Mahidol University
International College

ITCS323 Computer Data Communication
Lectured by: Asst. Prof. Thitinan Tantidham

Class Project

Working Title: Error Detection Project

Submitted by: 6388133 Pitchaya Teerawongpairoj
6388196 Sasima Srijanya

Parity Bit and Parity Check

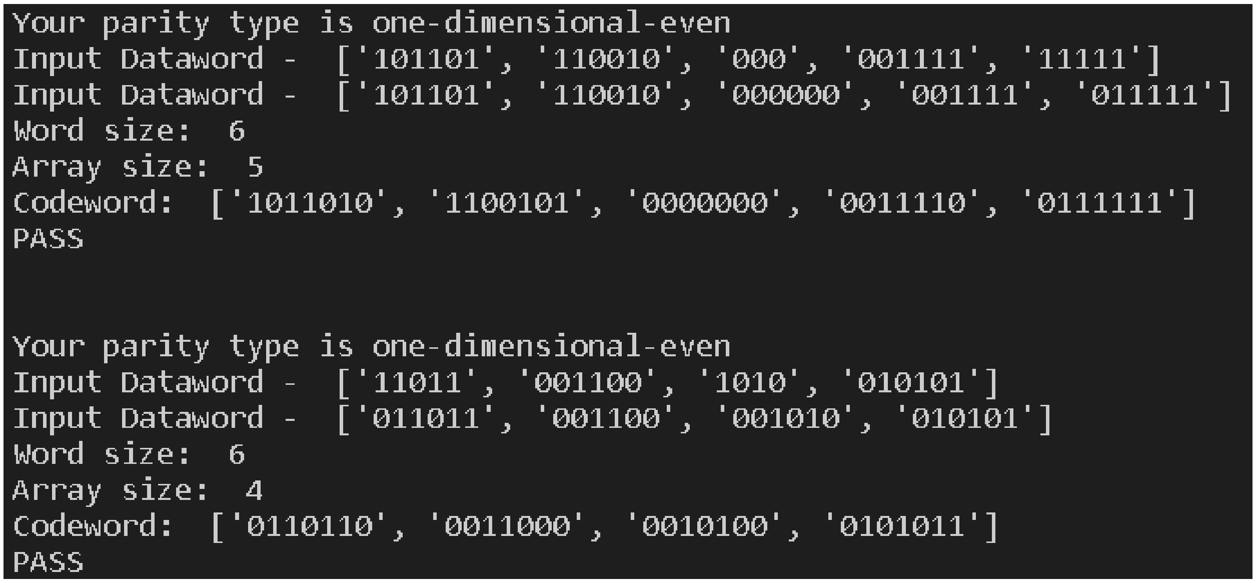
- How to run 1 -

- Run “python parity_test.py” in the terminal
- For the Generator Part: type your Dataword, Word Size, Parity type, and Array size then the program will print out a codeword at terminal.
- For the Checker Part: the program will automatically parity check and it will print out after the codeword.

- How to run 2 -

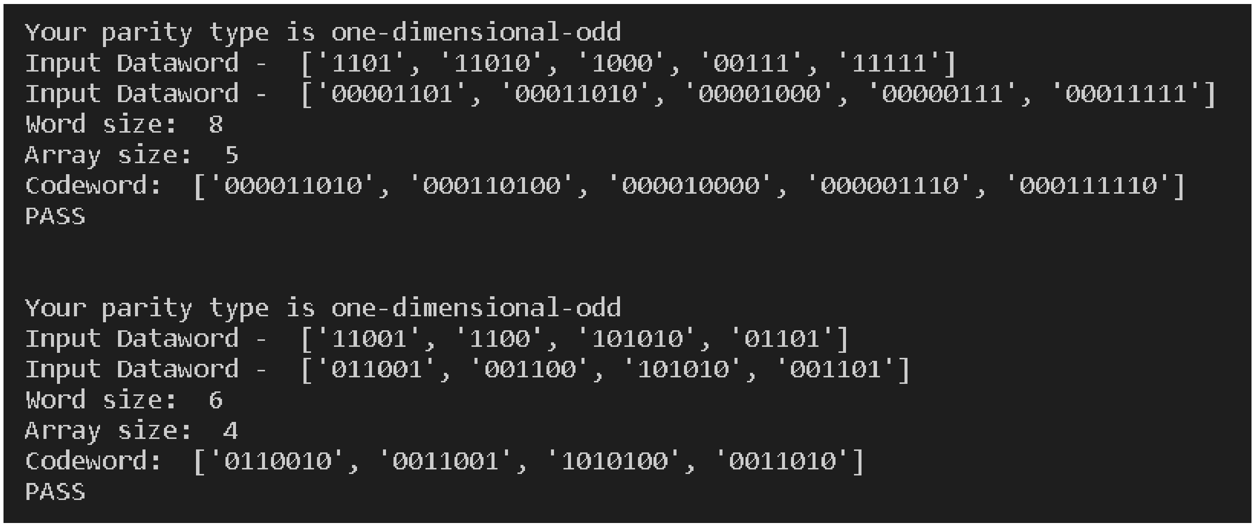
- Uncomment at main
- Type your input at the main function including Dataword, Word Size, Parity type, and Array size.
- Run “python parity.py” in the terminal

One-dimensional Even



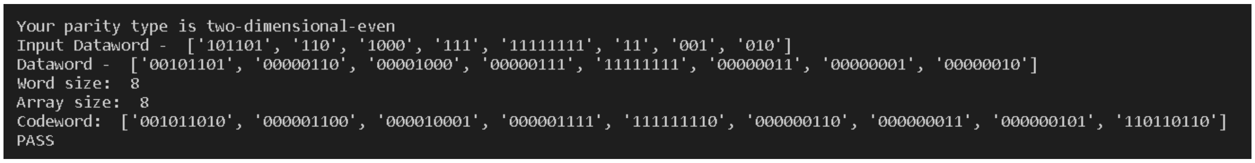
```
Your parity type is one-dimensional-even
Input Dataword -  ['101101', '110010', '000', '001111', '11111']
Input Dataword -  ['101101', '110010', '000000', '001111', '011111']
Word size:  6
Array size:  5
Codeword:  ['1011010', '1100101', '0000000', '0011110', '0111111']
PASS


Your parity type is one-dimensional-even
Input Dataword -  ['11011', '001100', '1010', '010101']
Input Dataword -  ['011011', '001100', '001010', '010101']
Word size:  6
Array size:  4
Codeword:  ['0110110', '0011000', '0010100', '0101011']
PASS
```

One-dimensional Odd

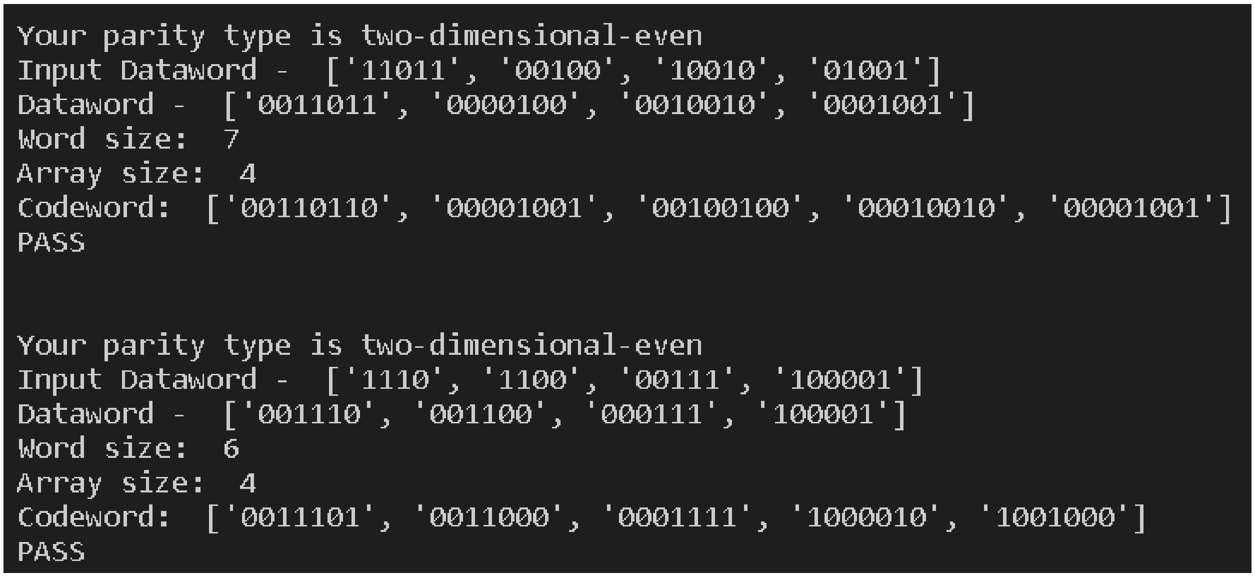


```
Your parity type is one-dimensional-odd
Input Dataword -  ['1101', '11010', '1000', '00111', '11111']
Input Dataword -  ['00001101', '00011010', '00001000', '00000111', '00011111']
Word size:  8
Array size:  5
Codeword:  ['000011010', '000110100', '000010000', '000001110', '000111110']
PASS


Your parity type is one-dimensional-odd
Input Dataword -  ['11001', '1100', '101010', '01101']
Input Dataword -  ['011001', '001100', '101010', '001101']
Word size:  6
Array size:  4
Codeword:  ['0110010', '0011001', '1010100', '0011010']
PASS
```

Two-dimensional Even

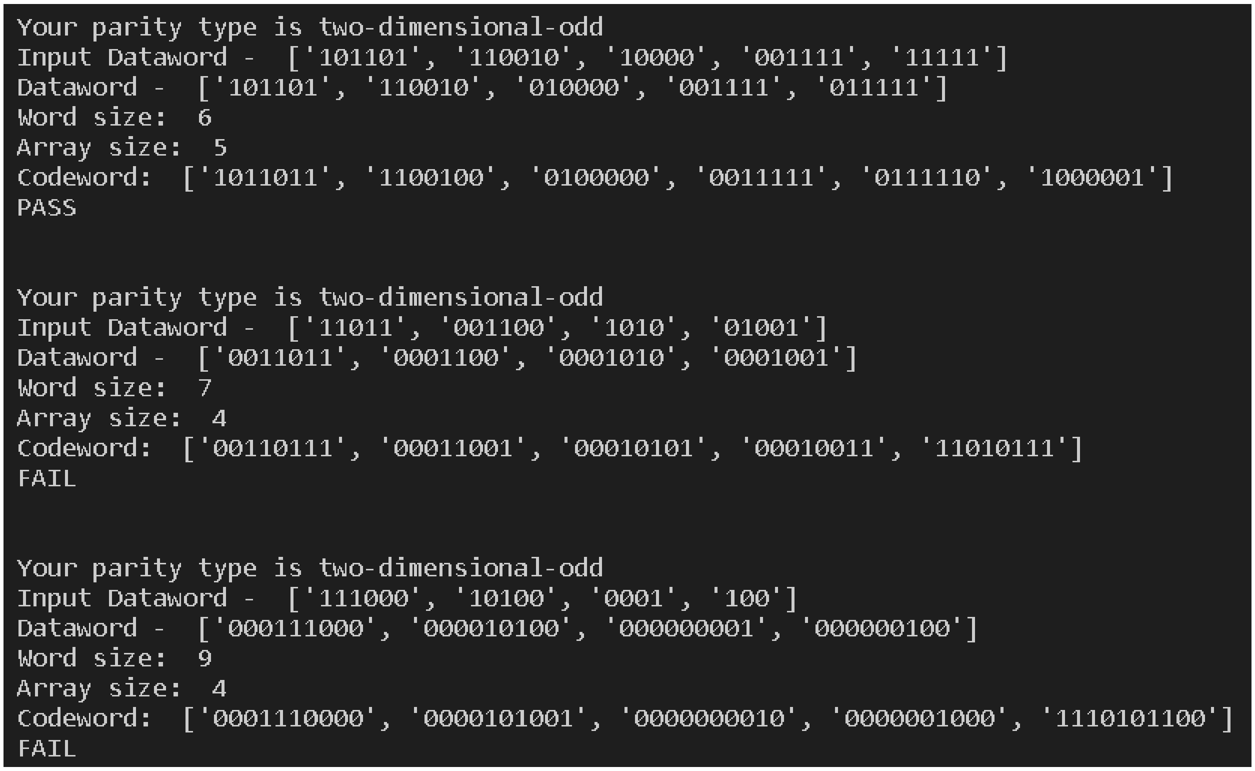


```
Your parity type is two-dimensional-even
Input Dataword -  ['101101', '110', '1000', '111', '11111111', '11', '001', '010']
Dataword -  ['00101101', '00000110', '00001000', '00000111', '11111111', '00000011', '00000001', '00000010']
Word size:  8
Array size:  8
Codeword:  ['001011010', '000001100', '000010001', '000001111', '111111110', '000000110', '000000011', '000000101', '110110110']
PASS
```



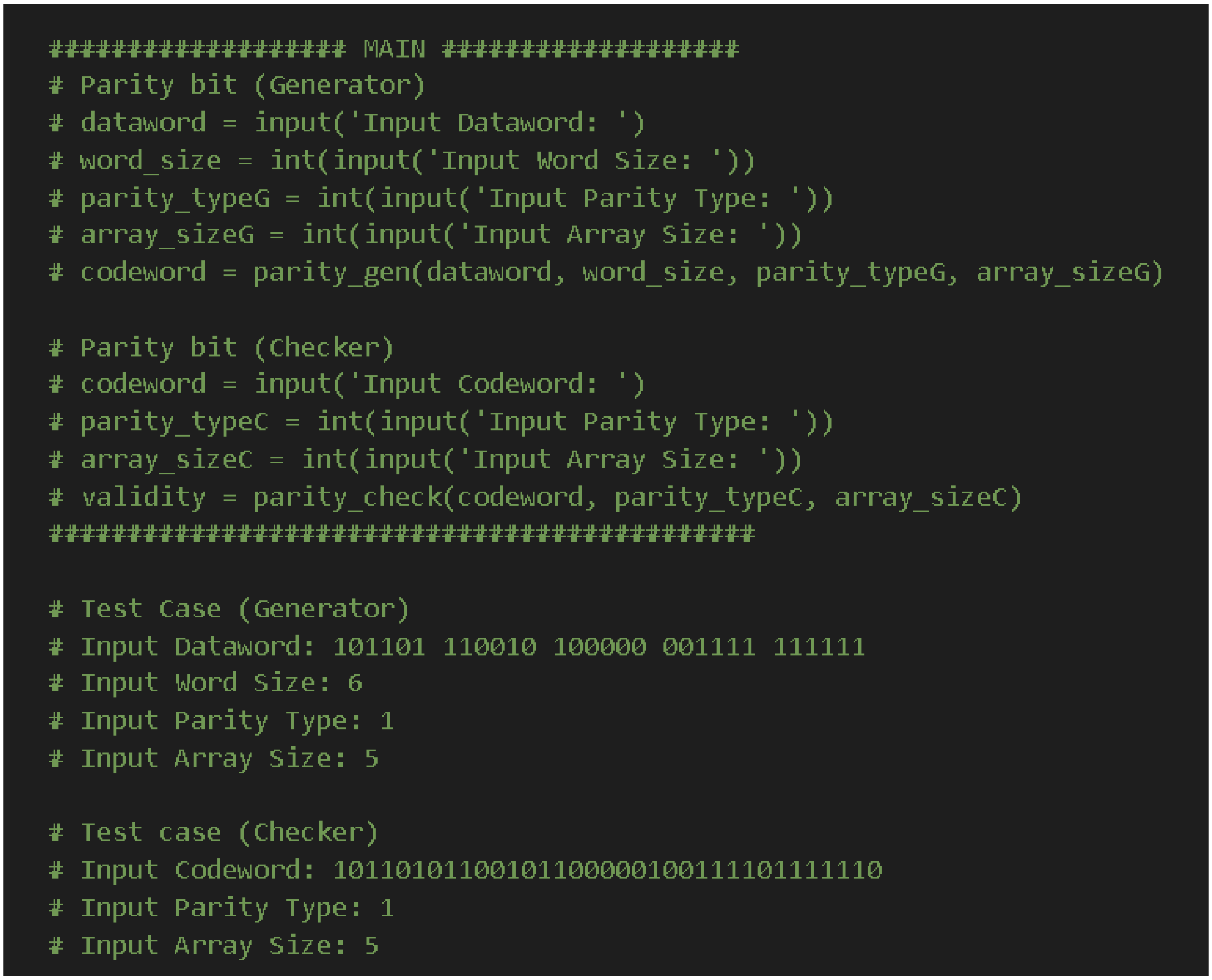
```
Your parity type is two-dimensional-even
Input Dataword -  ['11011', '00100', '10010', '01001']
Dataword -  ['0011011', '0000100', '0010010', '0001001']
Word size:  7
Array size:  4
Codeword:  ['00110110', '00001001', '00100100', '00010010', '00001001']
PASS


Your parity type is two-dimensional-even
Input Dataword -  ['1110', '1100', '00111', '100001']
Dataword -  ['001110', '001100', '000111', '100001']
Word size:  6
Array size:  4
Codeword:  ['0011101', '0011000', '0001111', '1000010', '1001000']
PASS
```

Two-dimensional Odd



```
Your parity type is two-dimensional-odd
Input Dataword -  ['101101', '110010', '10000', '001111', '11111']
Dataword -  ['101101', '110010', '010000', '001111', '011111']
Word size:  6
Array size:  5
Codeword:  ['1011011', '1100100', '0100000', '0011111', '0111110', '1000001']
PASS


Your parity type is two-dimensional-odd
Input Dataword -  ['11011', '001100', '1010', '01001']
Dataword -  ['0011011', '0001100', '0001010', '0001001']
Word size:  7
Array size:  4
Codeword:  ['00110111', '00011001', '00010101', '00010011', '11010111']
FAIL


Your parity type is two-dimensional-odd
Input Dataword -  ['111000', '10100', '0001', '100']
Dataword -  ['000111000', '000010100', '000000001', '000000100']
Word size:  9
Array size:  4
Codeword:  ['0001110000', '0000101001', '0000000010', '0000001000', '1110101100']
FAIL
```

Uncomment and type your input at the main function including Dataword, Word Size, Parity type, and Array size.

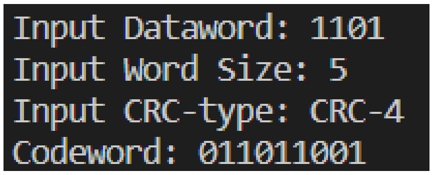


```
################### MAIN ###################
# Parity bit (Generator)
# dataword = input('Input Dataword: ')
# word_size = int(input('Input Word Size: '))
# parity_typeG = int(input('Input Parity Type: '))
# array_sizeG = int(input('Input Array Size: '))
# codeword = parity_gen(dataword, word_size, parity_typeG, array_sizeG)

# Parity bit (Checker)
# codeword = input('Input Codeword: ')
# parity_typeC = int(input('Input Parity Type: '))
# array_sizeC = int(input('Input Array Size: '))
# validity = parity_check(codeword, parity_typeC, array_sizeC)
############################################

# Test Case (Generator)
# Input Dataword: 101101 110010 100000 001111 111111
# Input Word Size: 6
# Input Parity Type: 1
# Input Array Size: 5

# Test case (Checker)
# Input Codeword: 101101011001011000001001111011111110
# Input Parity Type: 1
# Input Array Size: 5
```

Cyclic Redundancy Check Code (CRC)

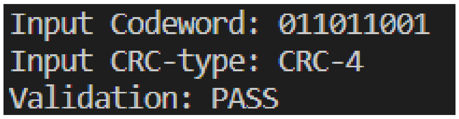
- How to run -

- Run “python crc.py” in the terminal
- For the Generator Part: Put in a Dataword, Word Size, and CRC-Type then the program will print out a codeword.



```
Input Dataword: 1101
Input Word Size: 5
Input CRC-type: CRC-4
Codeword: 011011001
```

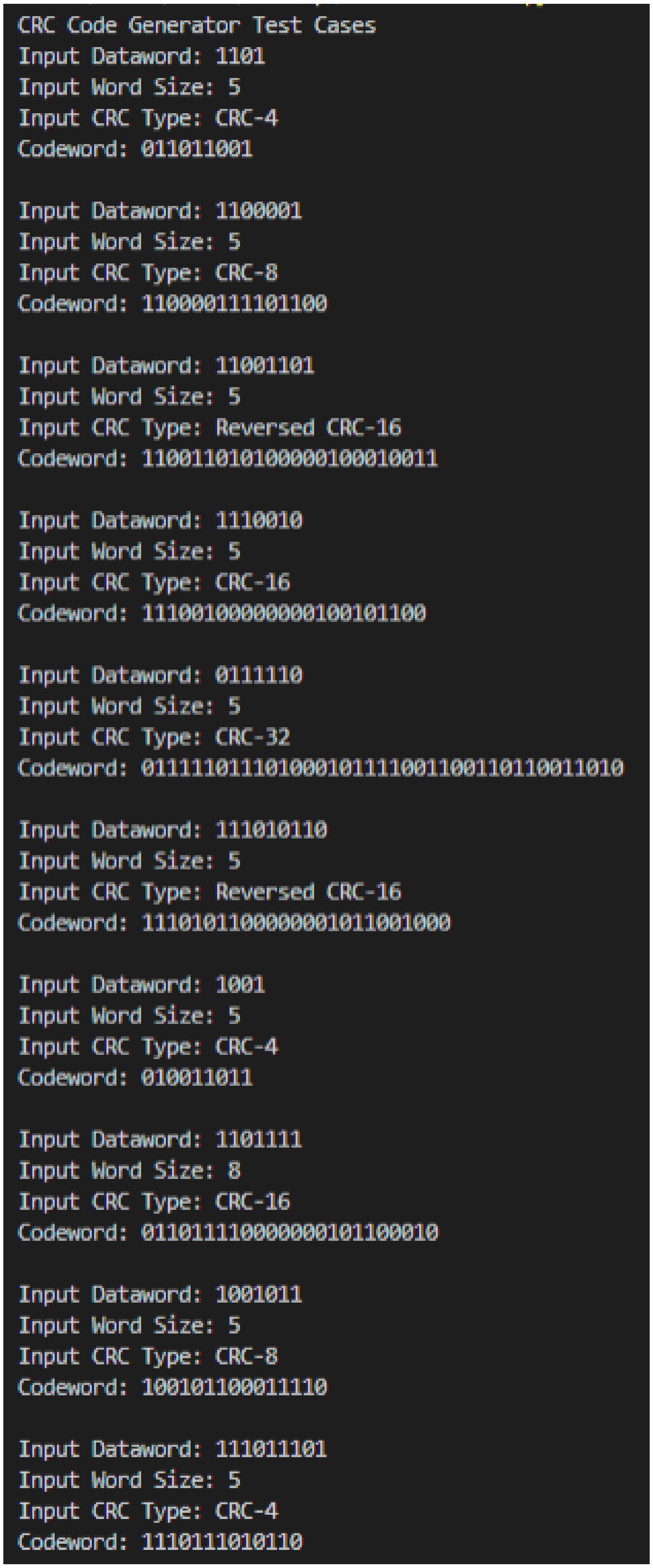
- For the Checker Part: Put in a Codeword and CRC-Type then the program will print out Pass message if there is no error and print out Fail message if there is an error.



```
Input Codeword: 011011001
Input CRC-type: CRC-4
Validation: PASS
```

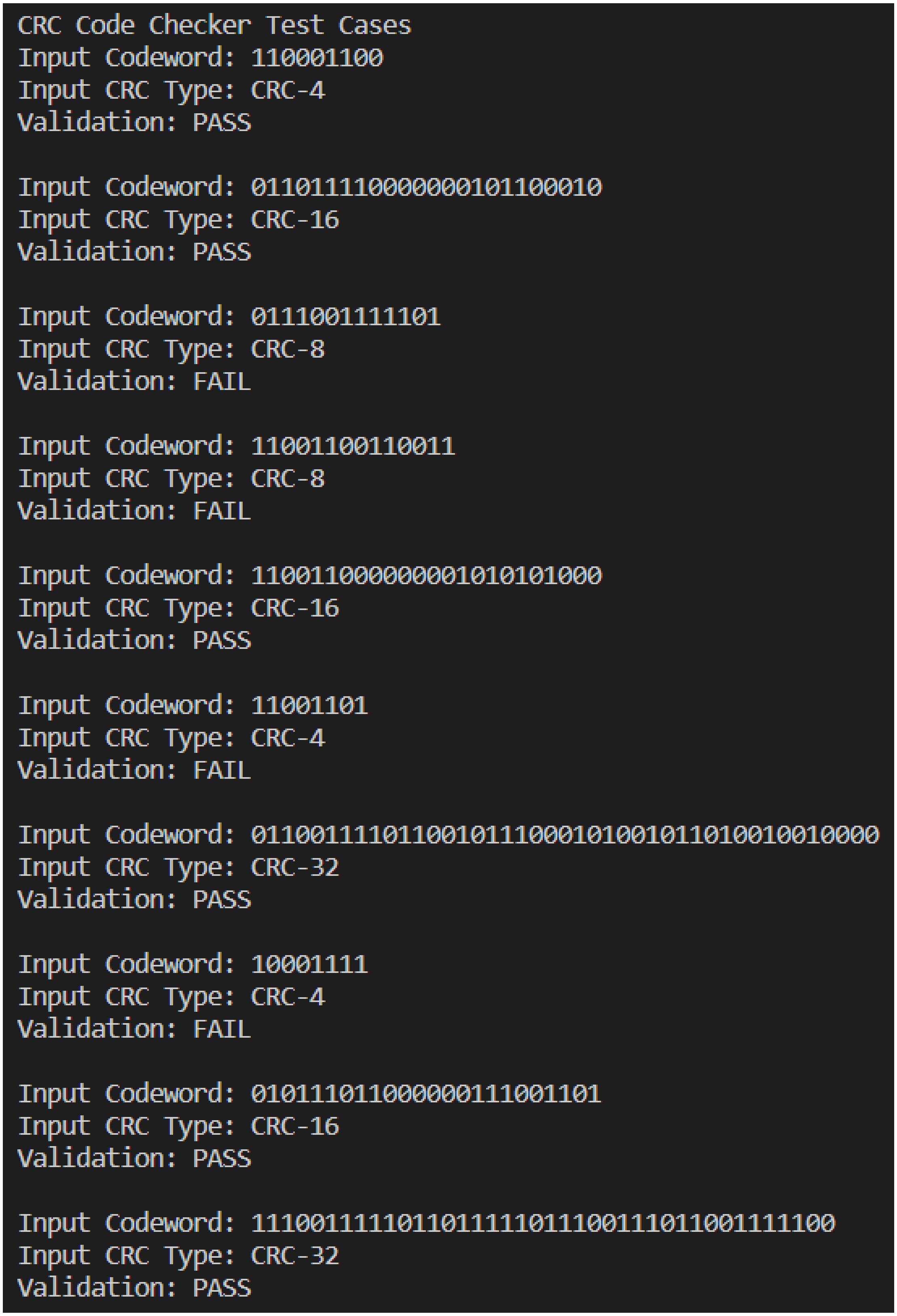
- Additional, run all of the test cases in a “python crc_test.py”

- Test Cases For Generator -



```
CRC Code Generator Test Cases
Input Dataword: 1101
Input Word Size: 5
Input CRC Type: CRC-4
Codeword: 011011001

Input Dataword: 1100001
Input Word Size: 5
Input CRC Type: CRC-8
Codeword: 110000111101100

Input Dataword: 11001101
Input Word Size: 5
Input CRC Type: Reversed CRC-16
Codeword: 1100110101000001000010011

Input Dataword: 1110010
Input Word Size: 5
Input CRC Type: CRC-16
Codeword: 11100100000000100101100

Input Dataword: 0111110
Input Word Size: 5
Input CRC Type: CRC-32
Codeword: 011111011101000101111001100110110011010

Input Dataword: 111010110
Input Word Size: 5
Input CRC Type: Reversed CRC-16
Codeword: 1110101100000001011001000

Input Dataword: 1001
Input Word Size: 5
Input CRC Type: CRC-4
Codeword: 010011011

Input Dataword: 1101111
Input Word Size: 8
Input CRC Type: CRC-16
Codeword: 0110111100000000101100010

Input Dataword: 1001011
Input Word Size: 5
Input CRC Type: CRC-8
Codeword: 100101100011110

Input Dataword: 111011101
Input Word Size: 5
Input CRC Type: CRC-4
Codeword: 1110111010110
```
)

- Test Cases For Checker -



```
CRC Code Checker Test Cases
Input Codeword: 110001100
Input CRC Type: CRC-4
Validation: PASS

Input Codeword: 011011110000000101100010
Input CRC Type: CRC-16
Validation: PASS

Input Codeword: 0111001111101
Input CRC Type: CRC-8
Validation: FAIL

Input Codeword: 11001100110011
Input CRC Type: CRC-8
Validation: FAIL

Input Codeword: 110011000000001010101000
Input CRC Type: CRC-16
Validation: PASS

Input Codeword: 11001101
Input CRC Type: CRC-4
Validation: FAIL

Input Codeword: 0110011110110010111000101001011010010010000
Input CRC Type: CRC-32
Validation: PASS

Input Codeword: 10001111
Input CRC Type: CRC-4
Validation: FAIL

Input Codeword: 010111011000000111001101
Input CRC Type: CRC-16
Validation: PASS

Input Codeword: 1110011111011011111011100111011001111100
Input CRC Type: CRC-32
Validation: PASS
```

Checksum

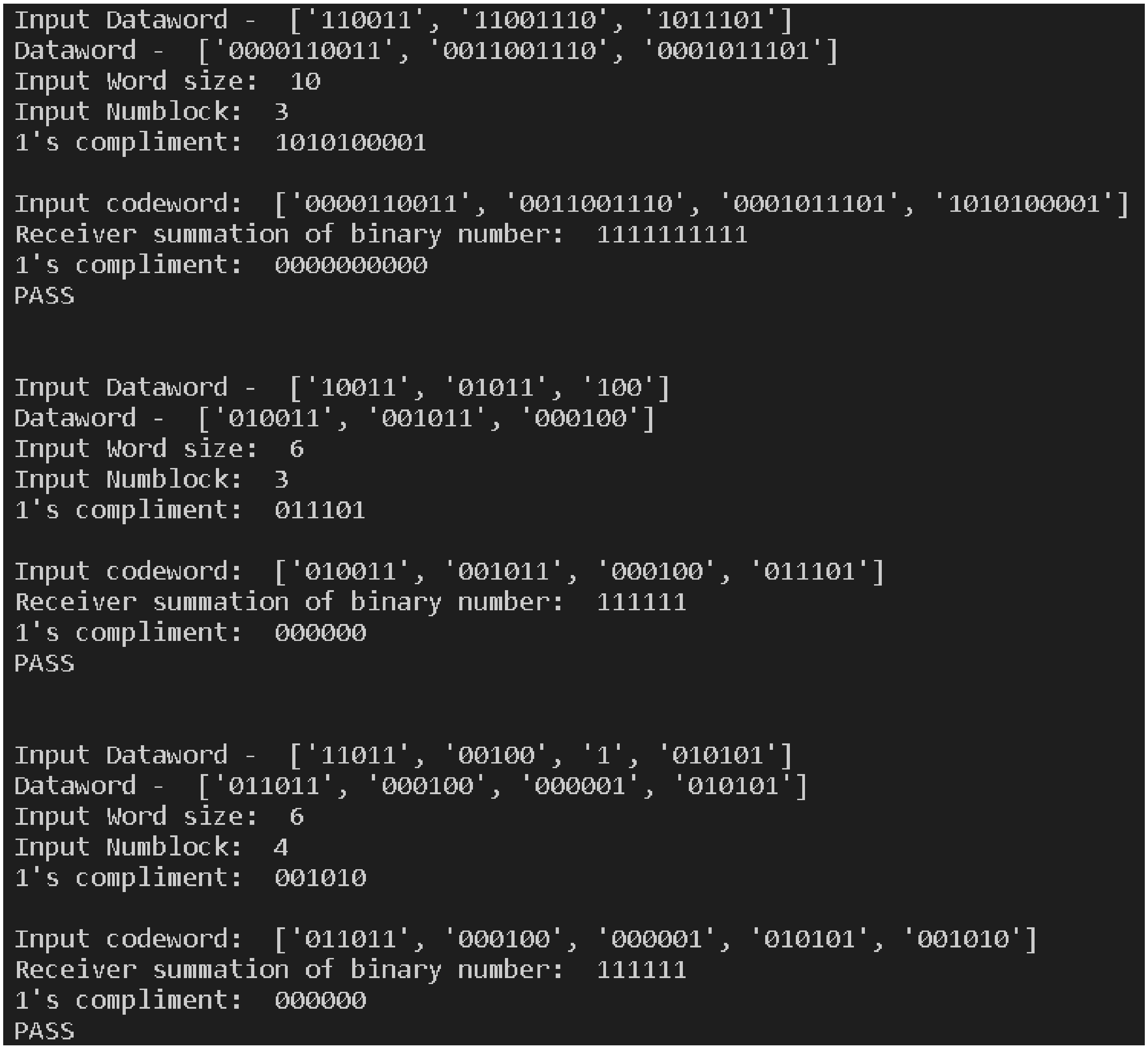
- How to run 1 -

- Run “python checksum_test.py” in the terminal
- For the Generator Part: type your Dataword, Word Size, and number of blocks. Then the program will print out a codeword at the terminal.
- For the Checker Part: the program will automatically parity check and it will print out that the data can accept or not. If FAIL, that means data cannot accept. If PASS that means data can accept.

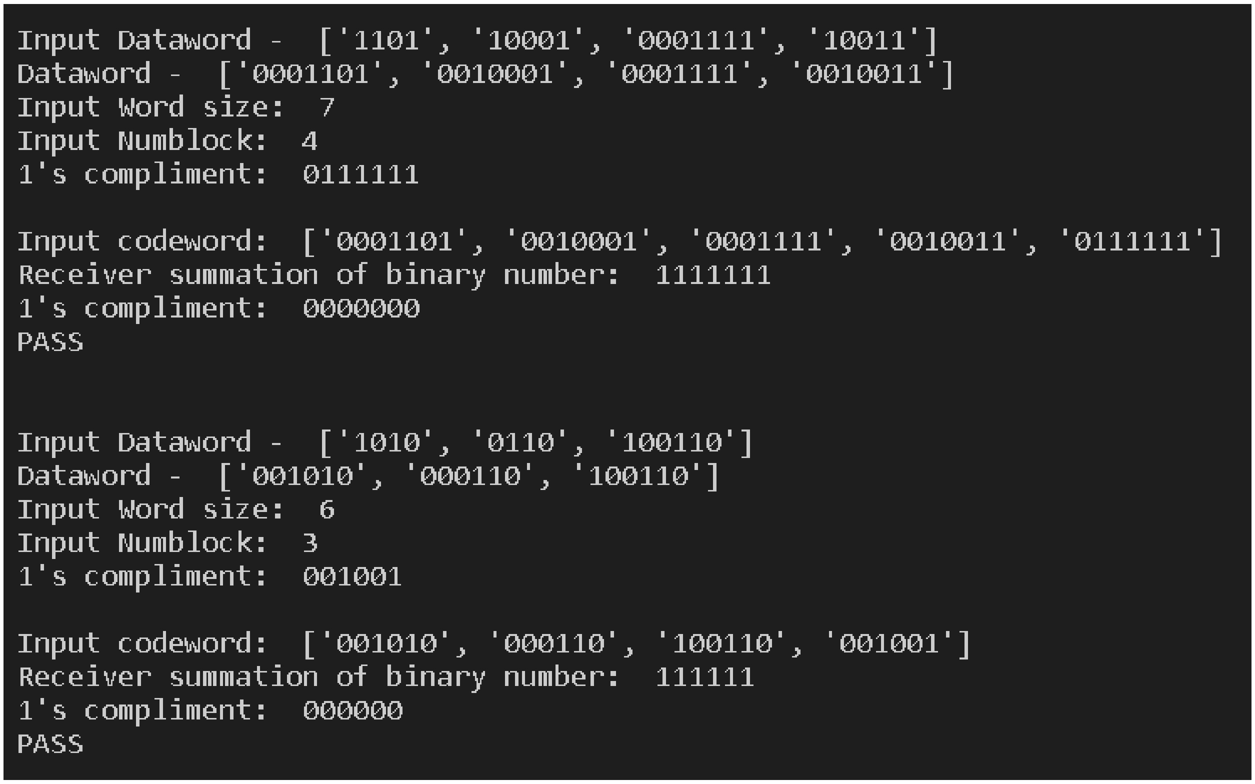
- How to run 2 -

- Uncomment at main
- Type your input at the main function including Dataword, Word Size, and number of blocks.
- Run “python checksum.py” in the terminal

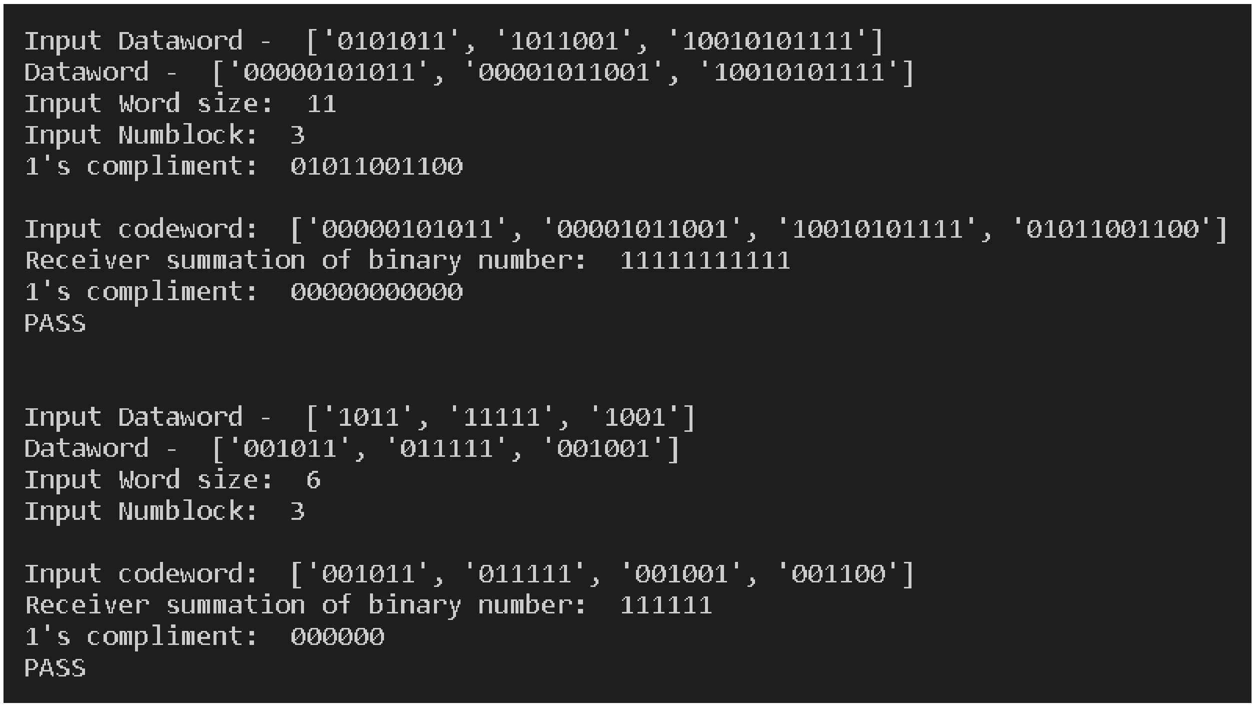
- Test Case -



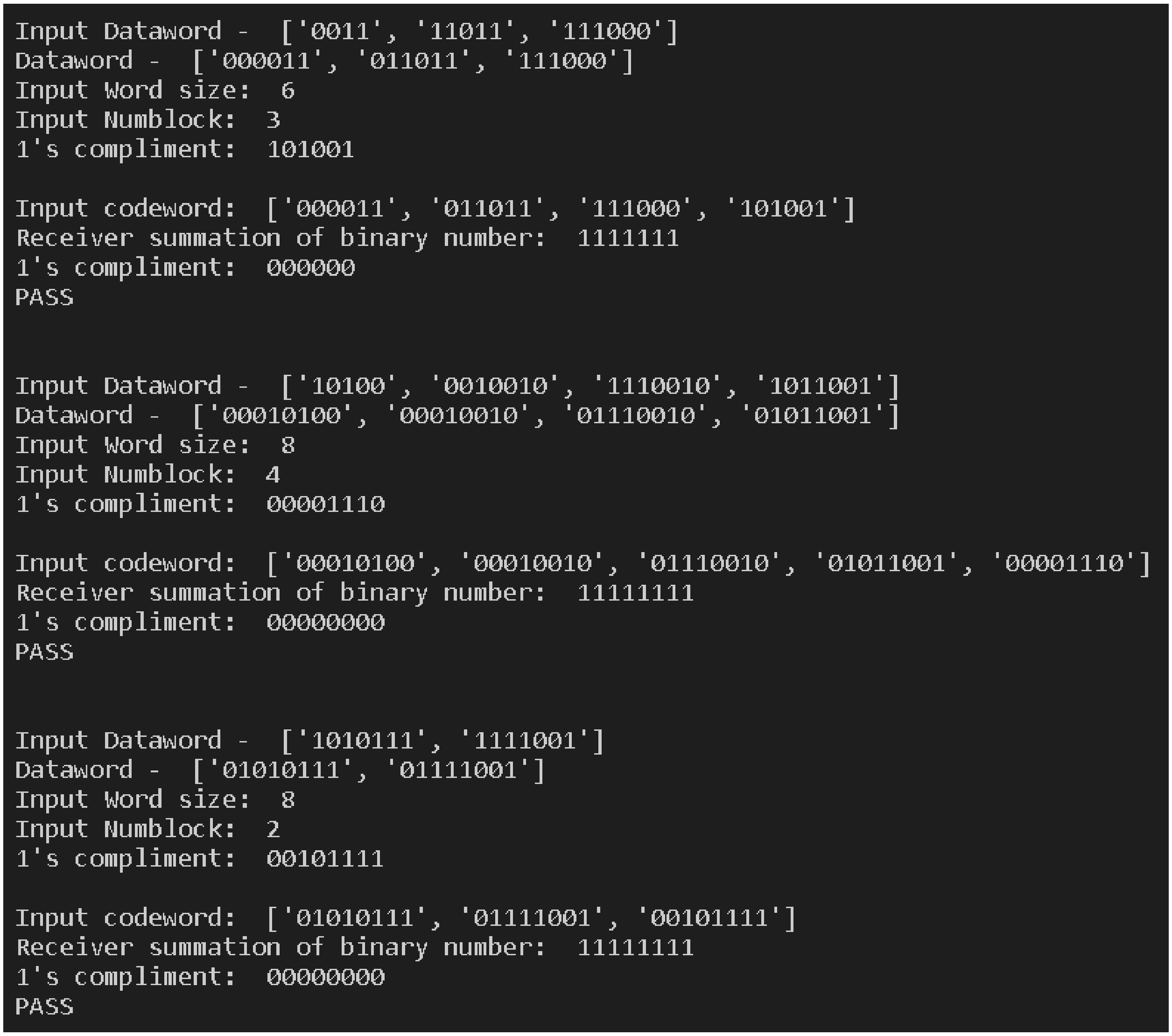
```
Input Dataword -  ['110011', '11001110', '1011101']
Dataword -  ['0000110011', '0011001110', '0001011101']
Input Word size:  10
Input Numblock:  3
1's compliment:  1010100001

Input codeword:  ['0000110011', '0011001110', '0001011101', '1010100001']
Receiver summation of binary number:  1111111111
1's compliment:  0000000000
PASS


Input Dataword -  ['10011', '01011', '100']
Dataword -  ['010011', '001011', '000100']
Input Word size:  6
Input Numblock:  3
1's compliment:  011101

Input codeword:  ['010011', '001011', '000100', '011101']
Receiver summation of binary number:  111111
1's compliment:  000000
PASS


Input Dataword -  ['11011', '00100', '1', '010101']
Dataword -  ['011011', '000100', '000001', '010101']
Input Word size:  6
Input Numblock:  4
1's compliment:  001010

Input codeword:  ['011011', '000100', '000001', '010101', '001010']
Receiver summation of binary number:  111111
1's compliment:  000000
PASS
```



```
Input Dataword -  ['1101', '10001', '0001111', '10011']
Dataword -  ['0001101', '0010001', '0001111', '0010011']
Input Word size:  7
Input Numblock:  4
1's compliment:  0111111

Input codeword:  ['0001101', '0010001', '0001111', '0010011', '0111111']
Receiver summation of binary number:  1111111
1's compliment:  0000000
PASS


Input Dataword -  ['1010', '0110', '100110']
Dataword -  ['001010', '000110', '100110']
Input Word size:  6
Input Numblock:  3
1's compliment:  001001

Input codeword:  ['001010', '000110', '100110', '001001']
Receiver summation of binary number:  111111
1's compliment:  000000
PASS
```

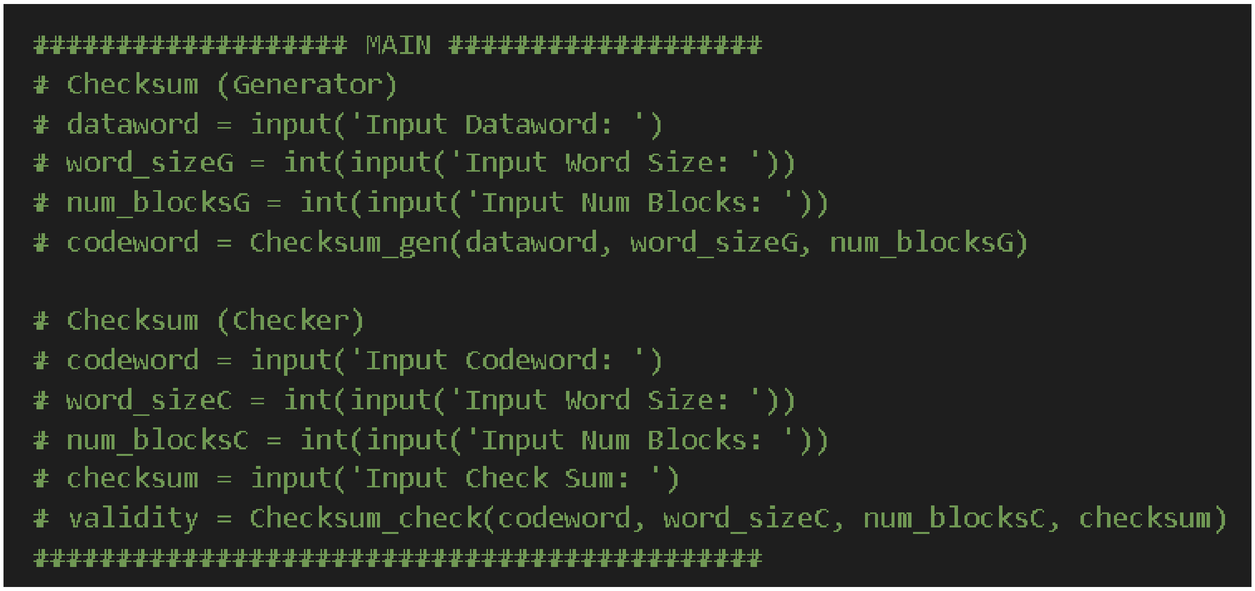


```
Input Dataword -  ['0101011', '1011001', '10010101111']
Dataword -  ['00000101011', '00001011001', '10010101111']
Input Word size:  11
Input Numblock:  3
1's compliment:  01011001100

Input codeword:  ['00000101011', '00001011001', '10010101111', '01011001100']
Receiver summation of binary number:  11111111111
1's compliment:  00000000000
PASS


Input Dataword -  ['1011', '11111', '1001']
Dataword -  ['001011', '011111', '001001']
Input Word size:  6
Input Numblock:  3

Input codeword:  ['001011', '011111', '001001', '001100']
Receiver summation of binary number:  111111
1's compliment:  000000
PASS
```



```
Input Dataword -  ['0011', '11011', '111000']
Dataword -  ['000011', '011011', '111000']
Input Word size:  6
Input Numblock:  3
1's compliment:  101001

Input codeword:  ['000011', '011011', '111000', '101001']
Receiver summation of binary number:  1111111
1's compliment:  000000
PASS


Input Dataword -  ['10100', '0010010', '1110010', '1011001']
Dataword -  ['00010100', '00010010', '01110010', '01011001']
Input Word size:  8
Input Numblock:  4
1's compliment:  00001110

Input codeword:  ['00010100', '00010010', '01110010', '01011001', '00001110']
Receiver summation of binary number:  11111111
1's compliment:  00000000
PASS


Input Dataword -  ['1010111', '1111001']
Dataword -  ['01010111', '01111001']
Input Word size:  8
Input Numblock:  2
1's compliment:  00101111

Input codeword:  ['01010111', '01111001', '00101111']
Receiver summation of binary number:  11111111
1's compliment:  00000000
PASS
```

- Uncomment and type your input at the main function including Dataword, Word Size, and number of blocks -

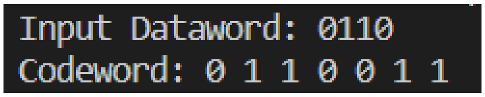


```
################### MAIN ###################
# Checksum (Generator)
# dataword = input('Input Dataword: ')
# word_sizeG = int(input('Input Word Size: '))
# num_blocksG = int(input('Input Num Blocks: '))
# codeword = Checksum_gen(dataword, word_sizeG, num_blocksG)

# Checksum (Checker)
# codeword = input('Input Codeword: ')
# word_sizeC = int(input('Input Word Size: '))
# num_blocksC = int(input('Input Num Blocks: '))
# checksum = input('Input Check Sum: ')
# validity = Checksum_check(codeword, word_sizeC, num_blocksC, checksum)
############################################
```

Hamming Code

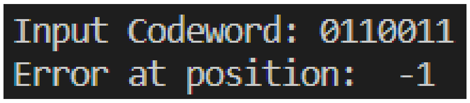
- How to run -

- Run “python hamming.py” in the terminal
- For the Generator Part: Put in a Dataword then the program will print out a codeword.



```
Input Dataword: 0110
Codeword: 0 1 1 0 0 1 1
```

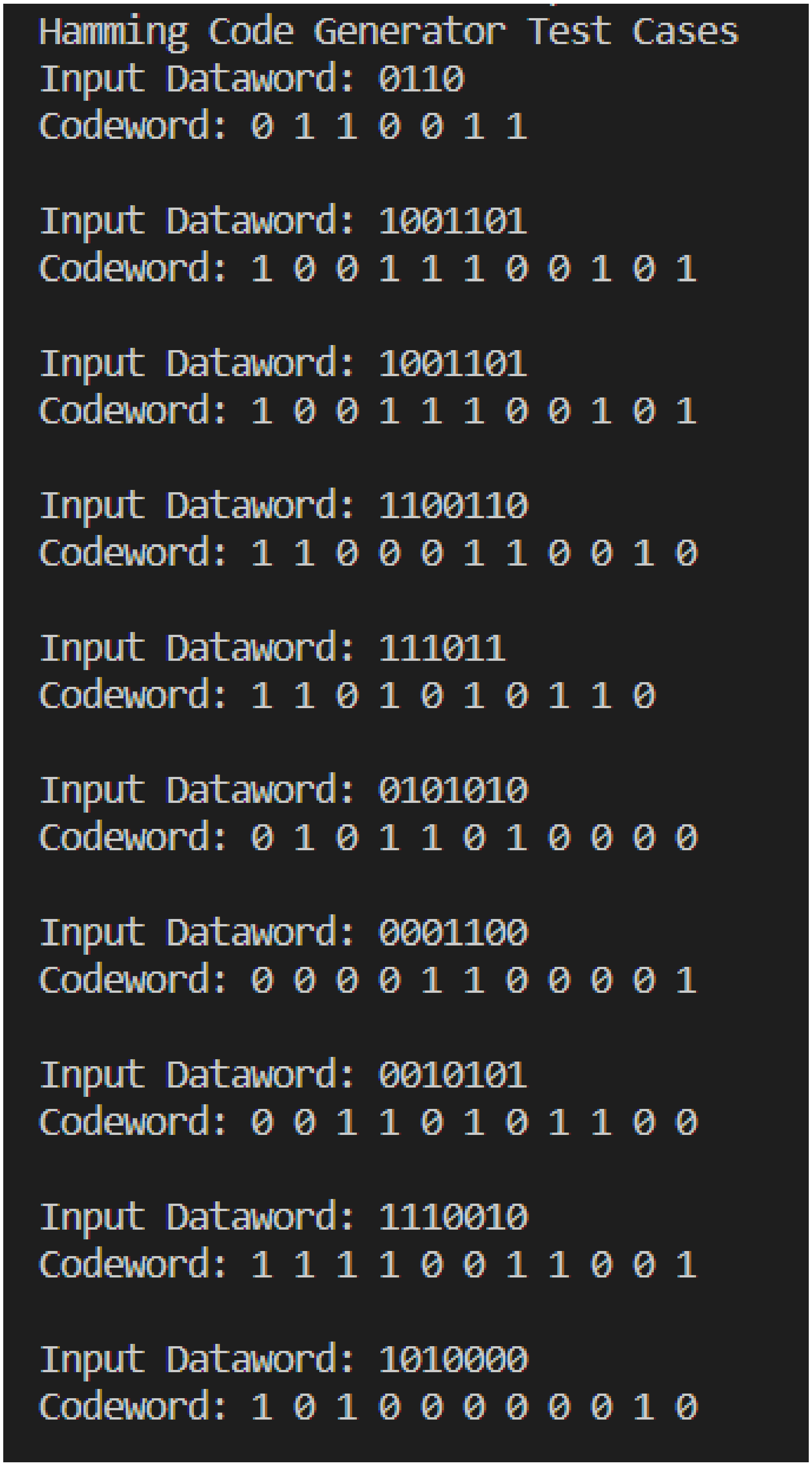
- For the Checker Part: Put in a Codeword then the program will print out an error position. But if the program returns -1 means there is no error.



```
Input Codeword: 0110011
Error at position:  -1
```

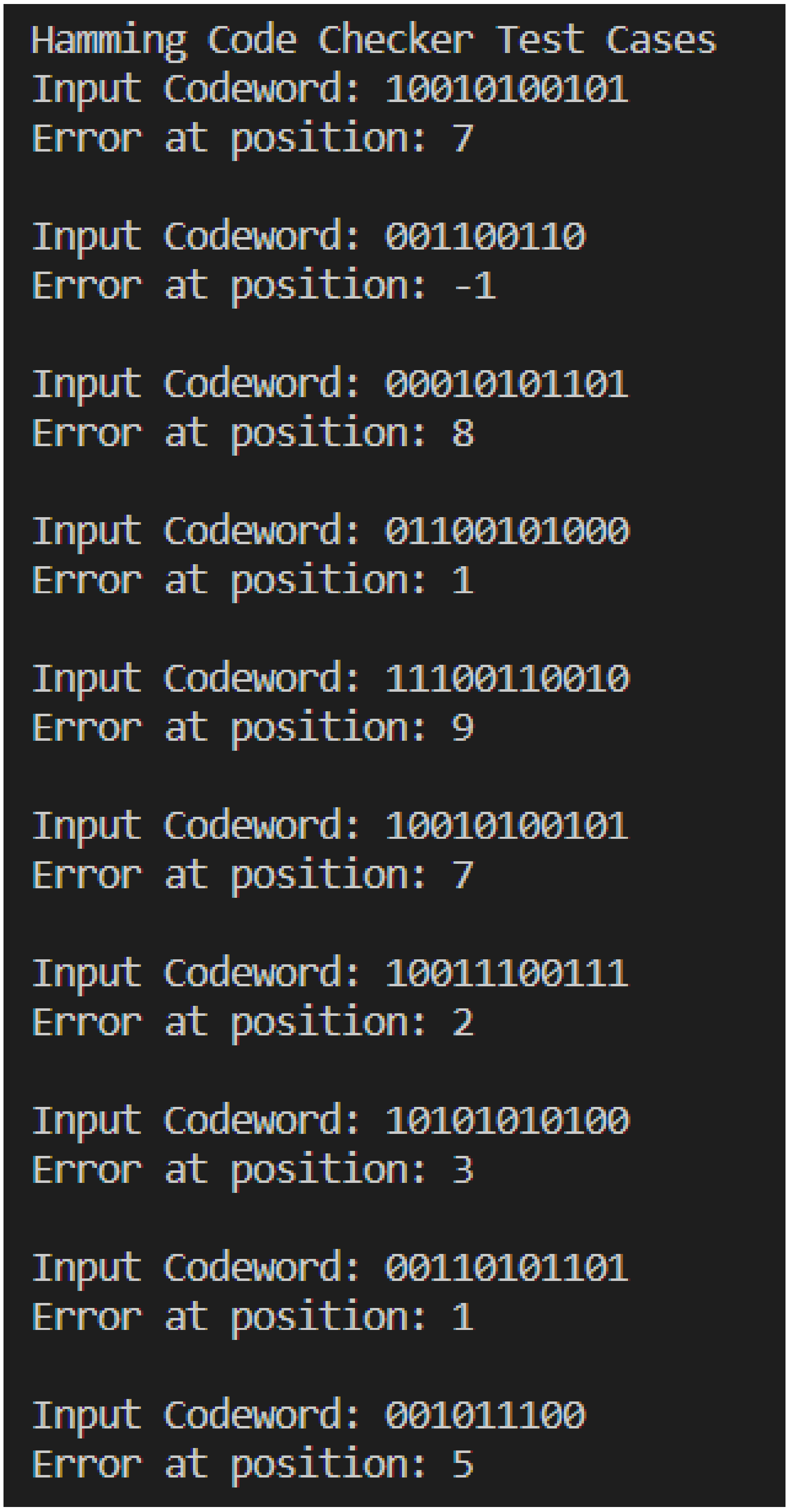
- Additional, run all of the test cases in a “python hamming_test.py”

- Test Cases For Generator -



```
Hamming Code Generator Test Cases
Input Dataword: 0110
Codeword: 0 1 1 0 0 1 1

Input Dataword: 1001101
Codeword: 1 0 0 1 1 1 0 0 1 0 1

Input Dataword: 1001101
Codeword: 1 0 0 1 1 1 0 0 1 0 1

Input Dataword: 1100110
Codeword: 1 1 0 0 0 1 1 0 0 1 0

Input Dataword: 111011
Codeword: 1 1 0 1 0 1 0 1 1 0

Input Dataword: 0101010
Codeword: 0 1 0 1 1 0 1 0 0 0 0

Input Dataword: 0001100
Codeword: 0 0 0 0 1 1 0 0 0 0 1

Input Dataword: 0010101
Codeword: 0 0 1 1 0 1 0 1 1 0 0

Input Dataword: 1110010
Codeword: 1 1 1 1 0 0 1 1 0 0 1

Input Dataword: 1010000
Codeword: 1 0 1 0 0 0 0 0 0 1 0
```

- Test Cases For Checker -



```
Hamming Code Checker Test Cases
Input Codeword: 10010100101
Error at position: 7

Input Codeword: 001100110
Error at position: -1

Input Codeword: 00010101101
Error at position: 8

Input Codeword: 01100101000
Error at position: 1

Input Codeword: 11100110010
Error at position: 9

Input Codeword: 10010100101
Error at position: 7

Input Codeword: 10011100111
Error at position: 2

Input Codeword: 10101010100
Error at position: 3

Input Codeword: 00110101101
Error at position: 1

Input Codeword: 001011100
Error at position: 5
```

References

https://www.geeksforgeeks.org/python-program-to-add-two-binary-numbers/

https://docs.python.org/3/tutorial/introduction.html

https://www.geeksforgeeks.org/1s-2s-complement-binary-number/

https://www.geeksforgeeks.org/hamming-code-implementation-in-c-cpp/

https://www.geeksforgeeks.org/hamming-code-implementation-in-python/#:~:text=Hamming%20code%20is%20a%20set,Hamming%20for%20error%20correction

https://www.geeksforgeeks.org/modulo-2-binary-division/