**Assignment -1**

**G SAI DEEPAK**

**192311432**

**CSA 5196**

* **Cryptography and security network for mobile applications**

**1.**

def caesar\_cipher(text, k, encrypt=True):

    result = ""

    for char in text:

        if char.isalpha():

            shift = k if encrypt else -k

            base = ord('A') if char.isupper() else ord('a')

            result += chr((ord(char) - base + shift) % 26 + base)

        else:

            result += char  # Non-alphabetic characters remain unchanged

    return result

# Example usage

text = input("Enter text: ")

k = int(input("Enter shift (1-25): "))

while not (1 <= k <= 25):

    print("Shift value must be between 1 and 25.")

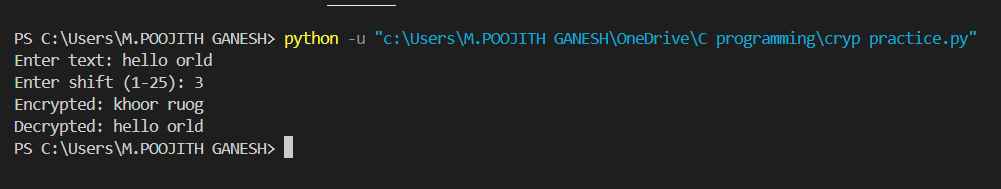
    k = int(input("Enter shift (1-25): "))

encrypted\_text = caesar\_cipher(text, k, encrypt=True)

print(f"Encrypted: {encrypted\_text}")

decrypted\_text = caesar\_cipher(encrypted\_text, k, encrypt=False)

print(f"Decrypted: {decrypted\_text}")



**2.**

import random

import string

# Function to generate a random substitution key

def generate\_key():

letters = list(string.ascii\_lowercase)

shuffled = letters[:]

random.shuffle(shuffled)

return dict(zip(letters, shuffled))

# Function to encrypt using the monoalphabetic substitution cipher

def encrypt(plaintext, key):

ciphertext = ""

for char in plaintext.lower():

if char in key:

ciphertext += key[char]

else:

ciphertext += char # Keep non-alphabetic characters unchanged

return ciphertext

# Function to decrypt using the monoalphabetic substitution cipher

def decrypt(ciphertext, key):

reverse\_key = {v: k for k, v in key.items()} # Reverse the key mapping

plaintext = ""

for char in ciphertext.lower():

if char in reverse\_key:

plaintext += reverse\_key[char]

else:

plaintext += char # Keep non-alphabetic characters unchanged

return plaintext

# Generate a random substitution key

key = generate\_key()

# Example plaintext

plaintext = "hello world"

ciphertext = encrypt(plaintext, key)

decrypted\_text = decrypt(ciphertext, key)

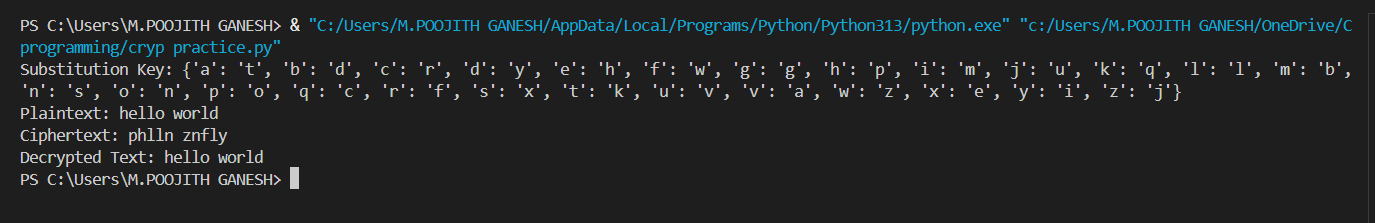
# Output results

print("Substitution Key:", key)

print("Plaintext:", plaintext)

print("Ciphertext:", ciphertext)

print("Decrypted Text:", decrypted\_text)



**3.**

import numpy as np

# Create Playfair matrix

def create\_playfair\_matrix(key):

key = "".join(dict.fromkeys(key.upper().replace("J", "I"))) # Remove duplicates, replace J with I

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

matrix = [c for c in key + "".join(c for c in alphabet if c not in key)]

return np.array(matrix).reshape(5, 5)

# Find position of a letter in the matrix

def find\_position(matrix, letter):

pos = np.where(matrix == letter)

return pos[0][0], pos[1][0]

# Encrypt a digraph using Playfair rules

def encrypt\_digraph(matrix, digraph):

a, b = digraph

row1, col1 = find\_position(matrix, a)

row2, col2 = find\_position(matrix, b)

if row1 == row2: # Same row

return matrix[row1, (col1 + 1) % 5] + matrix[row2, (col2 + 1) % 5]

elif col1 == col2: # Same column

return matrix[(row1 + 1) % 5, col1] + matrix[(row2 + 1) % 5, col2]

else: # Rectangle swap

return matrix[row1, col2] + matrix[row2, col1]

# Encrypt message

def playfair\_encrypt(message, key):

matrix = create\_playfair\_matrix(key)

message = message.upper().replace("J", "I").replace(" ", "")

if len(message) % 2 != 0:

message += "X" # Padding if odd length

ciphertext = "".join(encrypt\_digraph(matrix, message[i:i+2]) for i in range(0, len(message), 2))

return ciphertext

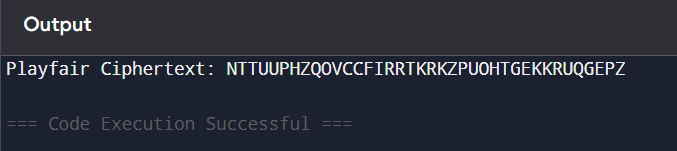
# Example usage

key = "CIPHER"

plaintext = "MUSTSEEYOUOVERCADOGANWESTCOMINGATONCE"

ciphertext = playfair\_encrypt(plaintext, key)

print("Playfair Ciphertext:", ciphertext)



**4.**

from collections import Counter

def frequency\_analysis(ciphertext):

    freq = Counter(ciphertext)

    sorted\_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

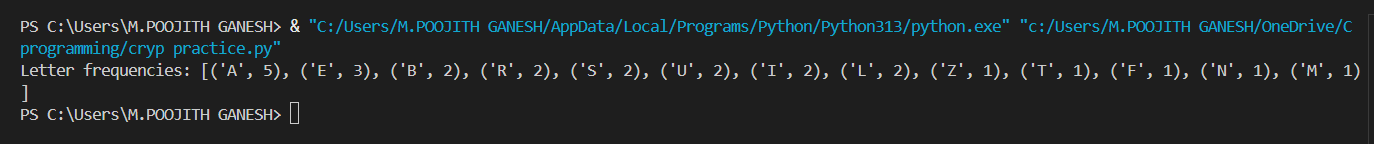
    return sorted\_freq

# Example usage

ciphertext = "ZEBRAS ARE BEAUTIFUL ANIMALS"

freq\_list = frequency\_analysis(ciphertext.replace(" ", ""))

print("Letter frequencies:", freq\_list)



**5.**

import string

def generate\_cipher\_alphabet(keyword):

    keyword = "".join(dict.fromkeys(keyword.upper()))  # Remove duplicates

    alphabet = string.ascii\_uppercase.replace("J", "")  # Playfair-style alphabet

    cipher\_alphabet = keyword + "".join(c for c in alphabet if c not in keyword)

    return dict(zip(string.ascii\_uppercase, cipher\_alphabet))

def monoalphabetic\_encrypt(plaintext, keyword):

    cipher\_map = generate\_cipher\_alphabet(keyword)

    return "".join(cipher\_map.get(c, c) for c in plaintext.upper())

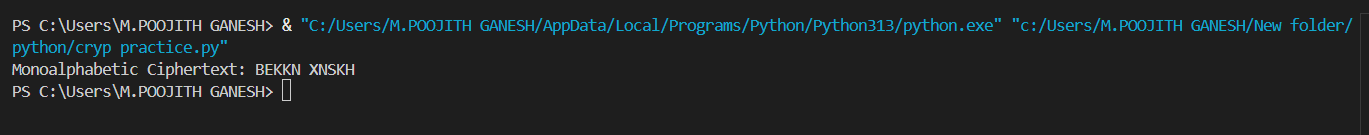
# Example usage

keyword = "CIPHER"

plaintext = "HELLO WORLD"

ciphertext = monoalphabetic\_encrypt(plaintext, keyword)

print("Monoalphabetic Ciphertext:", ciphertext)



**6.**

import numpy as np

# Convert text to number indices

def text\_to\_numbers(text):

return [ord(c) - ord('A') for c in text.upper().replace(" ", "")]

# Convert number indices back to text

def numbers\_to\_text(numbers):

return "".join(chr(n % 26 + ord('A')) for n in numbers)

# Encrypt using Hill cipher

def hill\_encrypt(plaintext, key\_matrix):

numbers = text\_to\_numbers(plaintext)

numbers += [0] \* (len(numbers) % 2) # Padding for even length

numbers = np.array(numbers).reshape(-1, 2)

encrypted = (np.dot(numbers, key\_matrix) % 26).flatten()

return numbers\_to\_text(encrypted)

# Example usage

key\_matrix = np.array([[9, 4], [5, 7]]) # Given key

plaintext = "MEETMEATUSUALPLACE"

ciphertext = hill\_encrypt(plaintext, key\_matrix)

print("Hill Ciphertext:", ciphertext)

