using electronic gates based on bit adder is included in the coursework.

- **Targeted Audience**

  This program can be useful for the students, teachers, and programmers and for those who are interested in learning python programming language. This is for those who want to get the knowledge about how computer carries out mathematical operation.

- **Hardware and Software Requirements**

  The software that is required to run the program was python 3 where as there is no specific requirements for hardware part for the completion of this task. Draw.io was used for construction logic gate and designing the flow chart. Logic.ly was used to design the model circuit of the 8 bit adder.

Table of Contents

Table of figure

Table of table

1. Introduction

Python is a high-level programming language designed to be easy to read and simple to implement as it is an open source, which means it is free to use, even for commercial applications. Scripts written in Python can be parsed and run immediately as well as they can also be saved as complied programs. (techterms)

Python is easy, simple to learn syntax and reduces the cost of program maintenance. Python encourages program modularity and code to be reused as it supports modules and packages. Python can run on every system like Mac, Window and UNIX. (ThePSF)

Some of the features of python programming language are:

- **Easy to code:**

  Python is very easy to learn the language as compared to other languages such as C, C#, Java etc. It is very easy to code in python language and anybody can learn python basic in a few hours or mostly a day.

- **Free and open source:**

  Python language is freely available at the official website and can be downloaded through the link provided in the page. The source code is also available to the public as it is an open source.

- **Object-oriented language:**

  Like Java, Python is also Object-Oriented Programming language and it is the key feature of Python. Python supports Object-Oriented language and conceps of classes, etc.

- **High-Level Language:**

  Python is a high-level language. When the programs are written in python there is no need to remember the system architectures or the management of memory.

- **Interpreted language:**

  Python is an interpreted language as Python code is executed line by line at a time similar to other programming languages like C, C++, Java, etc. The Python code does not require to be compiled so that makes it much easier to debug the codes. The source code is converted into an immediate form called byte code. (GeeksforGeeks, 2020)

The history of the Python programming language dates back to the late 1980s and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands. Python's name is derived from the television series Monty Python's Flying Circus. Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of January 2016, it is in the fifth position. It was ranked as Programming Language of the Year for the year 2007 and 2010. It is the third most popular language whose grammatical syntax is not predominantly based on C.

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. (Tulchak, 2016)

1.1 Introduction to Project

This is the report of the coursework that has been provided as a task of developing a program in Python programming language that mainly aims to convert binary number into decimal and decimal into binary. The function of this program is that it computes the sum of two integer's number based on binary operation and byte adder model is used for the process of addition. Moreover algorithm, pseudocode and flowcharts were made for developing this program. This project was about how the computer performs the addition operation of two integer number using 0 and 1. The process of addition was portrayed by the construction of byte adder model based on combination of eight bit adders with the help of logic gates. The flowchart was developed for demonstrating the flow and steps in the program. The task was performed in a very productive way within the time limit.

1.2 Aims

The aims of this project are:

i. To carry out the entire task in the best possible was.
ii. To accomplish the task of converting binary to decimal and decimal to binary in a simple way possible.
iii. Plenty of research regarding process through which the computer performs addition of the integer numbers.
iv. Acquiring the knowledge of algorithm and its uses in other projects.
v. Gaining experience and skills of model circuit to implement.
vi. Learning the actual mechanism behind the computer performing the process for conversion.

1.3 Objective

The major objectives of this project are:

i. Construction of byte adder model based on the bit adder that would describe the process addition.
ii. Writing the algorithm and pseudocode of the program that performs the operation.
iii. Constructing a flowchart to show the arrangements of the steps for the completion of the program.
iv. Writing a good program with the help of algorithm, pseudocode and flowchart for the conversion.
v. Obtaining an accurate and versatile program by undergoing the chain of testing the program.

2. Model of Circuit Diagram

A model circuit diagram represents the actual electrical connections along with the arrangements of wire, bulbs and other components.

2.1 Working mechanism

To perform the task different logical gate is implemented that are AND gate, OR gate, XOR gate, NOT gate, NAND gate and XOR gate. In the logical get three inputs is taken which are upper bit, lower bit and Carry-in. The three inputs are passed to form one full bit adder in the same manner other seven full bit adder are also produced to make one full byte adder.

More precisely, looking into the functionality in detail of one full adder, there are three inputs upper bit, lower bit and Carry-in. The upper bit value and lower bit value is passed through XOR gate and AND gate(first AND gate), XOR gate and Carry-in value is passed through next AND gate (second AND gate). The first and second AND gate value is pass through NOR gate and the value obtain from that is again passed through NOT gate which produce first Carry-out of the full adder.

Here XOR gate value and Carry-in value is passed through NAND gate and OR gate. The value obtained from NAND and OR gate is again passed through AND gate which give the final sum of full adder.

With the whole procedure of full adder two values is obtained i.e. Carry-out and sum. The obtained value of Carry-out will be the Carry-in for the next adder likewise again three values i.e upper bit, lower bit and Carry-in will pass through full adder.

Similarly, the whole process is repeated for 8 times to form byte adder and after performing all the process a value is obtained that is considered as the sum which is the output produced from the logical gate of upper bit and lower bit.

For better understanding, two value is taken that are 77 and 127 but in logic gate the value in the form of 0s and 1s is only accepted so 77 and 127 is converted into binary form.

Table 1 Conversion of decimal into binary

Decimal number	Binary number
77	01001101
127	01111111
Total= 204	11001100

The above table demonstrates the conversion of decimal value to binary value. The last bit of the first binary number is considered as first upper bit i.e. 1, the last bit of the second given binary number is considered as first lower bit i.e. 1 and initially the first Carry-in value is 0 likewise the three value are passed into full adder similarly each values from the last is added in the full adder until it makes byte adder. The sum obtained from byte adder will be the output of two binary numbers.

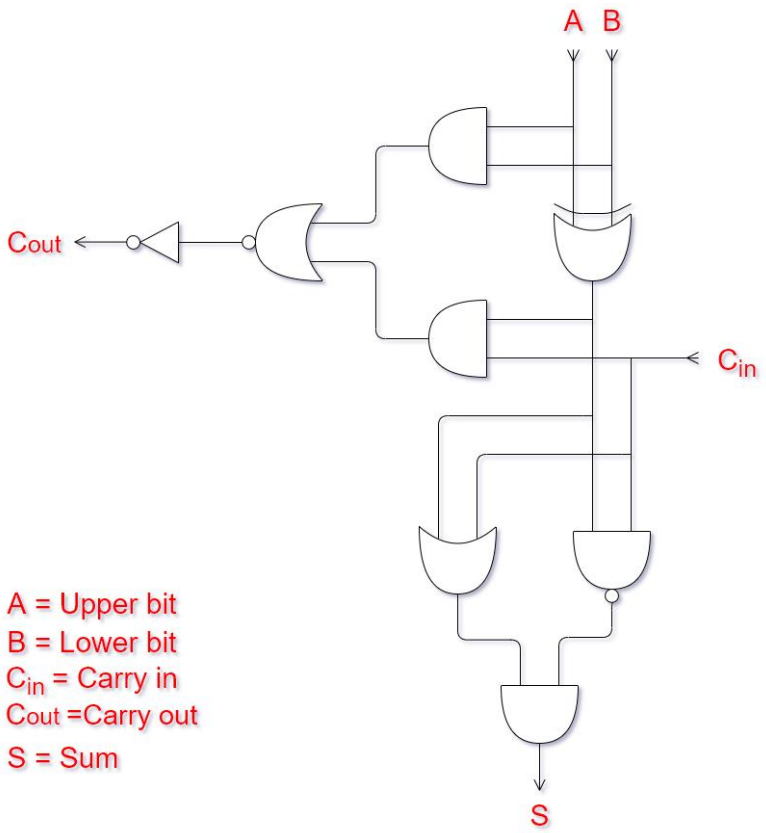

Figure 1 Full adder

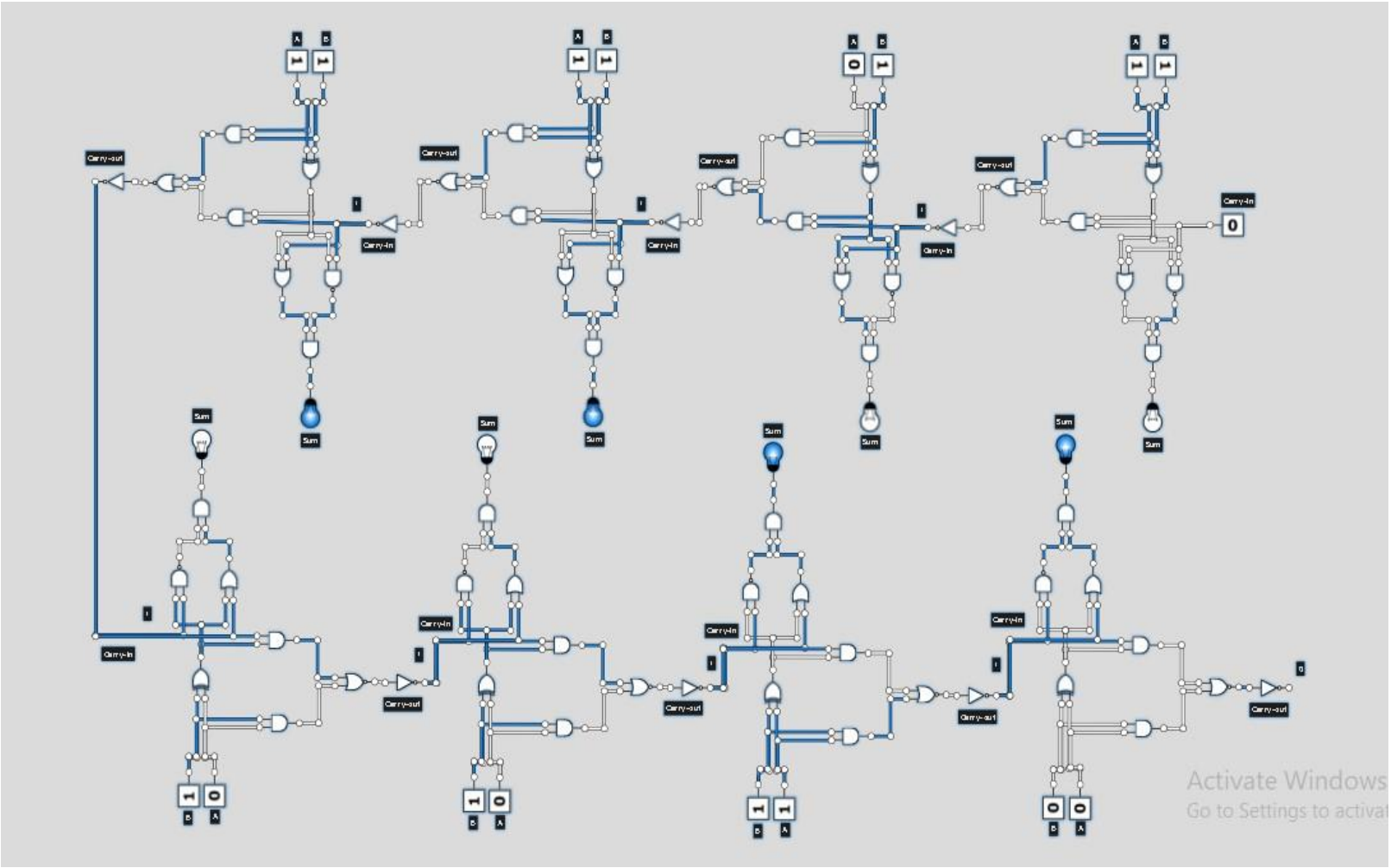

Figure 2 Circuit diagram of 8 bit adder

The above figure illustrates the circuit diagram of 8 bit adder which is created on the basis of full adder. The circuit diagram not only illustrates the process of addition but it also shows the working procedure of full adder to give the sum. The full adder was produced with the help of the combination of different logic gates. Here in this circuit diagram the input provided to the full adder is denoted by rectangular box and sum obtained from each adder is denoted by bulb. Bulb with blue and white colour indicates 1 and 0 respectively.

2.2 Truth table of byte adder

Table 2 *Truth table of byte adder*

A	**B**	**C_{in}**	**A** xor **B** **(xor)**	**A** and **B** **(and1)**	**xor** and **C_{in}** **(and2)**	**and1** nor **and2** **(nor)**	**nor** not **(C_{OUT})**	**xor** nand **C_{in}** **(nand)**	**xor** or **C_{in}** **(or)**	**xor** and **nand** (Sum)
1	1	0	0	1	0	0	1	1	0	0
0	1	1	1	0	1	0	1	0	1	0
1	1	1	0	1	0	0	1	1	1	1
1	1	1	0	1	0	0	1	1	1	1
0	1	1	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	0	1	0
1	1	1	0	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	1	1	1

2.3 Parallel Circuit Diagram

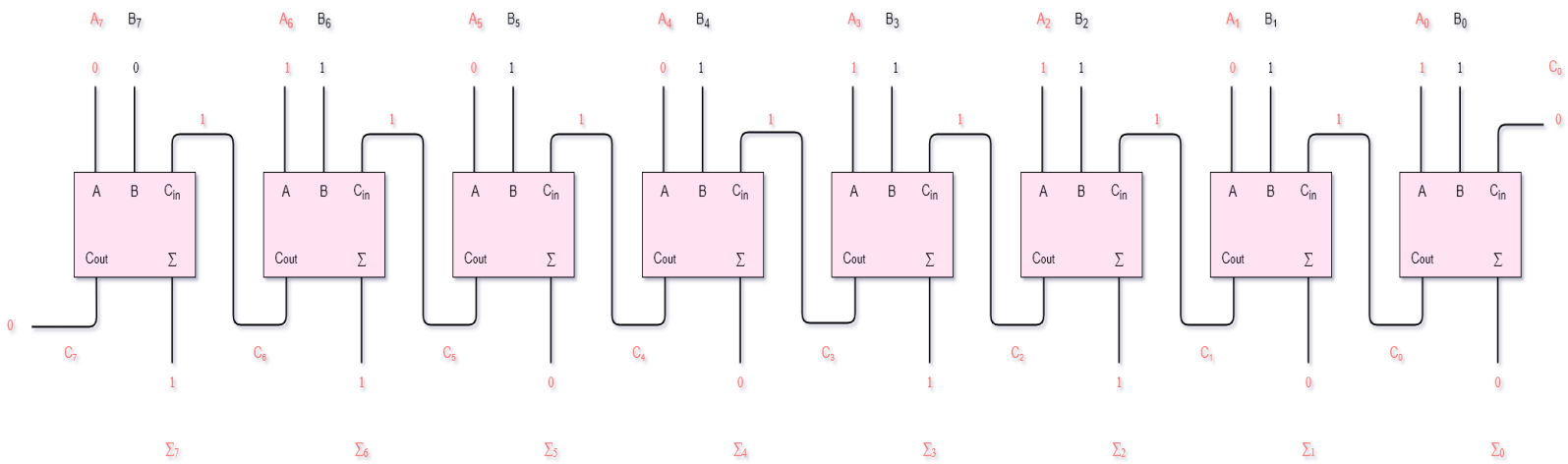

Figure 3 Eight bit parallel circuit diagram

The above figure demonstrates the eight bit parallel circuit diagram. Here in this circuit diagram two binary values are passes for producing the sum of both the value. 77 and 127 is passed in the circuit diagram.

2.4 Logic gates

- **AND gate:** The AND gate is an electronic circuit that gives output 1 only if all its inputs are high. A dot is used to show the AND operation i.e. (A.B)

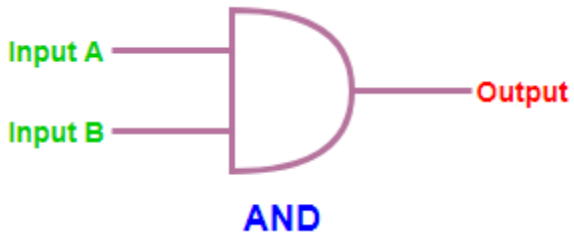

Figure 4 Circuit diagram of ANG gate

Table 3 Truth table of AND gate

Input		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

- **OR Gate:** The OR gate is an electronic circuit that produces a high output if one or more of its inputs are high. A plus is used to denote the OR operation i.e. (A+B).

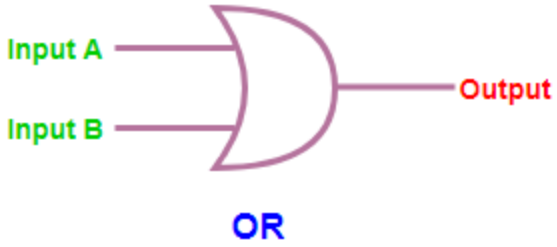

Figure 5 Circuit diagram of Or gate

Table 4 Truth table of OR gate

Input		Output
A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

- **XOR Gate:** The XOR gate is a circuit which gives high output when the number of two inputs is odd. An encircled plus sign is used to represent XOR operation.

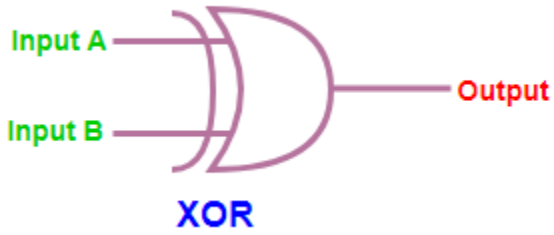

Figure 6 Circuit diagram of XOR gate

Table 5 Truth table of XOR gate

Input		Output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

- **NOR Gate:** The NOR gate is an electronic circuit that produces a high output if both inputs are low or else all low.

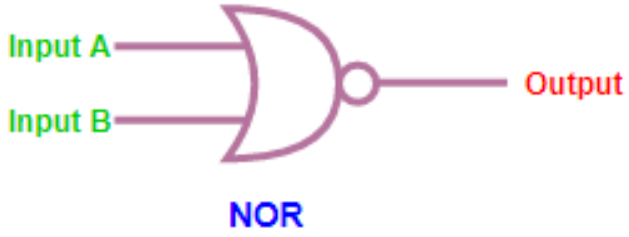

Figure 7 Circuit diagram of NOR gate

Table 6 Truth table of NOR gate

Input		**Output**
A	**B**	**A NOR B**
0	0	1
0	1	0
1	0	0
1	1	0

- **NOT gate:** The NOT gate is an electronic circuit that where inputs are high then the output is low and if the input is low than the output is high.

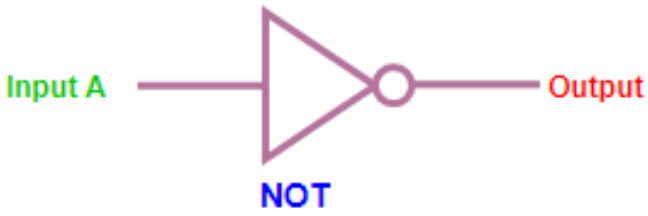

Figure 8 Circuit diagram of NOT gate

Table 7 Truth table of NOT gate

Input	Output
A	**A NOT B**
0	1
1	0

- **NAND gate:** The NAND gate is an electronic circuit that produces a low output if both inputs are high or else the output is high.

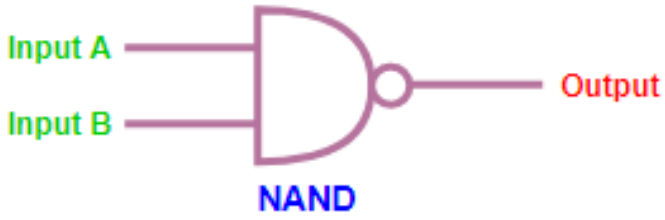

Figure 9 Circuit diagram of NAND gate

Table 8 Truth table of NAND gate

Input		Output
A	**B**	**A NAND B**
0	0	1
0	1	1
1	0	1
1	1	0

3. Algorithm

An algorithm is a set of well-defined instructions in step by step manner to solve a particular problem. In other word it is a set of rule which explains how a problem can be solved in steps. An algorithm should not include computer code and it must be written in such a way that it can also be useful in different programming languages. An algorithm must be composed of a finite set of steps where each may require one or more operation. (Kullabs, 2019)

The good algorithm should have the following features:

- Input: It must specify and requires input values.
- Output: It must determine outcome values or solution of the problem.
- Definite: It must define each process and the progress clearly.
- Effective: It must be more effective among many different ways to solve a problem.
- Finite: It must consist of finite number of steps and operations. (laps)

Example of algorithms in programming while adding two numbers entered by the user:

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2

Step 4: Add num1 and num2

Step 5: Display sum

Step 6: Stop (Kullabs, 2019)

Writing the algorithm for the program:

Step 1: Start

Step 2: Ask user to input binary or decimal number

Step 3: If user input binary number

Step 3.1: Ask first binary number

Step 3.2: If first input is less than 0 or more than 255 repeat Step 3.1

Step 3.3: Ask second binary number

Step 3.4: If second input is less than 0 or more than 255 repeat Step 3.3

Step 3.5: Add two binary numbers

Step 3.6: If the output of two binary numbers is less than 0 or more than 255 repeat Step 3.1

Step 3.7: Convert binary into decimal number

Step 3.8: Display the output of both binary and decimal number

Step 4: If user input decimal number

Step 4.1: Ask first decimal number

Step 4.2: If first input is less than 0 or more than 255 repeat Step 4.1

Step 4.3: Ask second decimal number

Step 4.4: If second input is less than 0 or more than 255 repeat Step 4.3

Step 4.5: Add two decimal numbers

Step 4.6: If the output of two binary numbers is less than 0 or more than 255 repeat 4.1

Step 4.7: Convert decimal into binary number

Step 4.8: Display the output of both decimal and binary number

Step 5: Ask user to continue program or not

Step 5.1: If user want to continue program then repeat from step 2

Step 5.2: if user doesn’t want to continue than it goes to step 6

Step 6: Stop

4. Pseudocode

Pseudocode is an informal manner of writing the description of a program with no strict syntax like in programming language. Pseudocode is not considered as an actual programming language because it does not get complied into an executable program. It is very simple and easy to write because it uses short terms and minor English language syntaxes to write the code for the programs before the code is converted into any specific program. Pesudocode holds detailed and readable information about the program so that make easier for the process of developing a program. Pseudocode provides a good summary on the flow of the program and algorithm of the program in a very effortless manner. (Rouse, 2005)

Pseudocode is helpful for the programmer because catching errors and wrong program flow at the pseudocode stage becomes less costly than catching them later in the development process. It is also possible to write programs that will convert a given pseudocode language into a given programming language. Pseudocode enables the programmer to concentrate only on the algorithm part of the code development as it is used in planning an algorithm with sketching out the structure of the program before the actual coding takes place. The program description and the functions are collected and then the pseudocode is used to create statements to achieve the required results for a program. (The Economic Times, 2020)

4.1 Pseudocode of main.py module

IMPORTING asinput

FUNCTION main_function():

    CALLbinary_decimal_input()

    INITIALIZE check = False

    WHILE (not check):

    INITIALIZE exit_input = input("Do you want to continue [Yes/No] : ")

    IF (exit_input.lower() == "yes"):

CALLbinary_decimal_input()

ELIF (exit_input.lower() == "no"):

PRINT("Thank you for using this program.")

INITIALIZE check = True

ELSE:

PRINT ("Error !!! , Please enter Yes to run the program or No to exit program.")

CALL main_function()

END FUNCTION

4.2 Pseudocode of asinput.py module

IMPORTING forconversion

IMPORTING validation

IMPORTING foradder

FUNCTION binary_decimal_input():

INITIALIZE main = False

INITIALIZE input_check = False

INITIALIZE check_input_no1 = False

INITIALIZE check_input_no2 = False

WHILE (not main):

INITIALIZE input_option = input("Enter [B/b] for binary number and [D/d] for decimal number : ")

IF (input_option.lower() == "b"):

WHILE (not input_check):

WHILE (not check_input_no1):

INITIALIZE first_bin = input("Enter first binary number : ")

IF (binary_validation(first_bin)[0]):

PRINT (binary_validation(first_bin)[1])

ELSE:

INITIALIZE check_input_no1 = True

INITIALIZE n = 8 - len(first_bin)

INITIALIZE first_bin = ("0"*n+first_bin)

END IF

END WHILE

WHILE (not check_input_no2):

INITIALIZE second_bin = input("Enter second binary number : ")

IF(binary_validation(second_bin)[0

PRINT (binary_validation(second_bin)[1])

ELSE:

INITIALIZE check_input_no2 = True

```
        INITIALIZE n = 8 - len(second_bin)
        INITIALIZE second_bin = ("0"*n+second_bin)
            END IF
            END WHILE

            INITIALIZE a = bin_to_dec(int(first_bin))
        INITIALIZE b = bin_to_dec(int(second_bin))
        IF(a + b > 255):
            PRINT ("Error !!! Exceeded the limits of 11111111.")
            INITIALIZE check_input_no1 = False
            INITIALIZE check_input_no2 = False
        ELSE:
            INITIALIZE input_check =True
            INITIALIZE main = True
            END IF

INITIALIZE c = first_bin
INITIALIZE d = second_bin
PRINT (c.zfill(8))
PRINT (d.zfill(8))
PRINT (adder_gate(c,d))

INITIALIZE e = bin_to_dec(int(first_bin))
```

INITIALIZE f = bin_to_dec(int(second_bin))

PRINT (e)

PRINT (f)

PRINT (e + f)

END WHILE

ELIF (input_option.lower() == "d"):

WHILE (not input_check):

WHILE (not check_input_no1):

INITIALIZE first_dec = input("Enter first decimal number : ")

IF (decimal_validation(first_dec)[0]):

PRINT (decimal_validation(first_dec)[1])

ELSE:

INITIALIZE check_input_no1 = True

END IF

END WHILE

WHILE (not check_input_no2):

INITIALIZE second_dec = input("Enter second decimal number : ")

IF (decimal_validation(second_dec)[0]):

PRINT (decimal_validation(second_dec)[1])

ELSE:

INITIALIZE check_input_no2 = True

```
                    END IF
                END WHILE


            INITIALIZE a = int(first_dec)
            INITIALIZE b = int(second_dec)
            IF (a + b > 255):
                PRINT ("Error !!! Exceeded the limits of 255.")
                INITIALIZE check_input_no1 = False
                INITIALIZE check_input_no2 = False
            ELSE:
                INITIALIZE input_check = True
                INITIALIZE main = True
                END IF
            INITIALIZE c = int(first_dec)
        INITIALIZE d = int(second_dec)
        PRINT(c)
        PRINT(d)
PRINT(c + d)


        INITIALIZE e = dec_to_bin(int(first_dec))
        INITIALIZE f = dec_to_bin(int(second_dec))
        PRINT (e)
        PRINT (f)
```

```
            PRINT (adder_gate(e,f))
            END WHILE
            else:
            PRINT("Invalid Input.")
            PRINT("Enter Either [B/b] or [D/d].")
            END IF
        END WHILE
END FUNCTION
```

4.3 Pseudocode of gates.py module

```
FUNCTION and_gate(x, y):
        RETURNx&y
END FUNCTION

FUNCTION or_gate(x, y):
        RETURNx|y
END FUNCTION

FUNCTION xor_gate(x, y):
        RETURNx^y
END FUNCTION

FUNCTION not_gate(x):
```

RETURN (~x) + 2

END FUNCTION

FUNCTION nand_gate(x,y):

RETURNnot_gate(and_gate(x,y))

END FUNCTION

FUNCTION nor_gate(x,y):

RETURNnot_gate(or_gate(x,y))

END FUNCTION

4.4 Pseudocode of adder.py module

IMPORTING gates

FUNCTIONadder_gate(bin_no1,bin_no2):

INITIALIZE carry = 0

INITIALIZElist_of_eightbit = ["0","0","0","0","0","0","0","0"]

FORa in range (7,-1,-1):

INITIALIZEupper_bit =int(bin_no1[a])

INITIALIZElower_bit = int(bin_no2[a])

INITIALIZExor_ = xor_gate(upper_bit, lower_bit)

INITIALIZEnand_ = nand_gate(xor_,carry)

INITIALIZE or_ = or_gate(xor_,carry)

INITIALIZEsum_of_bit = and_gate(or_,nand_)

INITIALIZElist_of_eightbit[a] = str(sum_of_bit)

INITIALIZE and_no1 = and_gate(upper_bit, lower_bit)

INITIALIZE and_no2 = and_gate(xor_, carry)

INITIALIZE carry = not_gate(nor_gate(and_no1, and_no2))

RETURN "".join(list_of_eightbit)

END FOR

END FUNCTION

4.5 Pseudocode of conversion.py module

FUNCTIONbin_to_dec(value):

INITIALIZEdec_no = 0

INITIALIZE i = 0

WHILE (value > 0):

INITIALIZElast_no = value % 10

INITIALIZEdec_no += last_no * (2 ** i)

INITIALIZEi += 1

INITIALIZEvalue = int(value/10)

```
    RETURNdec_no
    END WHILE
END FUNCTION

FUNCTIONdec_to_bin(value):
    INITIALIZElist_of_eightbit = ["0","0","0","0","0","0","0","0"]
    INITIALIZEindex_ = 7
    WHILE(value > 0):
        INITIALIZEb = value % 2
        INITIALIZElist_of_eightbit[index_] = str(b)
        INITIALIZEindex_ -= 1
        INITIALIZEvalue = int(value/2)
    RETURN"".join(list_of_eightbit)
    END WHILE
END FUNCTION
```

4.6 Pseudocode of validation.py module

```
Functionbinary_validation(input_bin):
    IF (input_bin == ””):
        RETURN[True,”Error!!! Please enter the value”]
    END
    TRY:
        INITIALIZE check = int(input_bin)
```

```
        EXCEPT:
            RETURN[True,"Error !!! Accepts integer value only."]
        IF(int(input_bin) < 0):
            RETURN[True, "Please enter positive value."]
        ELSE:
            FOR a in input_bin:
                IF(int(a) not in [0,1]):
                    RETURN[True,"Invalid input!!! Enter either 0 or 1."]
                END IF
            END FOR
        END
        IF (len(input_bin)> 8):
            RETURN[True, "Invalid input!!! Enter 8 bit binary number"]
        RETURN[False]
        END FALSE
END FUNCTION


FUNCTIONdecimal_validation(input_dec):
        IFinput_dec == "":
            RETURN [True,"Error !!! Please enter the value."]
        END IF
        TRY:
            INITIALIZE check = int(input_dec)
```

```
        EXCEPT:
            RETURN [True, "Error !!! Accepts integer value only."]
        IFint(input_dec) < 0 or int(input_dec) > 255:
            RETURN [True, "Please enter value more than zero and less than 255"]
        RETURN [False]
        END IF
END FUNCTION
```

5. Flowchart

A flowchart is simply a graphical representation of step by step solution of a problem. Flowchart is symbols whereas algorithms and pseudocode are text-based method of designing programs. They help us visualize complex processes, or make explicit the structure of problems and tasks. It shows steps in sequential order and is widely used in presenting the flow of algorithms also it is the regarded as the oldest techniques to depict an algorithm and its workflow or processes. (Kullabs, 2019)

It was originated from computer science as a tool for representing algorithms and programming logic but had extended to use in all other kinds of processes. A flowchart also makes debugging process easier which can be a gift for programmers. A flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows. Different flowchart shapes have different conventional meanings. (Paradigm)

The basic flowchart symbols are designed and meanings of some of the more common shapes are as follows:

a) Terminator: This oval shaped symbol represents the starting or ending point of the system which are written in the beginning and end in flowchart.

Figure 10 Terminator

b) Input/output: A parallelogram indicates the function of input/output.

Figure 11 Input/ Output

c) Process: This rectangular box indicates arithmetic operations, processes and data manipulations.

Figure 12 Process

d) Decision making: A diamond represents a decision or branching point. Lines coming out from the diamond indicate different possible situations, leading to different sub-processes.

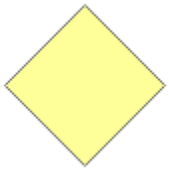

Figure 13 Decision making

e) Connector: A circle symbol in the flowchart is helpful to utilize the connector to stay away from any confusion.

Figure 14 Connector

f) Flow: Lines represent the flow of the sequence and direction of a process.

Figure 15 Flow

Example of a flowchart: Sum of 529 and 256

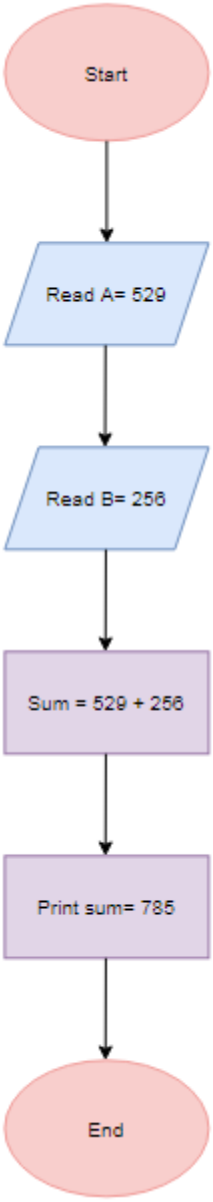

Figure 16 Example of Flowchart

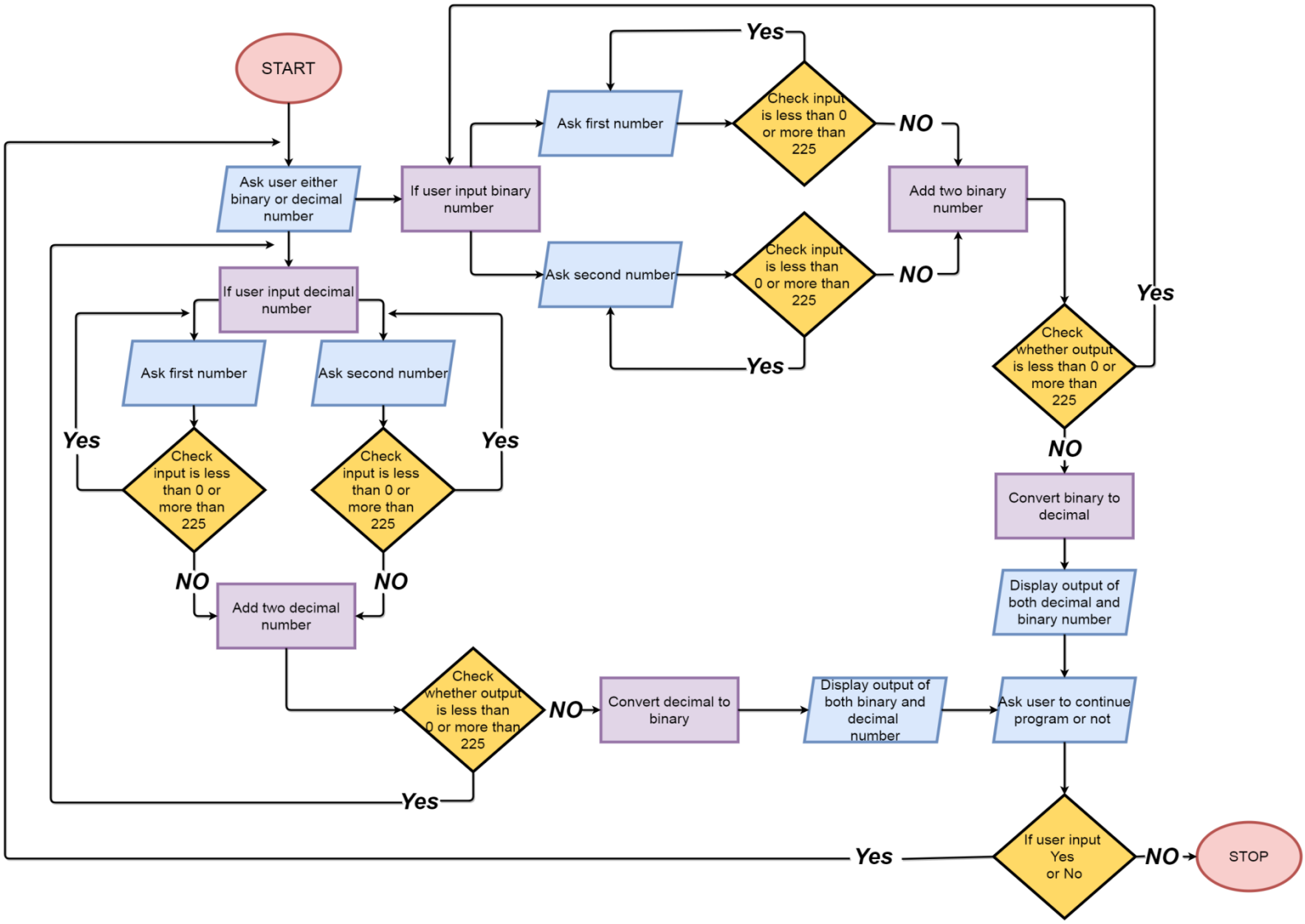

Figure 17 Flow chart of 8 bit adder

6. Data Structure

Python has been used worldwide for different fields but to make this possible data plays a very vital role where data should be stored efficiently so managing, organizing and storing data is important. Data structure makes easier access and efficient modifications of data as it enables to store collections of data, related and perform operations on those data accordingly. (StudyTonight)

Python allow to create own data structure with the functionality and it enables user to control the functionality. Python provides varieties of data types such as integer, string, Boolean etc. Python has a host of in-built data structures that help us to easily organize our data

In python there are two data structures i.e. Primitive data structure and Non- primitive data structure. (Akash, 2019)

6.1 Primitive data structures

Primitive data structure is the most basic data structure and they are the building blocks for the data manipulation. The types of primitive data structures used in this project are:

- **Integers:**

  Integer is a numeric data which holds whole numbers from negative to positive. Integer is also written as ‘int’ which does not have decimal points and this data type has no limit for the value as it holds infinite integer value until the system memory can hold. Any numeric value can be added as integer value. (Jaiswaly)

  For example:

  a = 5

  print(a “is a integer.”)

  Output:

  5 is an integer.

In this project integer data type is used to store the integer values. . The number input by the user is in string so it is converted into integer value in asinput.py module. The sum obtained from the addition of two numbers is used as integer value In **asinput.py**, **forconversion.py**, **validation.py** and **foradder.py** module integer data type is used for storing the integer value.

- **String:**

String is a sequence of one or more characters which is written inside single quotation mark or double quotation mark. It stores both constants or variables and it is enclosed with single or double quotation but only one type should be used throughout the program. In python the quotes represents the beginning and ending of the string. The string data type in python can be converted in to other data types by using various in-built function provided by python. String can hold as many characters as the users wants because it stores until the system memory can hold. (geeksforgeeks)

For example:

a = “This is Python.”

print(a)

Output:

This is Python.

In this project string data type is used in foradder.py module to add the sum of two binary numbers. In the main.py module string data type is used to obtain the decision from the user for either continuing the program or end the program. In asinput.py module string data type is used to obtain a single string value which is input by the user in variable for the purpose of storing the binary values of the string.

6.2 Non- primitive data structure

Non-primitive data structures are the complicated members of the data structure family.

It does not only store the value but its stores collection of values in various formats. The types of non-primitive data structures used in this project are:

- **List:**

  In python list is a well-organized sequence of items which is considered as the most used data type in python. It is declared by separating the items with commas and enclosing inside the square brackets. List can make changes in their content without changing the identity. Python has many methods to work with lists changes can also be made during the time of execution as per the requirements. Even the size of list can also be changed so overall list is considered as very flexible data structure. (Jaiswaly)
  For example:
  a = [1,2,3]
  print(a)
  Output:
  [1,2,3]
  In this project list was used in foradder.py module to store 8 bit binary number.

- **Dictionary:**

  Dictionary is an unordered collection of huge amount of data values which is used to store data values. Dictionary holds key: value pair as it is used to make it more optimized. In Python, a dictionary is declared by separating elements by commas and enclosed in curly braces. It holds a pair of values where one value is considered as the key and other as key: value. Any data type can be used in dictionary and the keys cannot be repeated or changed.
  Dictionary keys are case sensitive so the elements with same name but eith different cases are not considered same. (Programiz)

For example:

```
a= {“number” : 1, “Name” : Python}
print(a[number])
print(a[Name])
```

Output:

```
1
Python
```

7. Program

7.1 Main module

```
# main.py module controls all the module to complete the conversion of binary and decimal number
# It import the asInput module to perform user interaction.
# Auther: Alisha Shrestha, 10 September, 2020

from asinput import *

# Creating the function to make its user experience better
def main_function():
    # Making a design of 8 bit adder
    print( "_____~ Python ~ _____")
    print(":                                                                                :")
    print(":                                                                                :")
    print(":  /     \    ||     \    ||       ||          / \      |      \    |      \    |       |      \    :")
    print(": |       |   ||      |   ||       ||         /   \     |       \   |       \   |       |       |   :")
    print(": \       /   ||      /   ||       ||        /     \    |        \  |        \  |       |       |   :")
    print(":   ------     ||------    ||       ||       /_ ____\   |        )  |        )  |-------  | ___ /    :")
    print(":  /      \    ||     \    ||       ||      /       \   |       /   |       /   |       |     \    :")
    print(": |        |   ||      |   ||       ||     /         \  |      /    |      /    |       |      \   :")
    print(":  \___ /     ||_____/  __||__     ||    /           \ |_____/     |_____/     |_____  |       \  :")
    print(":                                                                                :")
    print(" _____~~Alisha Shrestha ~~_____")
    print()
    binary_decimal_input() # calling funciton
    check = False  # initializing False boolean value
    # while loop is created to check the condition untill user input correct value
    while(not check):
        print()
        # it ask user to input value
        exit_input = input("Do you want to continue [Yes/No] : ")
        print()
```

Figure 18 Screenshot no 1 of main.py module

```
        # lower() method is used to return the lowercased string
        # it converts all uppercase character to lower case
        if (exit_input.lower() == "yes"):
            binary_decimal_input()
        elif (exit_input.lower() == "no"):
            print()
            print("Thank you for using this program.")
            print()
            check = True
        else:
            print("Error !!! , Please enter Yes to run the program or No to exit program.")

main_function()
```

Figure 19 Screenshot no 2 of main.py module

7.2 Input.py module

```
# asinput.py module is created to ask user input and display the output of binary and decimal additon
# it imports forconversion.py, validaiton.py and foradder.py module to complete the program
# Auther: Alisha Shrestha, 10 September, 2020

from forconversion import *
from validation import *
from foradder import *

# Creating function to ask user input and it display the output of binary and decimal addition.
def binary_decimal_input():
    main = False
    input_check = False
    check_input_no1 = False
    check_input_no2 = False

    # while loop is created to check the condition untill user input correct value
    while (not main):
        # it ask user to input value
        input_option = input("Enter [B/b] for binary number and [D/d] for decimal number : ")
        print()
        # lower() method is used to return the lowercased string
        # it converts all uppercase character to lower case
        if (input_option.lower() == "b"):
            while (not input_check):
                while (not check_input_no1):
                    # it ask user to input value
                    first_bin = input("Enter first binary number : ")
                    if (binary_validation(first_bin)[0]): # [0] value check the index
                        print()
                        print(binary_validation(first_bin)[1]) # [1] value check the index
```

Figure 20 Screenshot no 1 of asinput.py module

```
            print()
        else:
            print()
            check_input_no1 = True
            n = 8 - len(first_bin) # check the length of input value
            first_bin = ("0"*n+first_bin)

    while (not check_input_no2):
        # it ask user to input value
        second_bin = input("Enter second binary number : ")
        if (binary_validation(second_bin)[0]):# [0] value check the index
            print()
            print(binary_validation(second_bin)[1])# [1] value check the index
            print()
        else:
            print()
            check_input_no2 = True
            n = 8 - len(second_bin) # check the length of input value
            second_bin = ("0"*n+second_bin)

    a = bin_to_dec(int(first_bin))
    b = bin_to_dec(int(second_bin))
    # it check the value whether it is not more than 255
    if (a + b > 255):
        print("Error !!! Exceeded the limits of 11111111.")
        check_input_no1 = False
        check_input_no2 = False
    else:
        input_check =True
        main = True
```

Figure 21 Screenshot no 2 of asinput.py module

```
        c = first_bin
        d = second_bin
        print("          Binary Addition")
        print("          --------------------")
        print("1st  no     ",c.zfill(8)) # z.fill() method is used to fill 0 value
        print("2nd no      ",d.zfill(8))
        print("          --------------------")
        print("Output      ",adder_gate(c,d)) # it add two binary number
        print()

        e = bin_to_dec(int(first_bin)) # it convert binary no into decimal no
        f = bin_to_dec(int(second_bin)) # it convert binary no into decimal no
        print("          Decimal Addition")
        print("          ----------------------")
        print("1st no            ",e)
        print("2nd no            ",f)
        print("          ----------------------")
        print("Output            ",e + f)

    # lower() method is used to return the lowercased string
    # it converts all uppercase character to lower case
    elif (input_option.lower() == "d"):
        # while loop is created to check the condition untill user input correct value
        while (not input_check):
            while (not check_input_no1):
                # it ask user to input value
                first_dec = input("Enter first decimal number : ")
                if (decimal_validation(first_dec)[0]): # [0] value check the index
                    print()
                    print(decimal_validation(first_dec)[1]) # [1] value check the index
```

Figure 22 Screenshot no 3 of asinput.py module

```
            print()
        else:
            print()
            check_input_no1 = True

    while (not check_input_no2):
        # it ask user to input value
        second_dec = input("Enter second decimal number : ")
        if (decimal_validation(second_dec)[0]): # [0] value check the index
            print()
            print(decimal_validation(second_dec)[1]) # [1] value check the index
            print()
        else:
            print()
            check_input_no2 = True

    a = int(first_dec)
    b = int(second_dec)
    if (a + b > 255):
        # it check the value whether it is and not more than 255
        print("Error !!! Exceeded the limits of 255.")
        check_input_no1 = False
        check_input_no2 = False
    else:
        input_check = True
        main = True

c = int(first_dec)
d = int(second_dec)
print("          Decimal Addition")
```

Figure 23 Screenshot no 4 for asinput.py module

```
print("          ----------------------")
print("1st no           ",c)
print("2nd no           ",d)
print("          ----------------------")
print("Output           ",c + d)
print()

e = dec_to_bin(int(first_dec)) # it convert decimal no into binary no
f = dec_to_bin(int(second_dec)) # it convert decimal no into binary no
print("          Binary Addition")
print("          --------------------")
print("1st  no     ",e)
print("2nd no      ",f)
print("          --------------------")
print("Output     ",adder_gate(e,f)) # it add two binary number

else:
    print("Invalid Input.")
    print("Enter Either [B/b] or [D/d].")
```

Figure 24 Screenshot no 5 of asinput.py module

7.3 Gates module

```
# gates.py module is created for adding two gates value i.e. upper and lower gates
# In this module "x" is consider as upper gates and "y" as lower gates which use
# bitwise operators
# Auther: Alisha Shrestha, 10 September, 2020

# Creating and gate using bitwise operators
def and_gate(x, y):
    return x&y  # return keyword is used return the given value

# Creating or gate using bitwise operators
def or_gate(x, y):
    return x|y

# Creating xor gate using bitwise operator
def xor_gate(x, y):
    return x^y

# Creating not gate using bitwise operator
def not_gate(x):
    return (~x) + 2

# Creating nand gate
def nand_gate(x,y):
    return not_gate(and_gate(x,y))

#Creating nor gate
def nor_gate(x,y):
    return not_gate(or_gate(x,y))
```

Figure 25 Screenshot of gates.py module

7.4 Conversion module

```
# forconversion.py mpdule is created to covert binary and decimal number
# Auther: Alisha Shrestha, 10 September, 2020

# Creating function to convert binary into decimal
def bin_to_dec(value):
    dec_no = 0
    i = 0
    while (value > 0):
        last_no = value % 10
        dec_no += last_no * (2 ** i)
        i += 1
        value = int(value/10)
    return dec_no
##print(bin_to_dec(11111111))

# Creating function to convert decimal into binary
def dec_to_bin(value):
    list_of_eightbit = ["0","0","0","0","0","0","0","0"]
    index_ = 7
    while (value > 0):
        b = value % 2
        list_of_eightbit[index_] = str(b)
        index_ -= 1
        value = int(value/2)
    return "".join(list_of_eightbit)
##print(dec_to_bin(12))
##print(dec_to_bin(176))
```

Figure 26 Screenshot of forconversion.py module

7.5 Adder module

```
# foradder.py module is created to add the two binary number
# It import gates module to operate logical gates

from gates import *

# Creating adder gate function to add two number
def adder_gate(bin_no1,bin_no2):
    carry = 0 # initial carry is 0
    list_of_eightbit = ["0","0","0","0","0","0","0","0"] # Creating the list of 8 bits

    # Creating the iterator of range start, stop and step
    for a in range (7,-1,-1):
        upper_bit =int(bin_no1[a]) # upper bit value going through iteration
        lower_bit = int(bin_no2[a]) # lower bit value going through iteration

        xor_ = xor_gate(upper_bit, lower_bit) # call xor_gate function
        nand_ = nand_gate(xor_,carry) # call nand_gate function
        or_ = or_gate(xor_,carry) # call or_gate function
        sum_of_bit = and_gate(or_,nand_) # call and_gate function

        list_of_eightbit[a] = str(sum_of_bit)

        and_no1 = and_gate(upper_bit, lower_bit)
        and_no2 = and_gate(xor_, carry)
        carry = not_gate(nor_gate(and_no1, and_no2))

    # return 8 bit binary number after going through logical circuit
    return "".join(list_of_eightbit)
```

Figure 27 Screenshot of foradder.py module

7.6 Validation.py module

```
# validaiton.py module is created for validation of binary and decimal number.
# Auther: Alisha Shrestha, 10 September, 2020

# creating funcition for binary validation.
def binary_validation(input_bin):
    if (input_bin == ""): # it check empty value
        return [True,"Error !!! Please enter the value."]

    # try and except is used to handle the exception
    try:
        check = int(input_bin) # it check whether the value is integer type or not
    except:
        return [True,"Error !!! Accepts integer value only."]

    if (int(input_bin) < 0): # it check whether the value is less than zero or not
        return [True, "Please enter positive value."]
    else:
        for a in input_bin:
            if (int(a) not in [0,1]): # it check the input value is 0 and 1 only
                return [True,"Invalid input!!! Enter either 0 or 1."]

    if (len(input_bin)> 8): # it check the length of the input value
        return[True, "Invalid input!!! Enter 8 bit binary number"]
    return [False]
```

Figure 28 Screenshot no 1 of validation.py module

```
def decimal_validation(input_dec):
    if (input_dec == ""): # it check empty value
        return[True,"Error !!! Please enter the value."]

    # try and except is used to handle the exception
    try:
        check = int(input_dec) # it check whether the value is integer type or not
    except:
        return [True,"Error !!! Accepts integer value only."]

    # it check the value whether it is not less than 0 and not more than 255
    if (int(input_dec) < 0 or int(input_dec) > 255):
        return [True, "Please enter value more than zero and less than 255"]
    return [False]
```

Figure 29 Screenshot no 2 of validation.py module

8. Testing

8.1 Test no 1

Table 9 Test no 1

Test no	1
Action	Running the main module and enter wrong data type instead of either b/B or d/D.
Expected output	When the wrong data type is entered an error message should be displayed.
Actual output	An error message is displayed and the program asks for entering either binary or decimal number again.
Test result	Test has been successfully done.

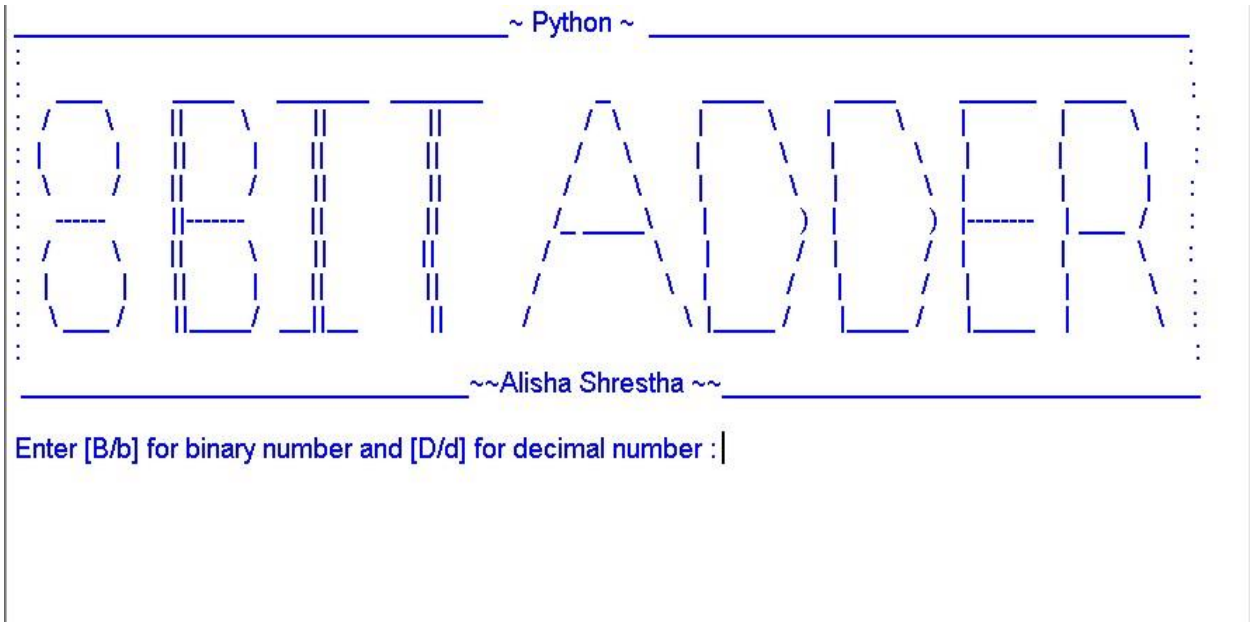

Figure 30 Running the main.py module

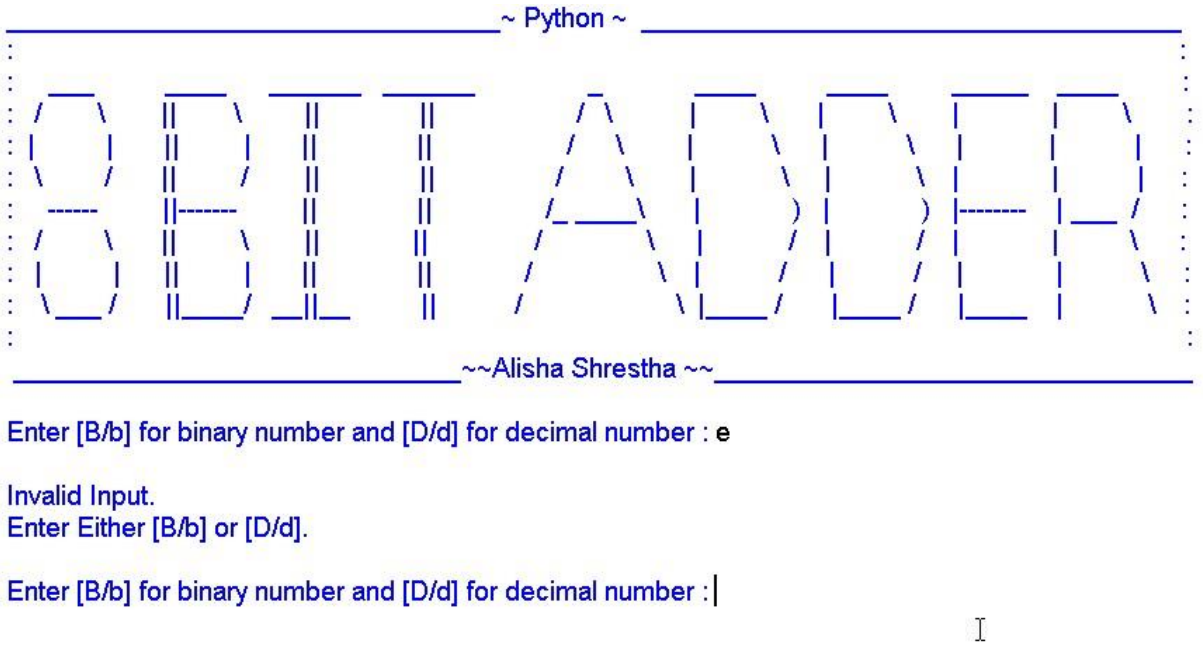

Figure 31 Entering wrong data

8.2 Test no 2

Table 10 Test no 2

Test no	2
Action	Entering wrong data type in first binary number.
Expected output	An error message must be displayed and the program must ask again for entering the first binary number.
Actual output	When entering the wrong data type it displays an error message and asks to enter the binary number again.
Test result	Test has been successfully done.

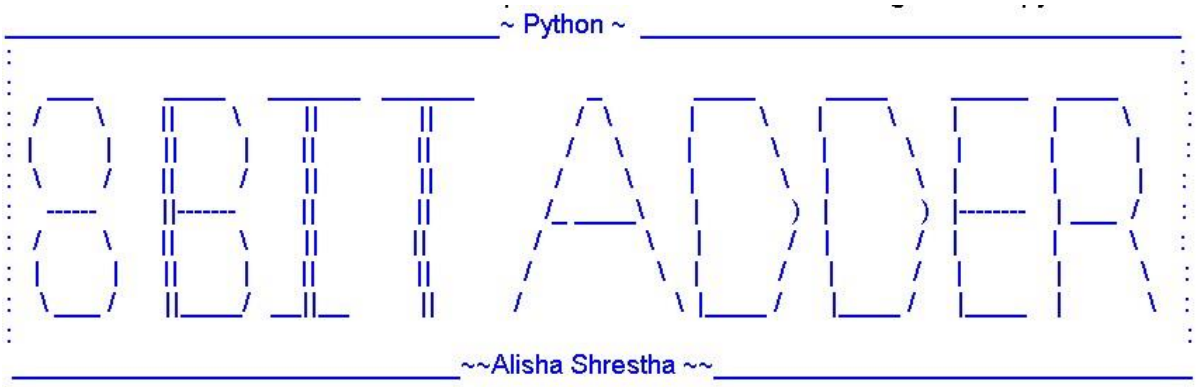

Figure 32 Entering value for binary number

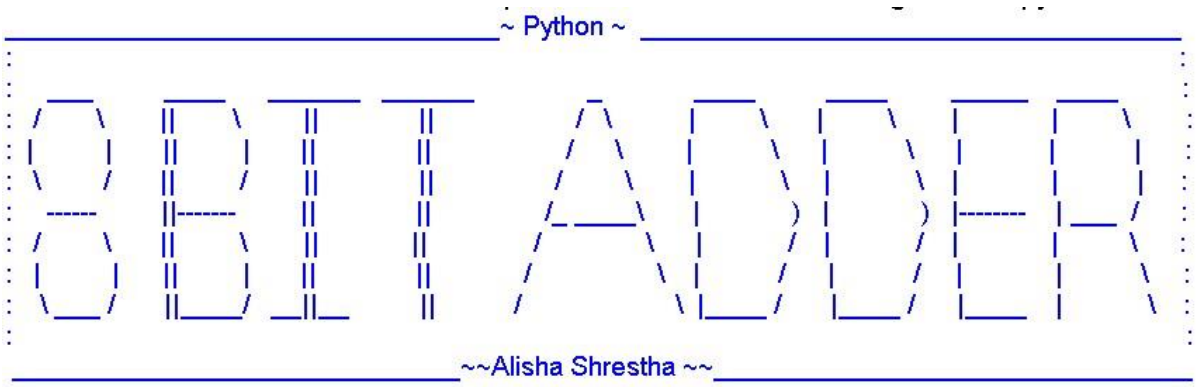

Figure 33 Entering wrong data type for first binary number

8.3 Test no 3

Table 11 Test no 3

Test no	3
Action	Entering two binary numbers for addition
Expected output	The addition operation of two binary numbers and the output must be displayed with the conversion of binary to decimal.
Actual output	The output with the addition value is displayed along with the conversion of binary to decimal.
Test result	Test has been successfully done.

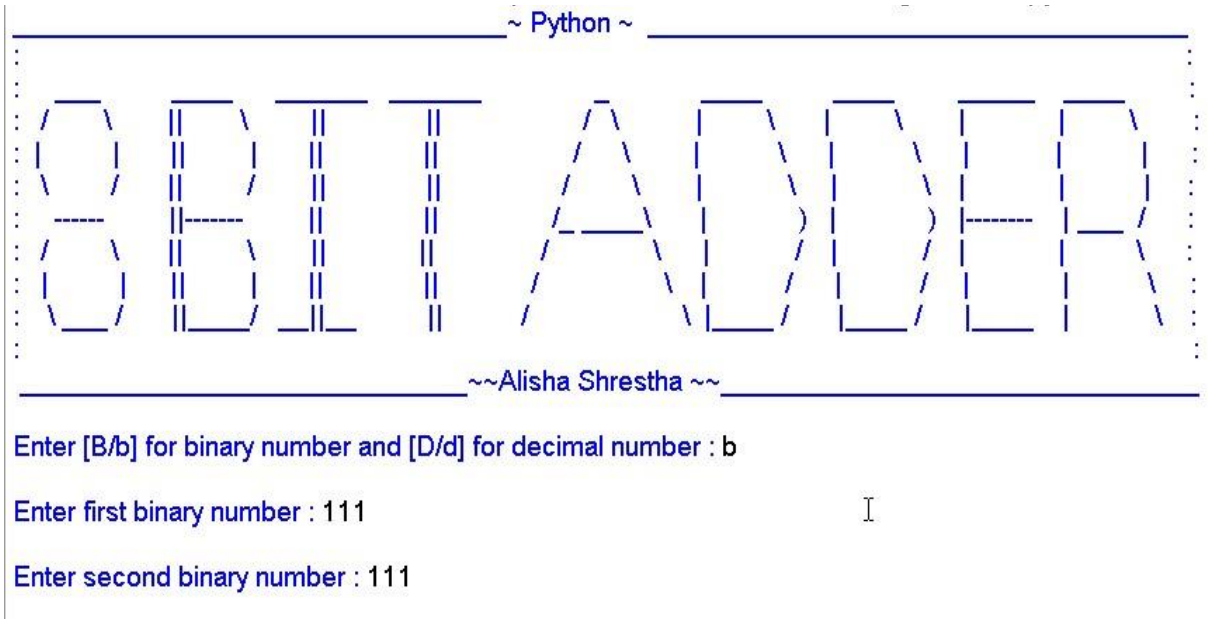

Figure 34 Entering two binary numbers for addition

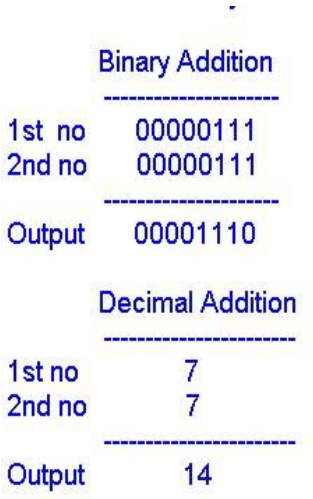

Figure 35 Output after entering two binary numbers

8.4 Test no 4

Table 12 Test no 4

Test no	4
Action	Entering two decimal numbers for addition
Expected output	The addition operation of two decimal numbers and the output must be displayed with the conversion of decimal to binary.
Actual output	The output with the addition value is displayed along with the conversion of binary to decimal.
Test result	Test has been successfully done.

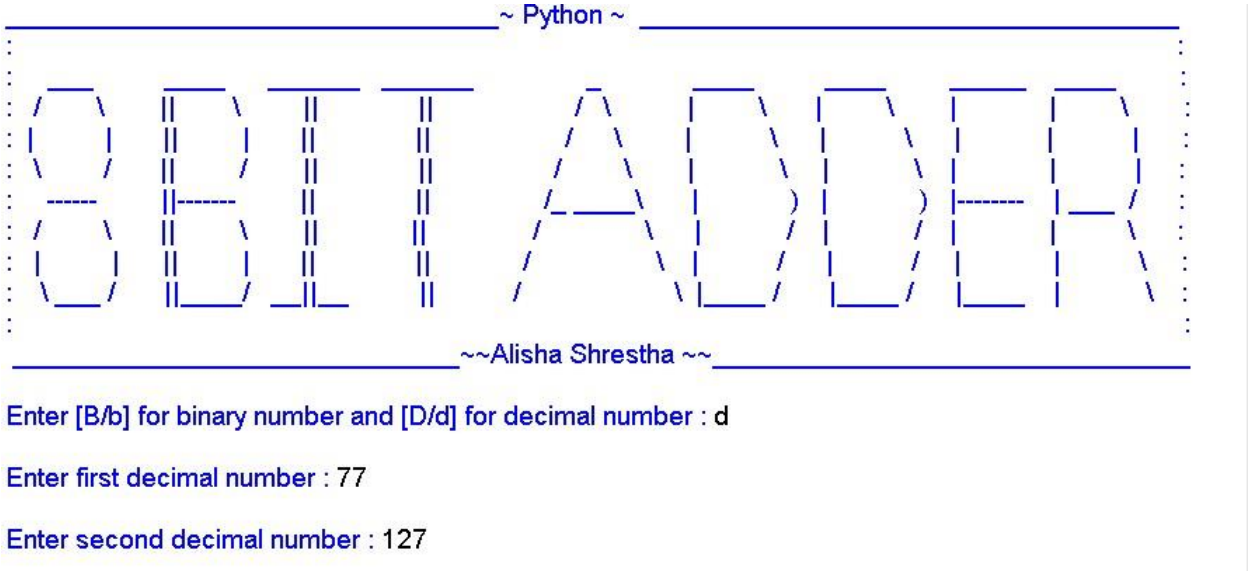

Figure 36 Entering two decimal numbers for addition

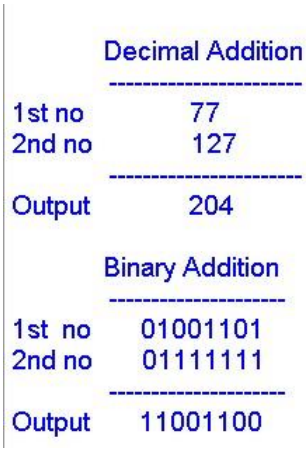

Figure 37 Output after entering two decimal number

8.5 Test no 5

Table 13 Test no 5

Test no	5
Action	Entering less than 0 and more than 255
Expected output	An error message must be displayed and program should ask again to enter number.
Actual output	The error message is displayed and the program asks for entering number again.
Test result	Test has been successfully done.

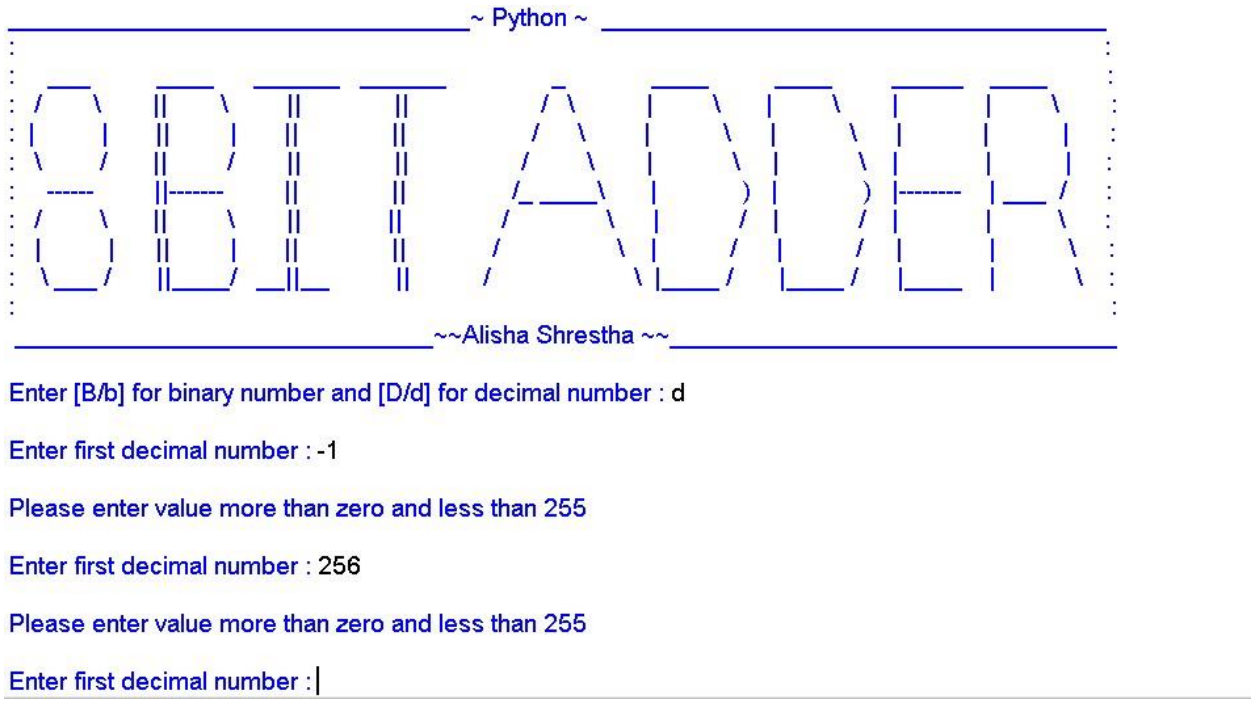

Figure 38 Entering wrong value

8.6 Test no 6

Table 14 Test no 6

Test no	5
Action	Entering “No” for exit
Expected output	After entering “No”, the program must ended and display information message.
Actual output	The program is ended by displaying a suitable message.
Test result	Test has been successfully done.

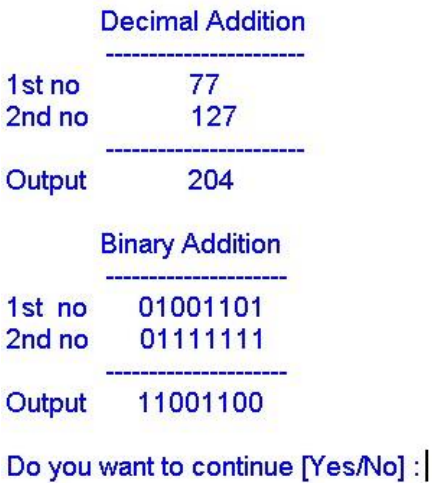

Figure 39 Output of binary and decimal numbers

```
          Decimal Addition
          ----------------------
1st no          77
2nd no          127
          ----------------------
Output          204

          Binary Addition
          --------------------
1st  no     01001101
2nd no      01111111
          --------------------
Output      11001100

Do you want to continue [Yes/No] : no


Thank you for using this program.

>>>
```

Figure 40 Exciting the program

8. Conclusion

To wrap up the overall documentation of this project many difficulties were faced throughout but many researches were also done to overcome those difficulties. This was the very first coursework in Python programming Language, though this was the first but after doing lots of researches and practices it was found that this programming language was much simpler than other programming languages and also this was fun as well.

In order to complete the report for the whole project all the procedure were included in the report with necessary screenshots, description etc. for more clarification. Many researches were carried out for successful completion of this task and researches were done on the relevant topics such as byte adder, algorithm, pseudocode, flowchart, data structure, circuit diagram, testing the program etc.

As a first step, model circuit was constructed where 8 bit adder and parallel circuit were prepared along with the selection of a data structure for writing the program. In the same manner as the next step algorithm and pseudocode of the program was written. With the completion of algorithm and pseudocode flow chart is prepared for describing the flow of the program. Afterthe coding part testing is performed to ensure about the program is providing accurate output or not.

The main focus of this project was integer addition based on the binary and decimal operation including the conversion of binary and decimal number. Data structure also helped in storing data, organizing data and operation performed in the program.

This project was not only the college assignment but it was more like a practice for performing the individual task with much ease and effectively which will be very useful for the future projects in the real world. The main plus point for this coursework was of learning a different programming language and implementing that into something useful and creative.

The concept while getting involved in this project was first focused on only completing the task but later on it got more interesting day by day as it was fun to code and easy to implement, moreover the algorithm, pseudocode and flow chart made it easier to

develop the program. The other benefit of this task in practical life was that many technical words got familiar and that was only possible due to lots of research. Another interesting finding was learning about how the computer computes the sum of two integer value. This project was not easy for completing but it was done with all the hard work and research.

8.1 Research and finding

8.1.1 Website

i. https://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR#:~:text=A%20logic%20gate%20is%20a,)%20or%20true%20(low).

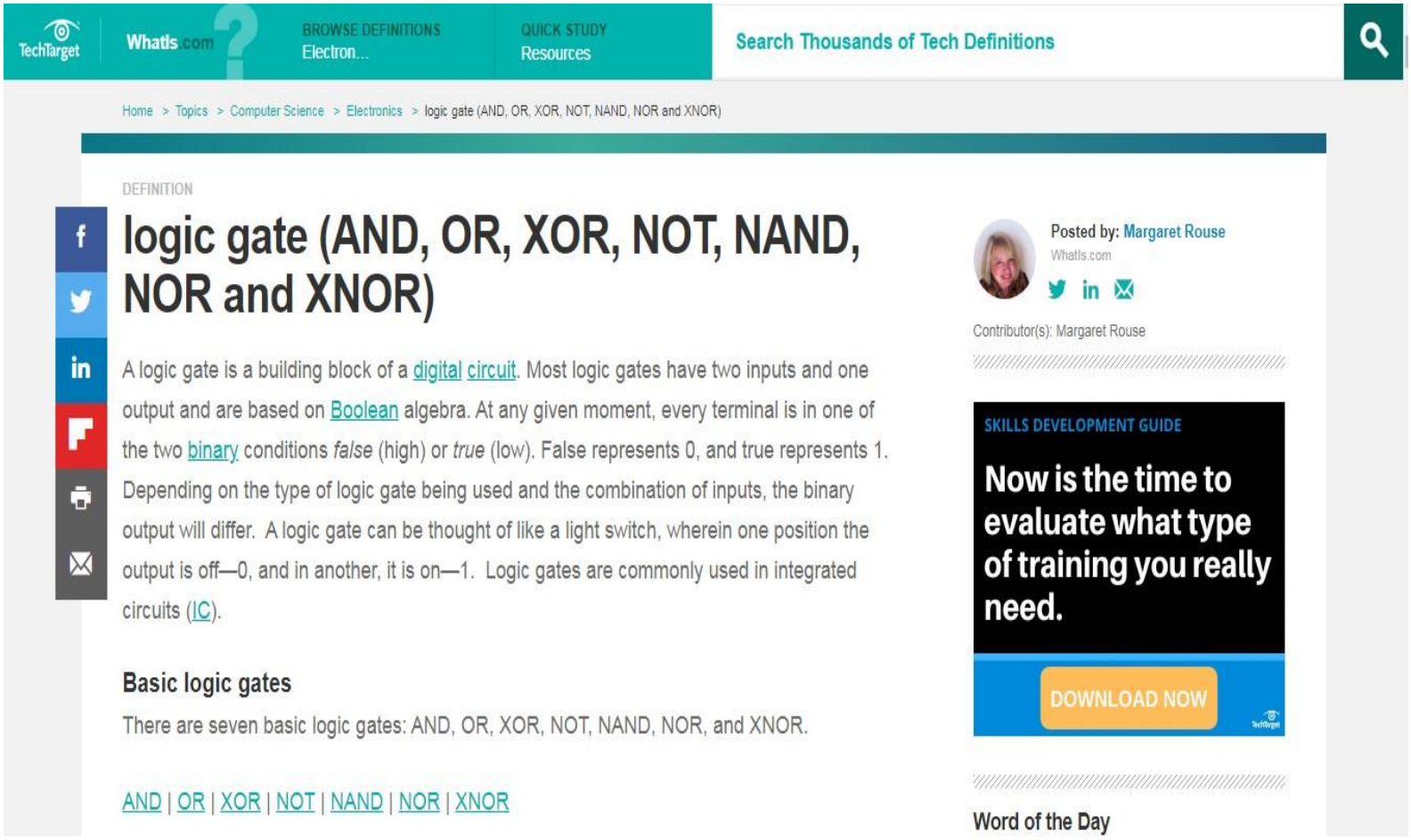

Figure 41 logical gates

ii. https://www.elprocus.com/half-adder-and-full-adder/

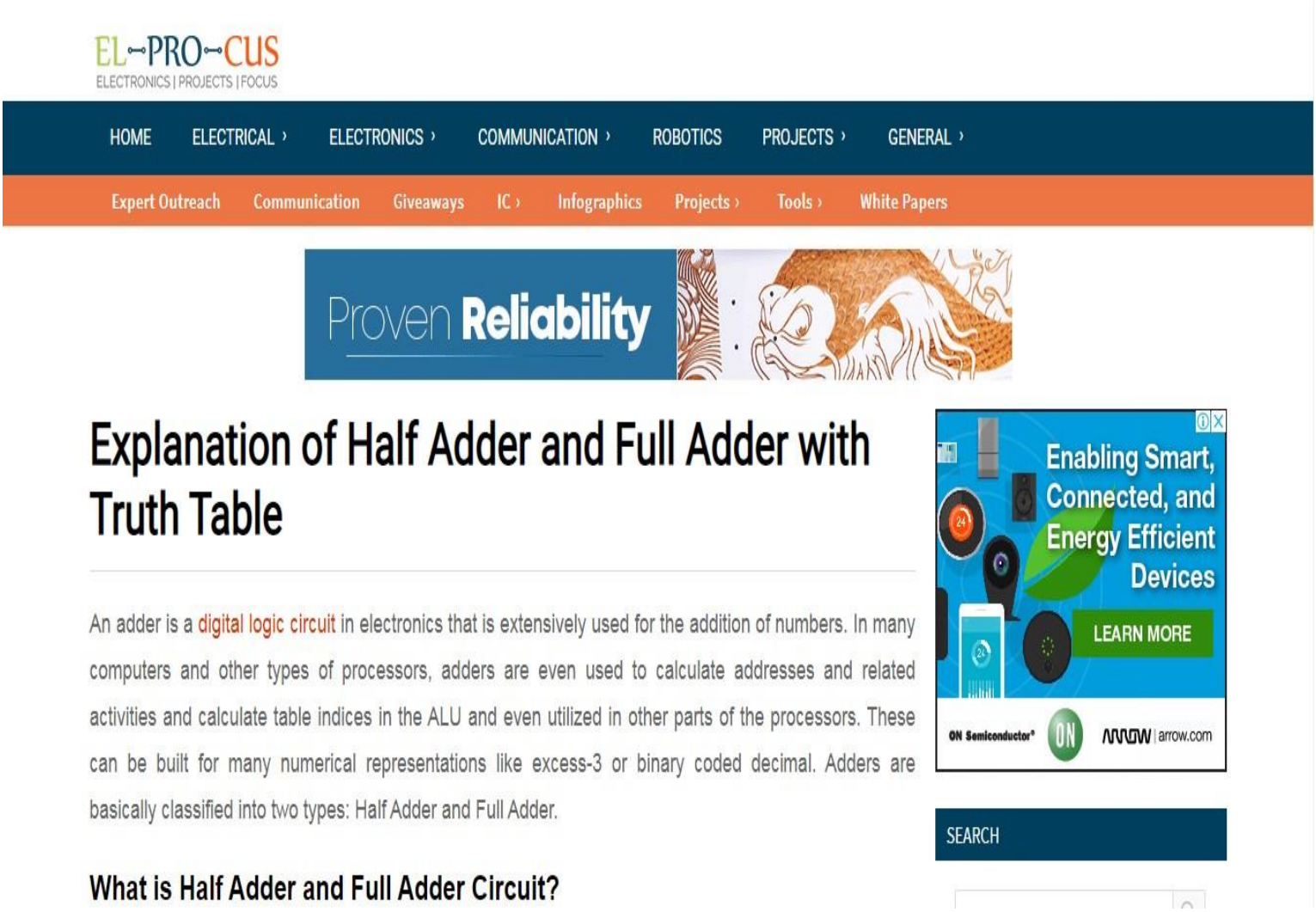

Figure 42 Half and full adder Circuit

iii. https://www.geeksforgeeks.org/data-structures/

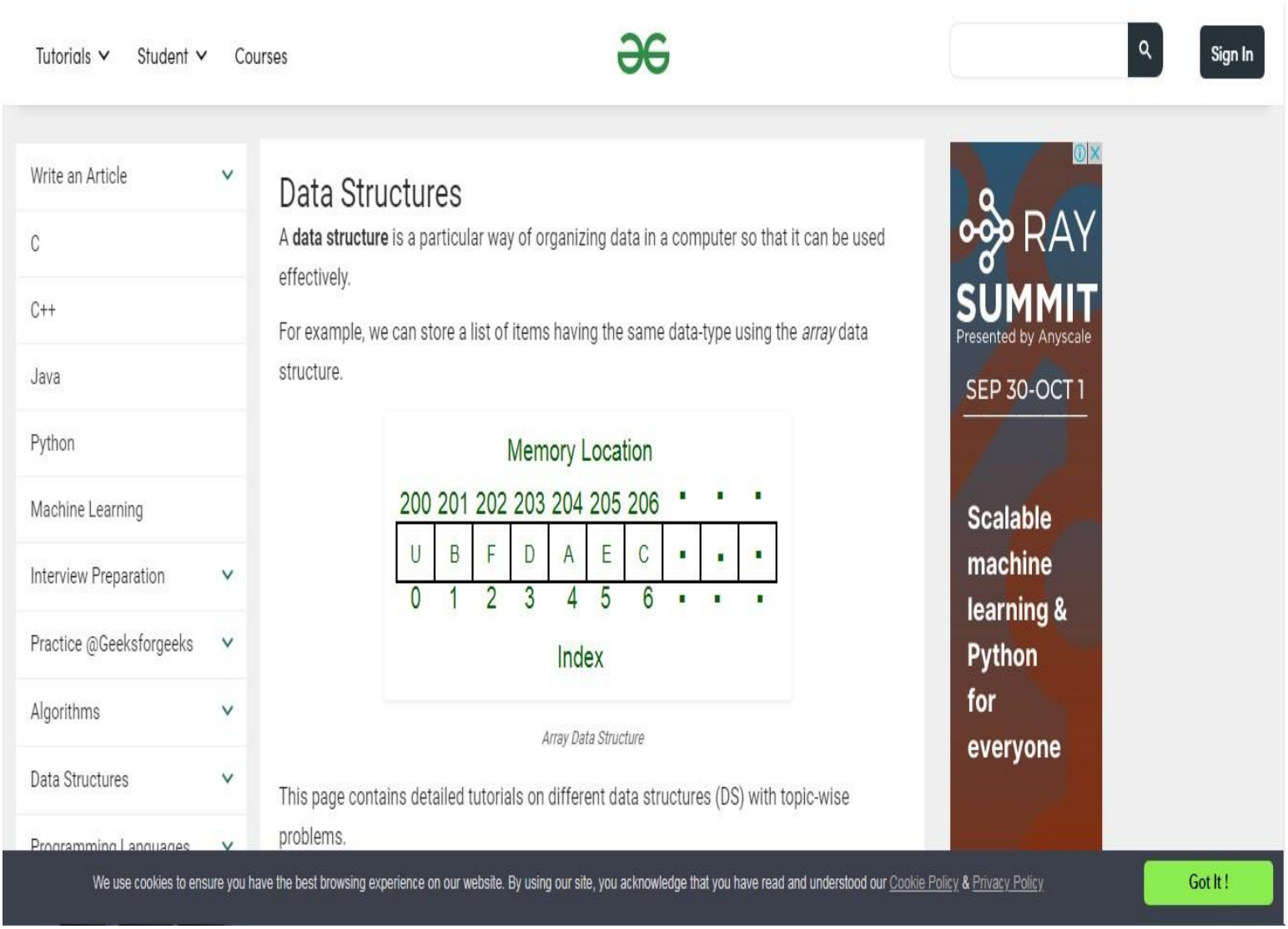

Figure 43 Data structure

iv. https://kullabs.com/class-9/computer-1/algorithm-and-flowchart/algorithm-flowchart-and-its-types

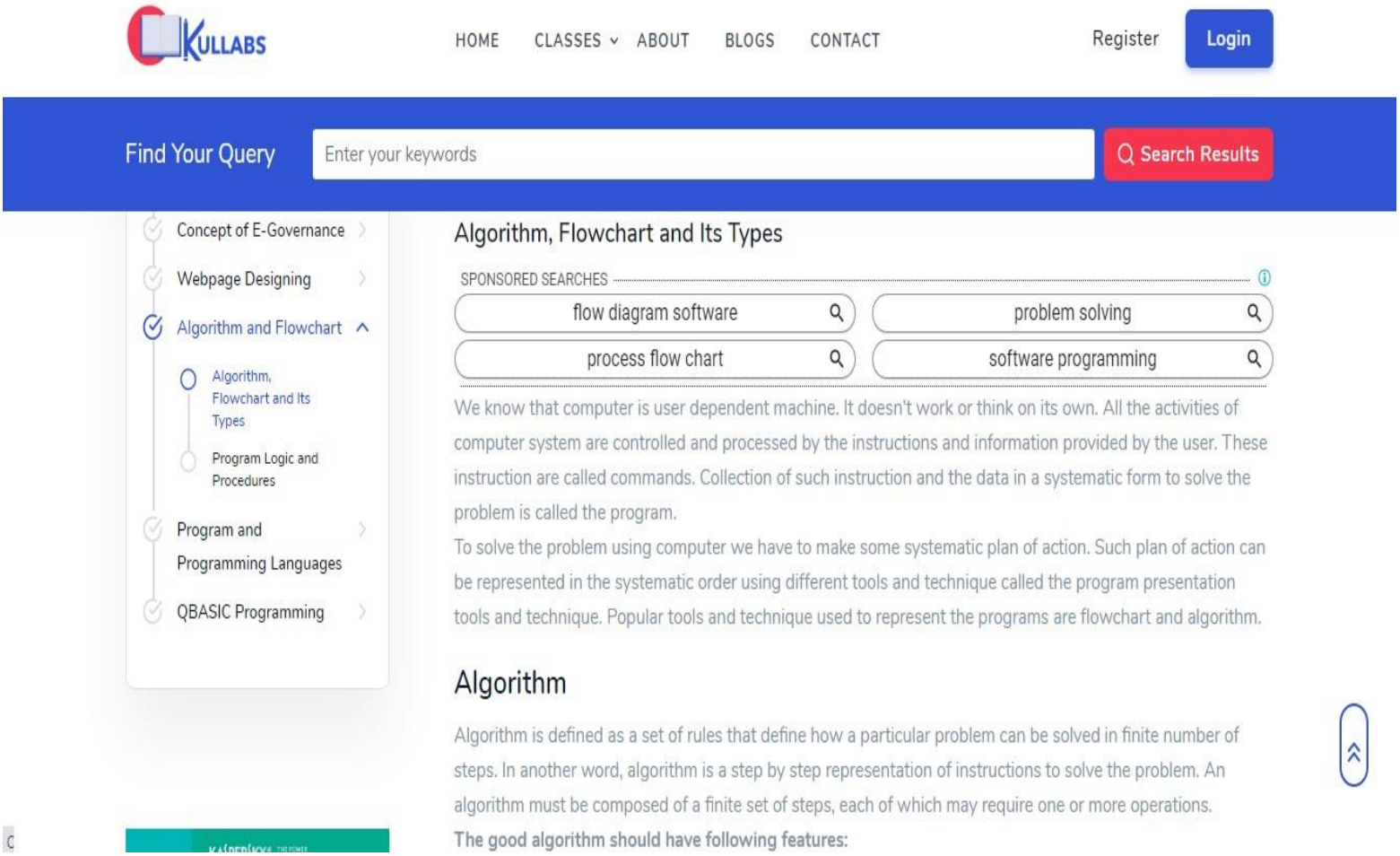

Figure 44 Algorithm

v. https://www.datacamp.com/community/tutorials/data-structures-python

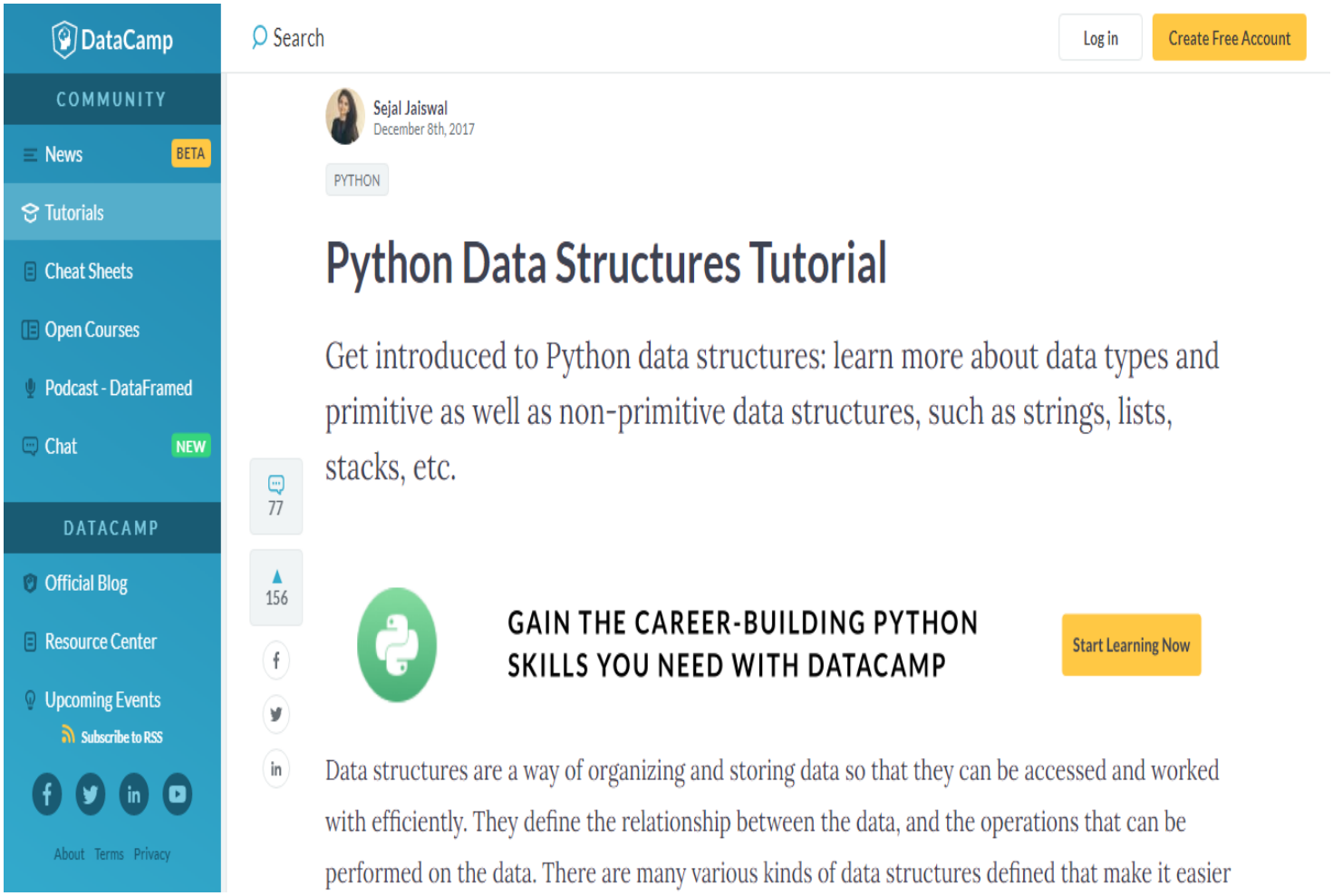

Figure 45 Data Structure

vi. https://economictimes.indiatimes.com/definition/pseudocode

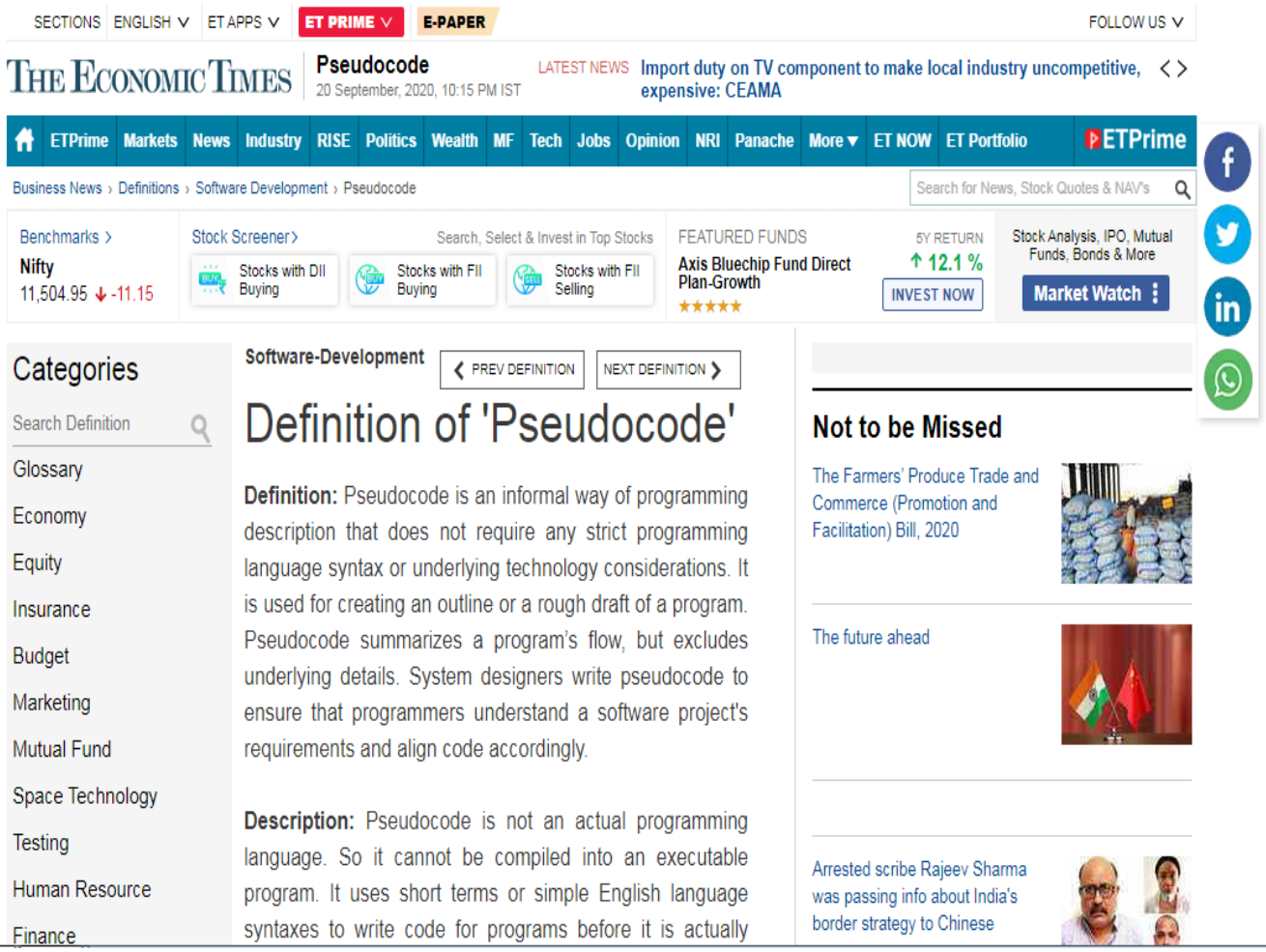

Figure 46 Pseudocode

Bibliography

Akash. (2019, November 26). *Edureka*. Retrieved from www.edureka.com: https://www.edureka.co/blog/data-structures-in-python/

GeeksforGeeks. (2020, 06 22). *GeeksforGeeks*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/python-features/

geeksforgeeks. (n.d.). *Python Dictionary*. Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/python-dictionary/#:~:text=Dictionary%20in%20Python%20is%20an,Dictionary%20holds%20key%3Avalue%20pair.&text=Note%20%E2%80%93%20Keys%20in%20a%20dictionary%20doesn't%20allows%20Polymorphism.

Jaiswaly, S. (n.d.). *datacamp*. Retrieved december 2017, from www.datacamp.com: https://www.datacamp.com/community/tutorials/data-structures-python

Kullabs. (2019). *kullabs.com*. Retrieved from kullabs: https://kullabs.com/class-9/computer-1/algorithm-and-flowchart/algorithm-flowchart-and-its-types

laps, P. (n.d.). *programiz.com*. Retrieved from Programiz: https://www.programiz.com/dsa/algorithm

Paradigm, V. (n.d.). *visual-paradigm.com*. Retrieved from www.visual-paradigm.com: https://www.visual-paradigm.com/tutorials/flowchart-tutorial/

Programiz. (n.d.). *www.programiz.com*. Retrieved from programiz.com: https://www.programiz.com/python-programming/variables-datatypes#:~:text=Integers%2C%20floating%20point%20numbers%20and,or%20a%20value%20belongs%20to.&text=1%20is%20an%20integer%2C%201.0%20is%20a%20floating%2Dpoint%20number.

Rouse, M. (2005). *Whatis*. Retrieved from whatis.techtarget.com: https://whatis.techtarget.com/definition/pseudocode

StudyTonight. (n.d.). *studytonight.com*. Retrieved from www.studytonight.com: https://www.studytonight.com/data-structures/introduction-to-data-structures

techterms. (n.d.). *techterms*. Retrieved from techterms.com: https://techterms.com/definition/python

The Economic Times. (2020). Retrieved from economictimes.indiatimes.com: https://economictimes.indiatimes.com/definition/pseudocode

ThePSF. (n.d.). *Python*. Retrieved from www.python.org: https://www.python.org/doc/essays/blurb/

Tulchak, L. V. (2016). History of Python. In A. O. Marchuk, *History of Python.* ВНТУ.