Assignment – 1

Name : Hem Gabhawala
Id : 23000858

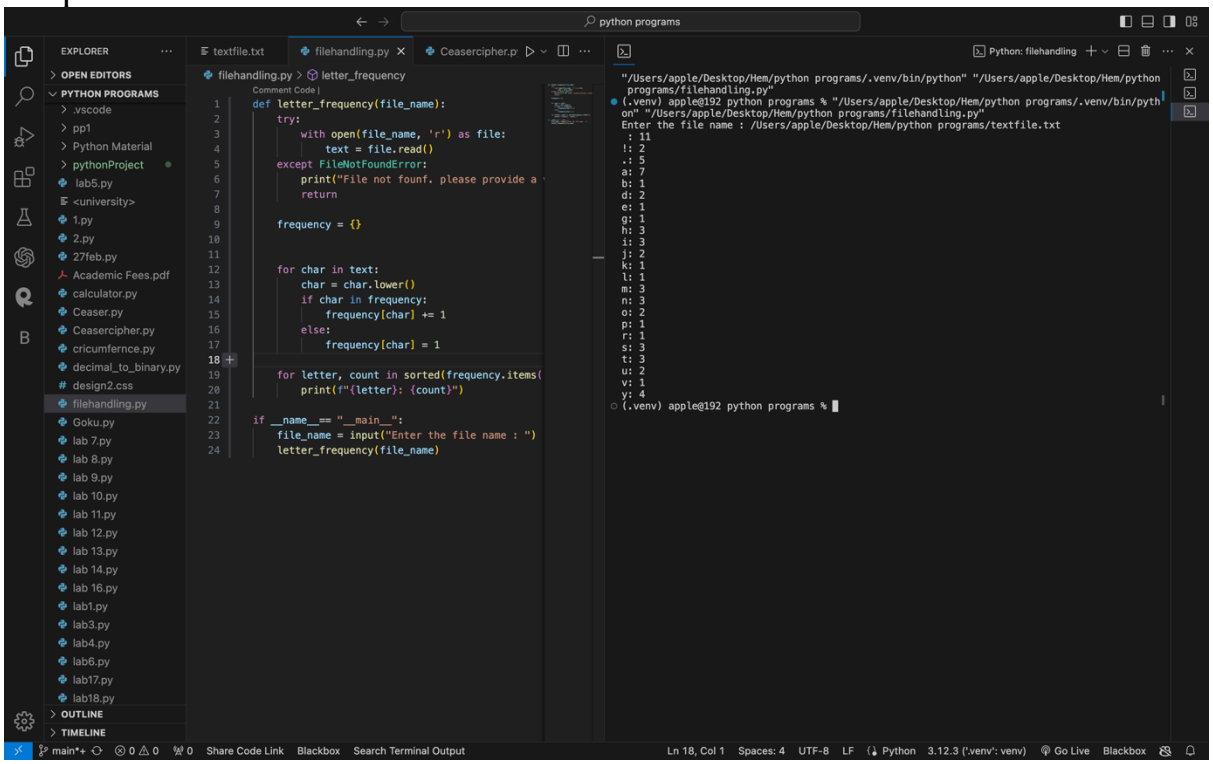
Q1

(a) Accept file name from the user and read text from the given file. Find the frequency of each letter.
(b) Given the above information and a cipher text, which has been encrypted by
Caesar Cipher encryption algorithm, decrypt the given cipher text to get back the
plain text

Code :

```
def letter_frequency(file_name):
    try:
        with open(file_name, 'r') as file:
            text = file.read()
    except FileNotFoundError:
        print("File not founf. please provide a value")
        return

    frequency = {}


    for char in text:
        char = char.lower()
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1

    for letter, count in sorted(frequency.items()):
        print(f"{letter}: {count}")

if __name__== "__main__":
    file_name = input("Enter the file name : ")
    letter_frequency(file_name)
```

Output:



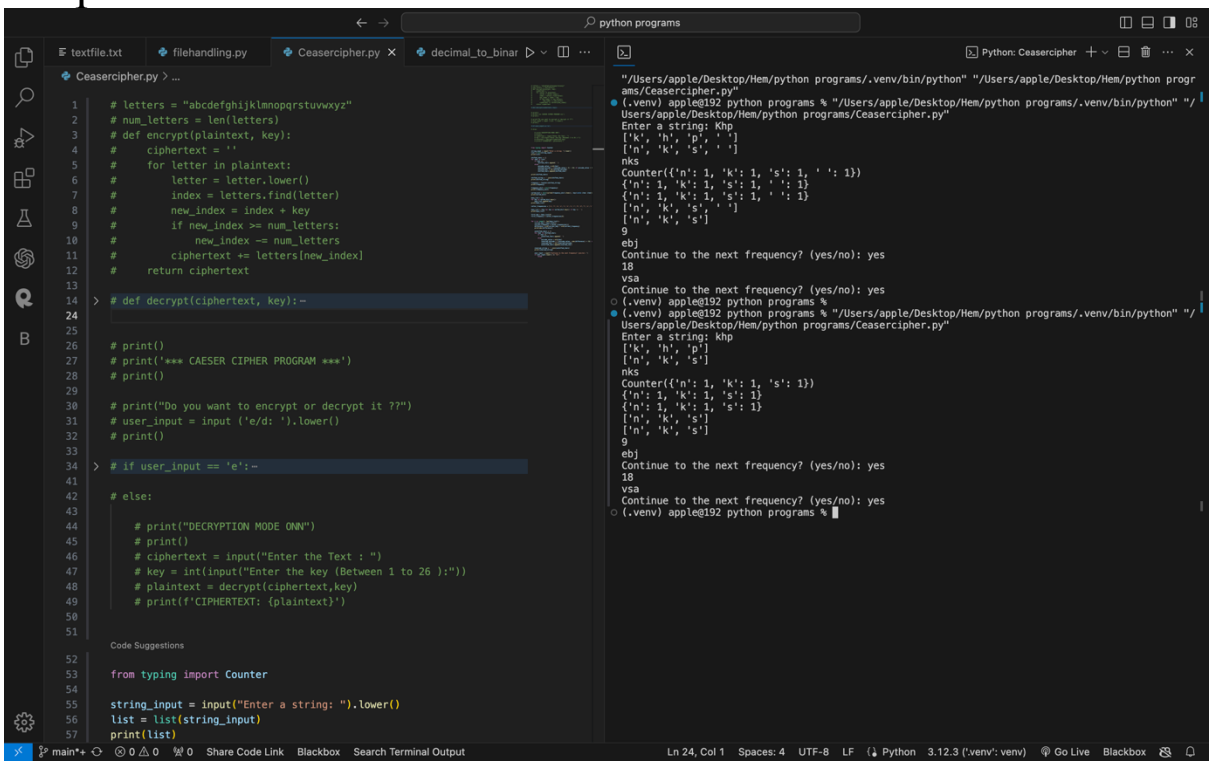
Code :

Method:1

```
from typing import Counter

string_input = input("Enter a string: ").lower()
list = list(string_input)
print(list)

shifted_chars = []
for char in list:
    if char == ' ':
        shifted_chars.append(' ')
    else:
        unicode_value = ord(char)
        shifted_unicode = ((unicode_value + 3) - 26) if unicode_value + 3 > 122
else unicode_value + 3
        shifted_char = chr(shifted_unicode)
        shifted_chars.append(shifted_char)

print(shifted_chars)

shifted_string = ''.join(shifted_chars)
print(shifted_string)
```

```
frequency = Counter(shifted_string)
print(frequency)

frequency_dict = dict(frequency)
print(frequency_dict)

sorted_dict = dict(sorted(frequency_dict.items(), key=lambda item: item[1],
reverse=True))
print(sorted_dict)

keys_list = []
for key in sorted_dict.keys():
    keys_list.append(key)
print(keys_list)

Letter_frequencies =
['e','t','a','o','i','n','s','r','h','d','l','u','c','m','f','y','w','g','p','b','v
','k','x','q','j','z']

keys_list = [key for key in sorted_dict.keys() if key != ' ']
print(keys_list)

first_key = keys_list[0]
first_frequency = Letter_frequencies[0]


for i in range(1, len(keys_list)):
    current_key = keys_list[i]
    current_frequency = Letter_frequencies[i]
    difference = ord(current_key) - ord(current_frequency)
    print(abs(difference))

    unshifted_chars = []
    for char in shifted_chars:
        if char == ' ':
            unshifted_chars.append(' ')
        else:
            unicode_value = ord(char)
            reversed_unicode = ((unicode_value - abs(difference)) + 26) if
unicode_value - abs(difference) < 97 else unicode_value - abs(difference);
            reversed_char = chr(reversed_unicode)
            unshifted_chars.append(reversed_char)

    reversed_string = ''.join(unshifted_chars)
    print(reversed_string)

    user_input = input("Continue to the next frequency? (yes/no): ")
    if user_input.lower() == "no":
        break
```

Output:



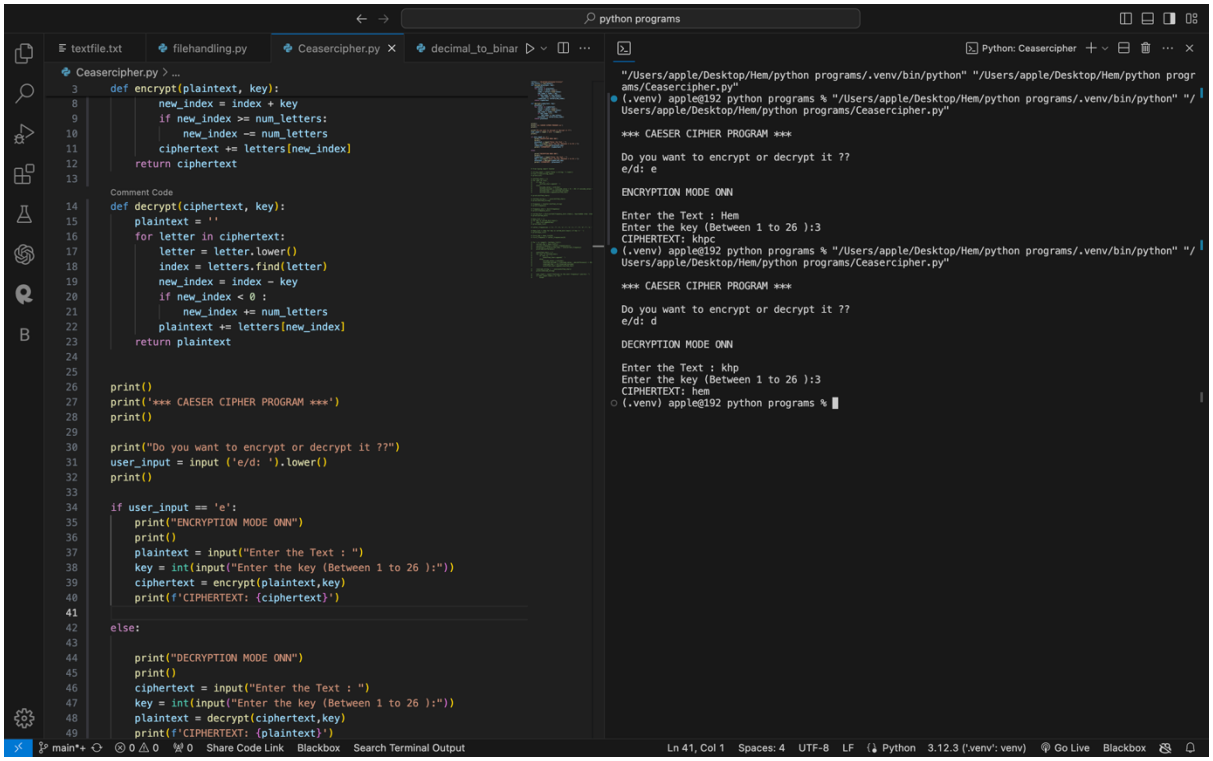
Method 2:

Code:

```
letters = "abcdefghijklmnopqrstuvwxyz"
num_letters = len(letters)
def encrypt(plaintext, key):
    ciphertext = ''
    for letter in plaintext:
        letter = letter.lower()
        index = letters.find(letter)
        new_index = index + key
        if new_index >= num_letters:
            new_index -= num_letters
        ciphertext += letters[new_index]
    return ciphertext

def decrypt(ciphertext, key):
    plaintext = ''
    for letter in ciphertext:
        letter = letter.lower()
        index = letters.find(letter)
        new_index = index - key
        if new_index < 0 :
```

```
            new_index += num_letters
        plaintext += letters[new_index]
    return plaintext


print()
print('*** CAESER CIPHER PROGRAM ***')
print()

print("Do you want to encrypt or decrypt it ??")
user_input = input ('e/d: ').lower()
print()

if user_input == 'e':
    print("ENCRYPTION MODE ONN")
    print()
    plaintext = input("Enter the Text : ")
    key = int(input("Enter the key (Between 1 to 26 ):"))
    ciphertext = encrypt(plaintext,key)
    print(f'CIPHERTEXT: {ciphertext}')

else:

    print("DECRYPTION MODE ONN")
    print()
    ciphertext = input("Enter the Text : ")
    key = int(input("Enter the key (Between 1 to 26 ):"))
    plaintext = decrypt(ciphertext,key)
    print(f'CIPHERTEXT: {plaintext}')
```

Output :



Q2:

(a) Accept a Decimal number from the user, the number may be positive or negative and it may have 0 or more decimal digits. Convert the given number into the
corresponding Binary number.

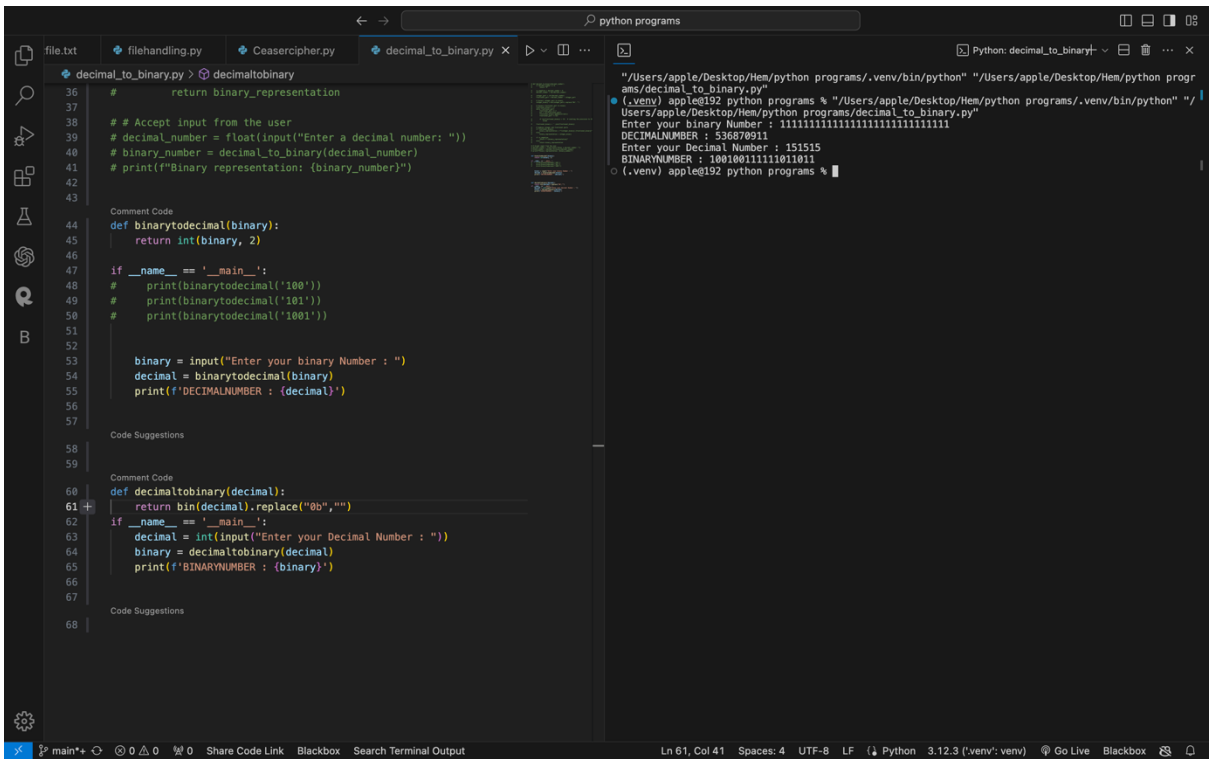
(b) Accept a positive Binary number from the user. The number may have integral as
well as fractional parts. Convert the given Binary number into the corresponding
Octal number.

Code:

```
def binarytodecimal(binary):
    return int(binary, 2)

if __name__ == '__main__':
#     print(binarytodecimal('100'))
#     print(binarytodecimal('101'))
#     print(binarytodecimal('1001'))


    binary = input("Enter your binary Number : ")
```

```
    decimal = binarytodecimal(binary)
    print(f'DECIMALNUMBER : {decimal}')



def decimaltobinary(decimal):
    return bin(decimal).replace("0b","")
if __name__ == '__main__':
    decimal = int(input("Enter your Decimal Number : "))
    binary = decimaltobinary(decimal)
    print(f'BINARYNUMBER : {binary}')
```

Output :



Code :

```
def binary_to_octal():
```

```
    binary_num = input("Enter a positive binary number: ")

    # Check if the input is a valid binary number
    if not set(binary_num).issubset({'0', '1', '.'}):
        print("Invalid binary number. Please enter a number consisting of 0s, 1s,
and an optional decimal point.")
        return

    # Split the binary number into integral and fractional parts
    parts = binary_num.split('.')

    # Convert the integral part to decimal
    integral_part = int(parts[0], 2)

    # Convert the fractional part to decimal
    fractional_part = 0
    if len(parts) > 1:
        fractional_part = sum([int(bit) * 2**(-i) for i, bit in enumerate(parts[1],
start=1)])

    # Combine the integral and fractional parts to get the decimal number
    decimal_num = integral_part + fractional_part

    # Convert the decimal number to octal
    octal_num = oct(int(decimal_num)).replace("0o", "")

    print("The corresponding octal number is:", octal_num)

binary_to_octal()
```

Output:

```
  "/Users/apple/Desktop/Hem/python programs/.venv/bin/python" "/Users/apple/Desktop/Hem/python progr
  ams/decimal_to_binary.py"
● (.venv) apple@192 python programs % "/Users/apple/Desktop/Hem/python programs/.venv/bin/python" "/
  Users/apple/Desktop/Hem/python programs/decimal_to_binary.py"
  Enter a positive binary number: 1111
  The corresponding octal number is: 17
○ (.venv) apple@192 python programs %
```

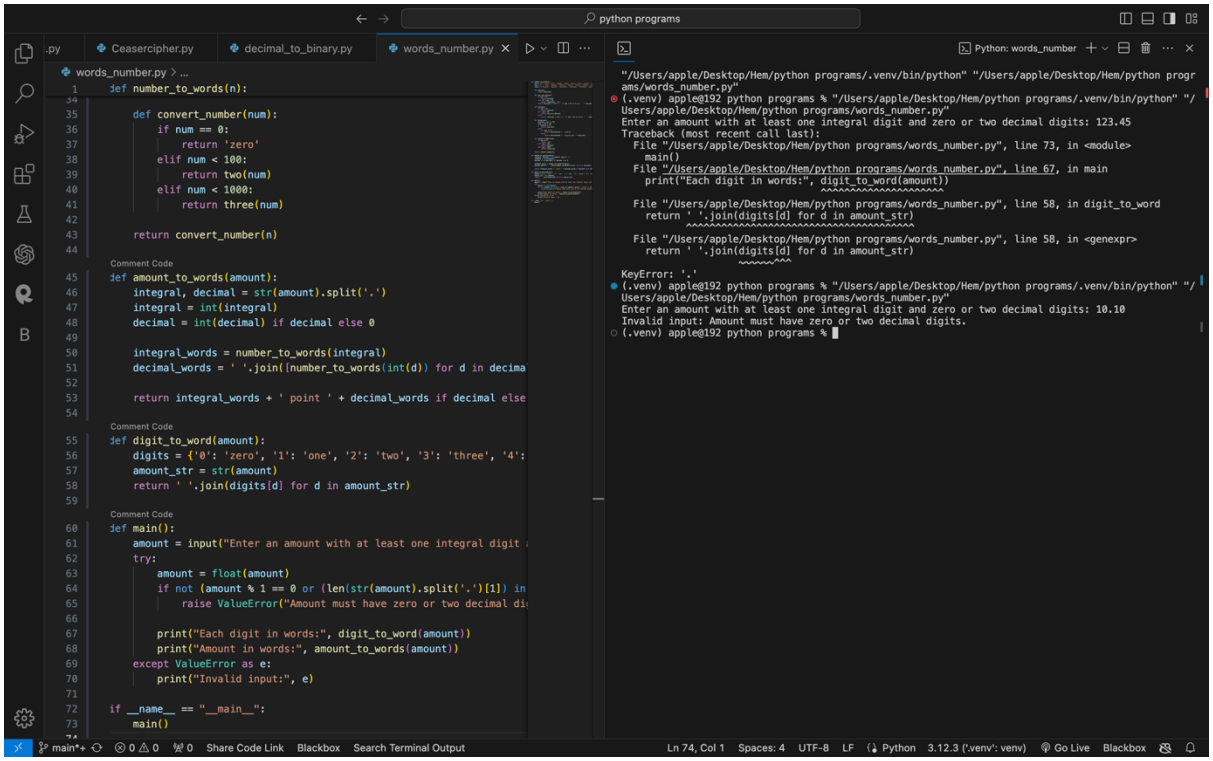
Q3 –

Accept an amount (from the user) having at least one integral digit and zero or two
decimal digits. Convert the given amount in words: (a) Each digit is converted to
corresponding word, (b) Entire integral part of the number is converted to words and
entire decimal part is also converted to words.

Code :
```
def number_to_words(n):
    ones = ["", "one", "two", "three", "four", "five", "six", "seven", "eight",
"nine"]
    tens = ["", "ten", "twenty", "thirty", "forty", "fifty", "sixty", "seventy",
"eighty", "ninety"]
    teens = ["eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen",
"seventeen", "eighteen", "nineteen"]

    def one(num):
        return ones[num]

    def two_less_20(num):
        if num < 10:
            return one(num)
        elif 10 < num < 20:
            return teens[num - 11]
        return tens[num // 10] + ('' if num % 10 == 0 else ' ' + one(num % 10))

    def two(num):
        if num < 10:
            return one(num)
        elif num < 20:
            return two_less_20(num)
        else:
            return tens[num // 10] + ('' if num % 10 == 0 else ' ' + one(num % 10))

    def three(num):
        hundred = num // 100
        rest = num % 100
        if hundred == 0:
            return two(rest)
        else:
            if rest == 0:
                return one(hundred) + ' hundred'
            else:
                return one(hundred) + ' hundred and ' + two(rest)

    def convert_number(num):
        if num == 0:
            return 'zero'
```

```
        elif num < 100:
            return two(num)
        elif num < 1000:
            return three(num)

    return convert_number(n)

def amount_to_words(amount):
    integral, decimal = str(amount).split('.')
    integral = int(integral)
    decimal = int(decimal) if decimal else 0

    integral_words = number_to_words(integral)
    decimal_words = ' '.join([number_to_words(int(d)) for d in decimal])

    return integral_words + ' point ' + decimal_words if decimal else
integral_words

def digit_to_word(amount):
    digits = {'0': 'zero', '1': 'one', '2': 'two', '3': 'three', '4': 'four', '5':
'five', '6': 'six', '7': 'seven', '8': 'eight', '9': 'nine'}
    amount_str = str(amount)
    return ' '.join(digits[d] for d in amount_str)

def main():
    amount = input("Enter an amount with at least one integral digit and zero or
two decimal digits: ")
    try:
        amount = float(amount)
        if not (amount % 1 == 0 or (len(str(amount).split('.')[1]) in [0, 2])):
            raise ValueError("Amount must have zero or two decimal digits.")

        print("Each digit in words:", digit_to_word(amount))
        print("Amount in words:", amount_to_words(amount))
    except ValueError as e:
        print("Invalid input:", e)

if __name__ == "__main__":
    main()
```

Output:



Q4 –

For a given positive integer, (a) find its divisors, (b) find its prime factors, and (c)
find the 5 co-prime numbers of the given integer.

Code :

```
def find_divisors(n):
    divisors = [i for i in range(1, n + 1) if n % i == 0]
    return divisors

def find_prime_factors(n):
    i = 2
    prime_factors = []
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            prime_factors.append(i)
    if n > 1:
```

```
        prime_factors.append(n)
    return prime_factors


def find_co_primes(n):
    co_primes = []
    for i in range(1, n * 2):
        if gcd(i, n) == 1:
            co_primes.append(i)
        if len(co_primes) == 5:
            break
    return co_primes

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a



n = int(input("Enter the Number : "))
Divisors = find_divisors(n)
PrimeFactors = find_prime_factors(n)
CoPrimeNumbers = find_co_primes(n)

print(f'DIVISORS : {Divisors}')
print(f'PRIMEFACTORS : {PrimeFactors}')
print(f'CO-PRIME NUMBERS : {CoPrimeNumbers}')
```

Output :

```
"/Users/apple/Desktop/Hem/python programs/.venv/bin/python" "/Users/apple/Desktop/Hem/python
ams/Q4.py"
(.venv) apple@192 python programs % "/Users/apple/Desktop/Hem/python programs/.venv/bin/pyth
on" "/Users/apple/Desktop/Hem/python programs/Q4.py"
Enter the Number : 123
DIVISORS : [1, 3, 41, 123]
PRIMEFACTORS : [3, 41]
CO-PRIME NUMBERS : [1, 2, 4, 5, 7]
(.venv) apple@192 python programs %
```

Q5 –
(a)In a 3 x 3 matrix arrange numbers 1, 2, 3 such that each row and each column
has different numbers.
(b) Make it general for n x n list for the given n unique numbers.
(c) First write a solution for 9 x 9 matrix with above-mentioned conditions. Each set
of 3 rows will have three 3 x 3 matrices, resulting in a total of nine 3 x 3 matrices.
Check whether each of nine 3 x 3 matrices have different numbers in each row and
each column.

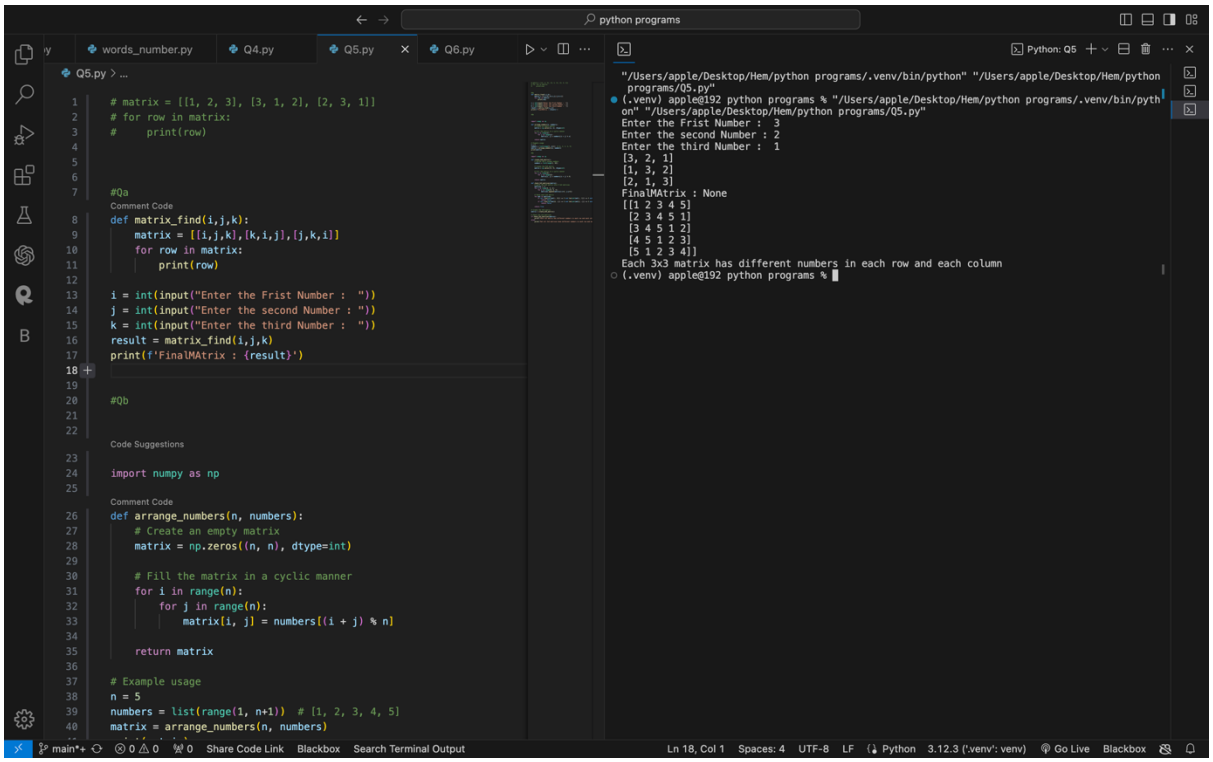
Code :

```
# matrix = [[1, 2, 3], [3, 1, 2], [2, 3, 1]]
# for row in matrix:
#     print(row)



#Qa
def matrix_find(i,j,k):
    matrix = [[i,j,k],[k,i,j],[j,k,i]]
    for row in matrix:
        print(row)

i = int(input("Enter the Frist Number :  "))
j = int(input("Enter the second Number : "))
k = int(input("Enter the third Number :  "))
result = matrix_find(i,j,k)
print(f'FinalMAtrix : {result}')


#Qb



import numpy as np

def arrange_numbers(n, numbers):
    # Create an empty matrix
    matrix = np.zeros((n, n), dtype=int)

    # Fill the matrix in a cyclic manner
    for i in range(n):
        for j in range(n):
```

```
            matrix[i, j] = numbers[(i + j) % n]

    return matrix

# Example usage
n = 5
numbers = list(range(1, n+1))  # [1, 2, 3, 4, 5]
matrix = arrange_numbers(n, numbers)
print(matrix)

#Qc

import numpy as np

def create_9x9_matrix():
    # Define the 9 unique numbers
    numbers = list(range(1, 10))

    # Create the 9x9 matrix
    matrix = np.zeros((9, 9), dtype=int)

    # Fill the matrix in a cyclic manner
    for i in range(9):
        for j in range(9):
            matrix[i, j] = numbers[(i + j) % 9]

    return matrix

def check_3x3_matrices(matrix):
    # Split the 9x9 matrix into 9 3x3 matrices
    matrices = []
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            matrices.append(matrix[i:i+3, j:j+3])

    # Check each 3x3 matrix
    for mat in matrices:
        if not (len(set(mat[:, 0])) == 3 and len(set(mat[:, 1])) == 3 and
len(set(mat[:, 2])) == 3):
            return False
        if not (len(set(mat[0, :])) == 3 and len(set(mat[1, :])) == 3 and
len(set(mat[2, :])) == 3):
            return False

    return True

# Create the 9x9 matrix
matrix = create_9x9_matrix()

# Check the 3x3 matrices
if check_3x3_matrices(matrix):
```

```
    print("Each 3x3 matrix has different numbers in each row and each column")
else:
    print("Not all 3x3 matrices have different numbers in each row and each
column")
```

Output:



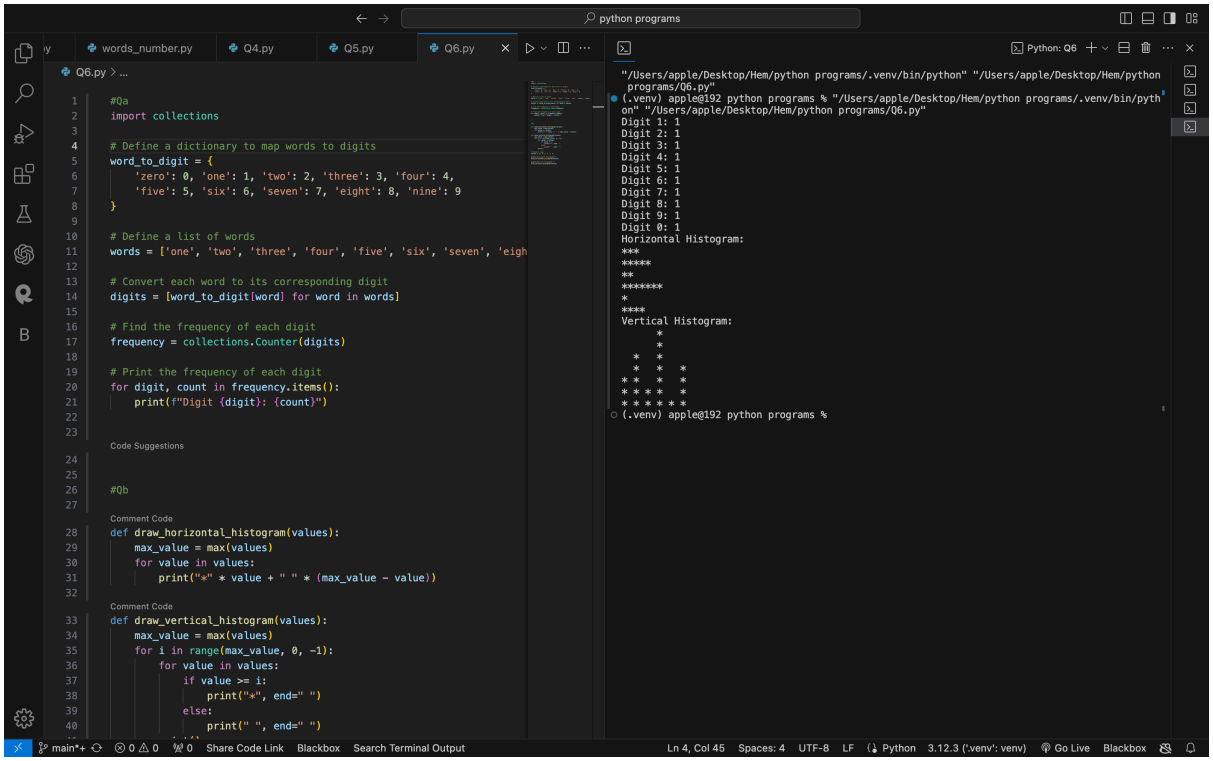
Q6-
(a) Convert each word (for a digit) to its corresponding digit and find frequency of
each digit.
(b) Draw two histograms using '*' symbols – one horizontal and one vertical.

Code :

```
#Qa
import collections

# Define a dictionary to map words to digits
word_to_digit = {
    'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4,
```

```
    'five': 5, 'six': 6, 'seven': 7, 'eight': 8, 'nine': 9
}

# Define a list of words
words = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine',
'zero']

# Convert each word to its corresponding digit
digits = [word_to_digit[word] for word in words]

# Find the frequency of each digit
frequency = collections.Counter(digits)

# Print the frequency of each digit
for digit, count in frequency.items():
    print(f"Digit {digit}: {count}")




#Qb

def draw_horizontal_histogram(values):
    max_value = max(values)
    for value in values:
        print("*" * value + " " * (max_value - value))

def draw_vertical_histogram(values):
    max_value = max(values)
    for i in range(max_value, 0, -1):
        for value in values:
            if value >= i:
                print("*", end=" ")
            else:
                print(" ", end=" ")
        print()

# Example usage
values = [3, 5, 2, 7, 1, 4]

print("Horizontal Histogram:")
draw_horizontal_histogram(values)

print("Vertical Histogram:")
draw_vertical_histogram(values)
```

Output:


python programs
words_number.py
Q4.py
Q5.py
Q6.py
Q6.py > ...
#Qa
import collections
# Define a dictionary to map words to digits
word_to_digit = {
'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4,
'five': 5, 'six': 6, 'seven': 7, 'eight': 8, 'nine': 9
}
# Define a list of words
words = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eigh
# Convert each word to its corresponding digit
digits = [word_to_digit[word] for word in words]
# Find the frequency of each digit
frequency = collections.Counter(digits)
# Print the frequency of each digit
for digit, count in frequency.items():
print(f"Digit {digit}: {count}")
Code Suggestions
#Qb
Comment Code
def draw_horizontal_histogram(values):
max_value = max(values)
for value in values:
print("*" * value + " " * (max_value - value))
Comment Code
def draw_vertical_histogram(values):
max_value = max(values)
for i in range(max_value, 0, -1):
for value in values:
if value >= i:
print("*", end=" ")
else:
print(" ", end=" ")
Python: Q6
"/Users/apple/Desktop/Hem/python programs/.venv/bin/python" "/Users/apple/Desktop/Hem/python programs/Q6.py"
(.venv) apple@192 python programs % "/Users/apple/Desktop/Hem/python programs/.venv/bin/python" "/Users/apple/Desktop/Hem/python programs/Q6.py"
Digit 1: 1
Digit 2: 1
Digit 3: 1
Digit 4: 1
Digit 5: 1
Digit 6: 1
Digit 7: 1
Digit 8: 1
Digit 9: 1
Digit 0: 1
Horizontal Histogram:
***
*****
**
****
*
****
Vertical Histogram:
(.venv) apple@192 python programs %
main*+
0 0
0
Share Code Link
Blackbox
Search Terminal Output
Ln 4, Col 45
Spaces: 4
UTF-8
LF
Python
3.12.3 ('.venv': venv)
Go Live
Blackbox