**Practical 1**

**AIM: Implement Caesar cipher encryption-decryption.**

Caesar cipher is one of the simplest and oldest method of encrypting message.

It was developed by Julius Caesar to protect military communication.

This technique involves shifting the letter of alphabet by fix number. which is known as “Shift/Key”.

It’s simplest type of substitution Cipher. In which each letter of given text is replaced by a shift or key position alphabet.

**CODE:**

def caesar\_cipher\_encrypt(msg, shift):

ciphertext = ""

for char in msg:

if char.isalpha():

if char.isupper(): #Checks if the character is alphabetic

shifted\_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))

else:

shifted\_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))

ciphertext += shifted\_char

else:

ciphertext += char

return ciphertext

def caesar\_cipher\_decrypt(ciphertext, shift):

msg = ""

for char in ciphertext:

if char.isalpha():

if char.isupper():

shifted\_char = chr((ord(char) - ord('A') - shift) % 26 + ord('A'))

else:

shifted\_char = chr((ord(char) - ord('a') - shift) % 26 + ord('a'))

msg += shifted\_char

else:

msg += char

return msg

msg = input("Enter the message: ")

shift = int(input("Enter the shift value: "))

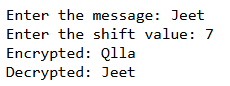
encrypted\_text = caesar\_cipher\_encrypt(msg, shift)

print("Encrypted:", encrypted\_text)

decrypted\_text = caesar\_cipher\_decrypt(encrypted\_text, shift)

print("Decrypted:", decrypted\_text)

**OUTPUT:**

****

**Practical 2**

**AIM: Implement Monoalphabetic cipher encryption-decryption.**

Monoalphabetic cipher is substitution technique in which a single alphabet is used for message.

It provides protection from brute force attack.

In Monoalphabetic cipher the mapping is done randomly not in uniform format.

**CODE:**

import numpy as np

import random

import string

def generate\_monoalphabetic\_key():

"""Generate a random Monoalphabetic cipher key."""

letters = list(string.ascii\_uppercase)

key = {}

for char in string.ascii\_uppercase:

random\_char = random.choice(letters)

key[char] = random\_char

letters.remove(random\_char) # Remove selected character to ensure unique mapping

return key

def encrypt\_monoalphabetic(message, key):

"""Encrypt a message using a Monoalphabetic cipher."""

encrypted\_message = []

capitalization\_info = []

for char in message:

if char.upper() in key:

encrypted\_char = key[char.upper()]

encrypted\_message.append(encrypted\_char)

capitalization\_info.append(char.isupper())

else:

encrypted\_message.append(char) # if character is not in the key, add it as-is

capitalization\_info.append(False) # mark as non-alphabetic or lowercase

return ''.join(encrypted\_message), capitalization\_info

def decrypt\_monoalphabetic(encrypted\_message, capitalization\_info, key):

"""Decrypt a message encrypted with a Monoalphabetic cipher."""

decrypted\_message = []

reverse\_key = {v: k for k, v in key.items()} # create reverse key for decryption

for i, char in enumerate(encrypted\_message):

if char.upper() in reverse\_key:

decrypted\_char = reverse\_key[char.upper()]

if capitalization\_info[i]:

decrypted\_char = decrypted\_char.upper()

else:

decrypted\_char = decrypted\_char.lower()

decrypted\_message.append(decrypted\_char)

else:

decrypted\_message.append(char) # if character is not in the key, add it as-is

return ''.join(decrypted\_message)

def analyze\_frequency(message):

"""Analyze the frequency of characters in a message."""

frequency = np.zeros((26,), dtype=int)

for char in message.upper():

if char.isalpha():

frequency[ord(char) - ord('A')] += 1

return frequency

# Generate a random Monoalphabetic key

key = generate\_monoalphabetic\_key()

print("Generated Monoalphabetic Key:")

print(key)

# Encrypt a message

message = input("Enter the message: ")

encrypted\_message, capitalization\_info = encrypt\_monoalphabetic(message, key)

print("Original Message:", message)

print("Encrypted Message:", encrypted\_message)

# Decrypt the message

decrypted\_message = decrypt\_monoalphabetic(encrypted\_message, capitalization\_info, key)

print("Decrypted Message:", decrypted\_message)

# Analyze the frequency of characters in the original and encrypted messages

original\_frequency = analyze\_frequency(message)

encrypted\_frequency = analyze\_frequency(encrypted\_message)

**OUTPUT:**

