# Name:sagili vani

**DAY 1**

**Q1**: Write a python program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet for k in the range 1 through 25

**PROGRAM:**

def caesar\_cipher(message, shift):

cipher = ''

for char in message:

if char.isalpha():

char\_code = ord(char) - shift

if char.isupper():

if char\_code > ord('Z'):

char\_code -= 26

elif char\_code < ord('A'):

char\_code += 26

elif char.islower():

if char\_code > ord('z'):

char\_code -= 26

elif char\_code < ord('a'):

char\_code += 26

cipher += chr(char\_code)

else:

cipher += char

return cipher

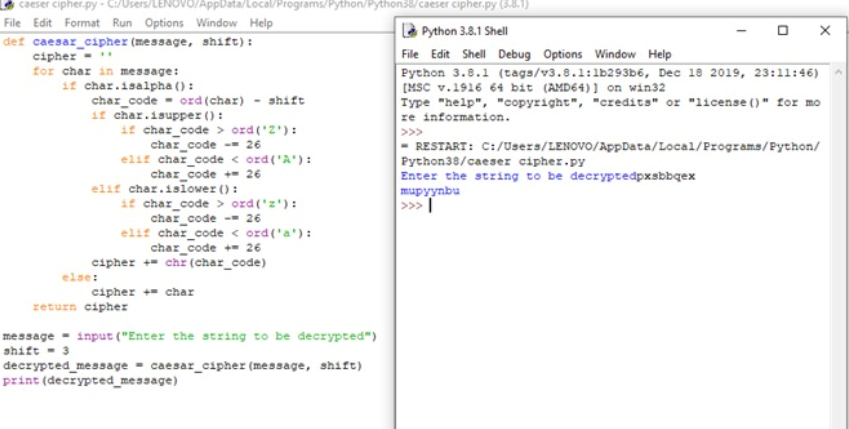
message = input("Enter the string to be decrypted")

shift = 3

decrypted\_message = caesar\_cipher(message, shift)

print(decrypted\_message)

**OUTPUT**



**Q2:** Write a python program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.

**PROGRAM:**

import string

# Define the mapping for the cipher

cipher\_map = {'a': 'q', 'b': 'w', 'c': 'e', 'd': 'r', 'e': 't',

'f': 'y', 'g': 'u', 'h': 'i', 'i': 'o', 'j': 'p',

'k': 'a', 'l': 's', 'm': 'd', 'n': 'f', 'o': 'g',

'p': 'h', 'q': 'j', 'r': 'k', 's': 'l', 't': 'z',

'u': 'x', 'v': 'c', 'w': 'v', 'x': 'b', 'y': 'n', 'z': 'm'}

# Define the reverse mapping for decryption

decipher\_map = {v: k for k, v in cipher\_map.items()}

def encrypt(message):

"""Encrypts the given message using the cipher map."""

# Convert message to lowercase

message = message.lower()

# Initialize the encrypted message

encrypted\_message = ''

# Encrypt each character in the message

for char in message:

if char in string.ascii\_lowercase:

encrypted\_char = cipher\_map[char]

else:

encrypted\_char = char

encrypted\_message += encrypted\_char

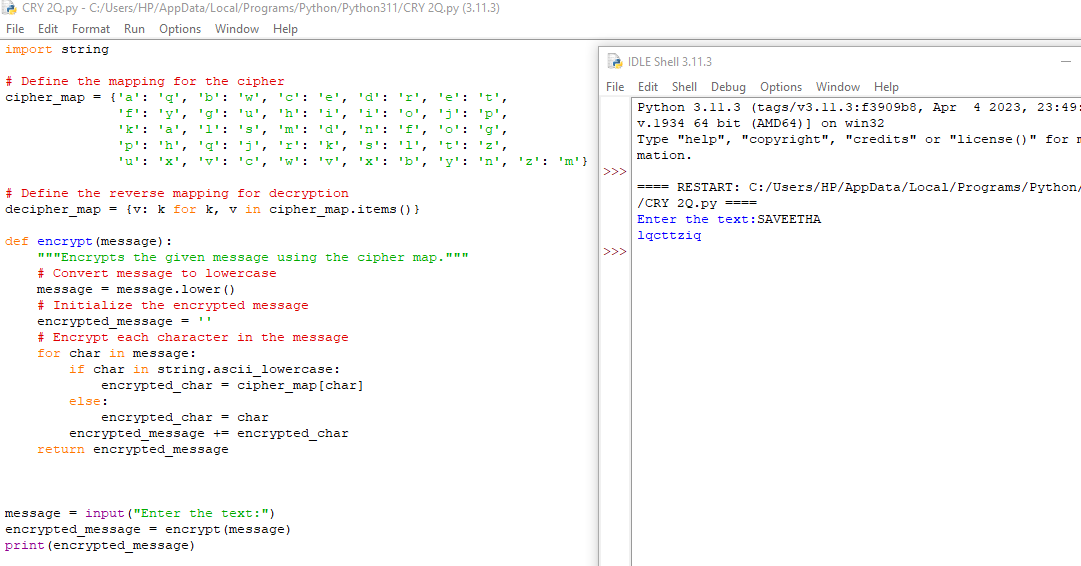
return encrypted\_message

message = input("Enter the text:")

encrypted\_message = encrypt(message)

print(encrypted\_message)

**OUTPUT:**



**Q3:** Write a Python program for the Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed on a keyword. Plaintext has encrypted two letters at a time using this matrix.

**PROGRAM:**

def toLowerCase(text):

return text.lower()

def removeSpaces(text):

newText = ""

for i in text:

if i == " ":

continue

else:

newText = newText + i

return newText

def Diagraph(text):

Diagraph = []

group = 0

for i in range(2, len(text), 2):

Diagraph.append(text[group:i])

group = i

Diagraph.append(text[group:])

return Diagraph

def FillerLetter(text):

k = len(text)

if k % 2 == 0:

for i in range(0, k, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

else:

for i in range(0, k-1, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

return new\_word

list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',

'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

def generateKeyTable(word, list1):

key\_letters = []

for i in word:

if i not in key\_letters:

key\_letters.append(i)

compElements = []

for i in key\_letters:

if i not in compElements:

compElements.append(i)

for i in list1:

if i not in compElements:

compElements.append(i)

matrix = []

while compElements != []:

matrix.append(compElements[:5])

compElements = compElements[5:]

return matrix

def search(mat, element):

for i in range(5):

for j in range(5):

if(mat[i][j] == element):

return i, j

def encrypt\_RowRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1c == 4:

char1 = matr[e1r][0]

else:

char1 = matr[e1r][e1c+1]

char2 = ''

if e2c == 4:

char2 = matr[e2r][0]

else:

char2 = matr[e2r][e2c+1]

return char1, char2

def encrypt\_ColumnRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1r == 4:

char1 = matr[0][e1c]

else:

char1 = matr[e1r+1][e1c]

char2 = ''

if e2r == 4:

char2 = matr[0][e2c]

else:

char2 = matr[e2r+1][e2c]

return char1, char2

def encrypt\_RectangleRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

char1 = matr[e1r][e2c]

char2 = ''

char2 = matr[e2r][e1c]

return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):

CipherText = []

for i in range(0, len(plainList)):

c1 = 0

c2 = 0

ele1\_x, ele1\_y = search(Matrix, plainList[i][0])

ele2\_x, ele2\_y = search(Matrix, plainList[i][1])

if ele1\_x == ele2\_x:

c1, c2 = encrypt\_RowRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

elif ele1\_y == ele2\_y:

c1, c2 = encrypt\_ColumnRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

else:

c1, c2 = encrypt\_RectangleRule(

Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

cipher = c1 + c2

CipherText.append(cipher)

return CipherText

text\_Plain = 'hello'

text\_Plain = removeSpaces(toLowerCase(text\_Plain))

PlainTextList = Diagraph(FillerLetter(text\_Plain))

if len(PlainTextList[-1]) != 2:

PlainTextList[-1] = PlainTextList[-1]+'z'

key = "network"

print("Key text:", key)

key = toLowerCase(key)

Matrix = generateKeyTable(key, list1)

print("Plain Text:", text\_Plain)

CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""

for i in CipherList:

CipherText += i

print("CipherText:", CipherText)

**OUTPUT:**



**Q4:** Write a Python program for the polyalphabetic substitution cypher uses a separate monoalphabetic substitution cypher for each successive letter of plaintext, depending on a key

**PROGRAM:**

import string

def poly\_sub\_cipher(plaintext, key):

plaintext = plaintext.lower()

key = key.lower()

alphabet = string.ascii\_lowercase

ciphertext = ''

for i, char in enumerate(plaintext):

key\_char = key[i % len(key)]

shift = alphabet.index(key\_char)

shifted\_alphabet = alphabet[shift:] + alphabet[:shift]

if char in alphabet:

ciphertext += shifted\_alphabet[alphabet.index(char)]

else:

ciphertext += char

return ciphertext

plaintext = 'GEEKSFORGEEKS'

key = 'AYUSH'

ciphertext = poly\_sub\_cipher(plaintext, key)

print(ciphertext)

**OUTPUT:**

